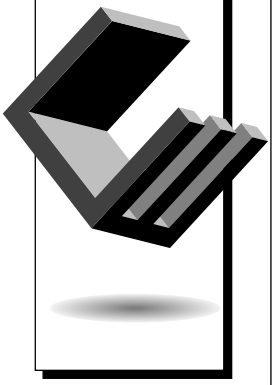
ComponentWorks



Getting Results with ComponentWorks™

June 1997 Edition
Part Number 321170B-01

© Copyright 1996, 1997 National Instruments Corporation. All rights reserved.

Internet Support

E-mail: `support@natinst.com`
`info@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 (U.S.)

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,
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 5734815, 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, U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 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

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

Trademarks

ComponentWorks™, LabVIEW®, National Instruments®, NI-DAQ®, DAQ-STC™, natinst.com™, and SCXI™ are trademarks of National Instruments Corporation.

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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

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

Contents

About This Manual

Organization of This Manual xi
Conventions Used in This Manual xiii
Customer Communication xiv

Chapter 1
Introduction to ComponentWorks

What is ComponentWorks? 1-1
Installing ComponentWorks 1-2
System Requirements 1-2
Installation Instructions 1-3
Installing the ComponentWorks ActiveX Control 1-3
Installing From Floppy Disks 1-3
Installing the Instrument Drivers DLLs 1-4
Installed Files 1-4
About the ComponentWorks Controls 1-5
Properties, Methods, and Events 1-5
Object Hierarchy 1-6
Collection Objects 1-7
Setting the Properties of an ActiveX Control 1-8
Using Property Sheets 1-8
Changing Properties Programmatically 1-10
Item Method 1-11
Working with Control Methods 1-12
Developing Event Handler Routines 1-12
Using the Analysis Library and Instrument Driver DLLs 1-13
The Help File—Learning the Properties, Methods, and Events 1-13

Chapter 2
Building ComponentWorks Applications with Visual Basic

Developing Visual Basic Applications 2-1
Loading the ComponentWorks Controls into the Toolbox 2-1
Building the User Interface Using ComponentWorks 2-2
Using Property Sheets 2-3

Using Your Program to Edit Properties 2-4
Working with Control Methods 2-5
Developing Control Event Routines 2-6
Using the ComponentWorks Instrument Driver DLLs in Visual Basic 2-7
Object Browser—Building Your Code in Visual Basic 2-8
Pasting Code into Your Program 2-11
Learning to Use Specific ComponentWorks Controls 2-11

Chapter 3
Building ComponentWorks Applications with Visual C++

Developing Visual C++ Applications 3-1
Creating Your Application 3-1
Adding ComponentWorks Controls to the Visual C++ Controls Toolbar 3-3
Building the User Interface Using ComponentWorks Controls 3-4
Programming with the ComponentWorks Controls 3-5
Using Properties 3-7
Using Methods 3-9
Using Events 3-10
Learning to Use Specific ComponentWorks Controls 3-11

Chapter 4
Building ComponentWorks Applications with Delphi

Developing Delphi Applications 4-1
Loading the ComponentWorks Controls into the Component Palette 4-1
Building the User Interface 4-3
Placing Controls 4-3
Using Property Sheets 4-4
Using Your Program to Edit Properties 4-5
Programming with ComponentWorks 4-6
Using Methods 4-7
Using Events 4-8
Learning to Use Specific ComponentWorks Controls 4-8

Chapter 5
Using the Graphical User Interface Controls

What are the GUI Controls? 5-2
Object Hierarchy and Common Objects 5-3
The Knob and Slide Controls 5-4
Knob and Slide Object 5-5
Pointers Collection 5-5
Pointer Object 5-5

Axis Object 5-6
Ticks and Labels Objects 5-6
ValuePairs Collection 5-7
ValuePair Object 5-7
Statistics Object 5-8
Events 5-8
The Numeric Edit Box Control 5-9
Events 5-9
Tutorial: Knob, Slide, and Numeric Edit Box Controls 5-10
Designing the Form 5-10
Developing the Program Code 5-11
Testing Your Program 5-13
The Button Control 5-14
Events 5-15
The Graph Control 5-15
Graph Object 5-17
Plot Methods 5-17
Chart Methods 5-18
Plots Collection 5-19
Plot Object 5-19
PlotTemplate Object 5-20
Cursors Collection 5-21
Cursor Object 5-21
Axes Collection 5-22
Axis Object 5-23
Events 5-23
Panning and Zooming 5-24
Tutorial: Graph and Button Controls 5-25
Designing the Form 5-25
Developing the Code 5-27
Testing Your Program 5-29

Chapter 6
Using the Data Acquisition Controls

Data Acquisition Configuration 6-2
Object Hierarchy and Common Properties 6-2
Device, DeviceName, and DeviceType 6-3
Channel Strings 6-3
SCXI Channel Strings 6-4
ExceptionOnError and ErrorEventMask 6-5
AIPoint Control—Single Point Analog Input 6-6
AIPoint Object 6-6

Channels Collection 6-7
Channel Object 6-8
ChannelClock Object 6-8
AI Control—Waveform Analog Input 6-9
AI Object 6-10
Methods and Events 6-10
Asynchronous Acquisition 6-10
Synchronous Acquisition 6-11
Error Handling 6-12
ScanClock and ChannelClock Objects 6-12
StartCondition, PauseCondition and StopCondition Objects 6-13
Tutorial: Using the AIPoint and AI DAQ controls 6-14
Designing the Form 6-15
Setting the DAQ Properties 6-16
Developing the Code 6-17
Testing Your Program 6-19
AOPoint Control—Single Point Analog Output 6-19
AOPoint Object 6-20
Methods 6-20
AO Control—Waveform Analog Output 6-21
AO Object 6-22
Methods and Events 6-23
UpdateClock and IntervalClock Objects 6-24
StartCondition Object 6-25
Tutorial: Using the AOPoint control 6-26
Designing the Form 6-26
Developing the Code 6-27
Testing Your Program 6-29
Digital Controls and Hardware 6-30
DIO Control—Single Point Digital Input and Output 6-30
DIO Object 6-31
Ports Collection and Port Object 6-32
Lines Collection and Line Object 6-33
Common Properties and Methods 6-34
DI Control—Buffered Waveform Digital Input 6-36
DI Object 6-36
UpdateClock Object 6-37
Methods and Events 6-38
DO Control—Buffered Waveform Digital Output 6-40
DO Object 6-40
UpdateClock Object 6-41
Methods and Events 6-42
Tutorial: Using the DIO control 6-44

Designing the Form 6-44
Developing the Code 6-45
Testing Your Program 6-47
DAQTools—Data Acquisition Utility Functions 6-48
Using DAQ Tools functions 6-49
Counter/Timer Hardware 6-50
Counter Control—Counting and Measurement Operations 6-50
Counter Object 6-51
Methods and Events 6-53
Buffered Measurements 6-54
Pulse Control—Digital Pulse and Pulsetrain Generation 6-55
Pulse Object 6-56
Methods 6-58
FSK and ETS Pulse Generation 6-59
Tutorial: Using the Counter and Pulse controls 6-60
Designing the Form 6-60
Developing the Code 6-62
Testing Your Program 6-65

Chapter 7
Using the Analysis Controls and Functions

What are the Analysis Controls? 7-1
Analysis Library Versions 7-2
Controls 7-12
Analysis Function Descriptions 7-13
Error Messages 7-13
Tutorial: Using Simple Statistics Functions 7-13
Designing the Form 7-14
Developing the Program Code 7-15
Testing Your Program 7-17

Chapter 8
Building Advanced Applications

Using Advanced ComponentWorks Features 8-1
A Virtual Oscilloscope 8-2
Data Acquisition Stop Condition Modes 8-3
Data Acquisition Pretriggering 8-3
User Interface Value Pairs 8-4
Virtual Spectrum Meter 8-5
DSP Analysis Library 8-7
Cursors 8-8
Graph Track Mode 8-10

A Virtual Data Logger 8-11
Multiple Graph Axes 8-12
Graph Axes Formats 8-13
File Input/Output 8-14
Adding Testing and Debugging to Your Application 8-14
Error Checking 8-14
Exceptions 8-15
Return Codes 8-17
Error and Warning Events 8-18
GetErrorText Function 8-19
Debugging 8-20
Debug Print 8-20
Breakpoint 8-20
Watch Window 8-21
Single Step, Step Into and Step Over 8-21

Appendix A
Common Questions

ComponentWorks Common Questions A-1

Appendix B
Error Codes

Appendix C
Customer Communication

Glossary

Index

Tables

Table 7-1. Analysis Control Function Tree 7-3
Table B-1. Data Acquisition Control Error Codes B-1
Table B-2. Analysis Error Codes B-22
Table B-3. General ComponentWorks Error Codes B-27

About This Manual

The *Getting Results with ComponentWorks* manual contains the information you need to get started with the ComponentWorks software package. ComponentWorks adds the instrumentation-specific tools for acquiring, analyzing, and displaying data in Visual Basic, Visual C, and Delphi.

This manual contains step-by-step instructions for building applications with ComponentWorks. You can then 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 should already be familiar with one of the supported programming environments and Windows 95 or Windows NT.

Organization of This Manual

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

- Chapter 1, *Introduction to ComponentWorks*, contains an overview of ComponentWorks, lists the ComponentWorks system requirements, describes how to install the software, and explains the basics of ActiveX controls.
- Chapter 2, *Building ComponentWorks Applications with Visual Basic*, contains an overview of using ComponentWorks controls with Visual Basic. This chapter explains how to insert the controls into the Visual Basic environment, set their properties, and use their methods and events.
- Chapter 3, *Building ComponentWorks Applications with Visual C++*, contains an overview of using ComponentWorks controls with Visual C++. This chapter explains how to insert the controls into the Visual C++ environment and create the necessary wrapper classes, as well as how to use the MFC AppWizard with ComponentWorks controls.

- Chapter 4, *Building ComponentWorks Applications with Delphi*, contains an overview of using ComponentWorks controls with Delphi. This chapter explains how to insert the controls into the Delphi environment, set their properties, and use their methods and events.
- Chapter 5, *Using the Graphical User Interface Controls*, shows you how to use the graphical user interface (GUI) controls to customize your application's interface to suit your needs.
- Chapter 6, *Using the Data Acquisition Controls*, shows how to use the ComponentWorks data acquisition (DAQ) controls in your application to perform input and output operations with your National Instruments DAQ hardware.
- Chapter 7, *Using the Analysis Library*, shows you how to use the ComponentWorks analysis controls and functions. The analysis functions can be used alone or with other controls to perform data analysis, manipulation, and simulation.
- Chapter 8, *Building Advanced Applications*, discusses how to build applications using more advanced features of ComponentWorks, including advanced data acquisition techniques, the DSP Analysis Library, and advanced GUI controls. It also discusses error tracking, error checking, and debugging techniques.
- Appendix A, *Common Questions*, contains a list of answers to frequently asked questions. It contains general ComponentWorks questions as well as specific data acquisition, graphical user interface, and Analysis Library questions.
- Appendix B, *Error Codes*, lists the error codes returned by the ComponentWorks DAQ controls and Analysis Library functions. It also lists some general ComponentWorks error codes.
- 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 acronyms, abbreviations, 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:

bold	Bold text denotes a parameter, menu name, palette name, menu item, return value, function panel item, or dialog box button or option.
italic	Italic text denotes mathematical variables, emphasis, a cross reference, or an introduction to a key concept.
bold italic	Bold italic text denotes an activity objective, note, caution, or warning.
`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 also appear in this font. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, variables, filenames, and extensions, and for statements and comments taken from program code.
<>	Angle brackets enclose the name of a key on the keyboard—for example, <PageDown>.
-	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>.
<Control>	Key names are capitalized.
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options»Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Fonts** option from the last dialog box.
paths	Paths in this manual are denoted using backslashes (\) to separate drive names, directories, and files, as in `C:\dir1name\dir2name\file`.
	Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

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.

Introduction to ComponentWorks

Chapter 1

This chapter contains an overview of ComponentWorks, lists the ComponentWorks system requirements, describes how to install the software, and explains the basics of ActiveX controls.

What is ComponentWorks?

ComponentWorks is a collection of ActiveX controls and DLLs (Dynamic Link Libraries) for acquiring, analyzing, and presenting data within any compatible ActiveX control container. ActiveX controls are also 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 and the routines of the DLLs. Access this information by opening the online ComponentWorks Reference Manual in the ComponentWorks folder.

With ComponentWorks, you can easily develop complex custom user interfaces to display your data, control your National Instruments Data Acquisition (DAQ) boards, and analyze data you acquired or received from some other source. The ComponentWorks package contains the following components:

- User Interface Controls—32-bit ActiveX controls for presenting your data in a technical format. These controls include a graph/strip chart control, sliders, thermometers, tanks, knobs, gauges, meters, LEDs, and switches.
- DAQ Controls—32-bit ActiveX controls for analog, digital I/O, and counter/timer I/O operations using National Instruments DAQ products.

- Analysis Library Controls—Functions for statistics, advanced signal processing, windowing, filters, curve-fitting, vector and matrix algebra routines, probability, and array manipulations. These functions are packaged in 32-bit ActiveX controls. Each ComponentWorks package (Base, Standard, and Full Development System) contains different options for analysis. The Base package includes functions for basic statistics and array operations; the Standard Development System includes additional Digital Signal Processing (DSP)functions for signal processing, windowing, and filtering operations; and the Full Development System adds advanced statistics and probability functions.
- Instrument Drivers (shipped only with the Full Development System)—32-bit DLLs for controlling common GPIB instruments with high-level instrument control routines.

The ComponentWorks ActiveX controls and DLLs 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 and DLLs in many other applications that support ActiveX controls. Visual C++, Access, and Delphi are just a few of the development environments outside of Visual Basic that support ActiveX controls and DLLs.

Installing ComponentWorks

You must install ComponentWorks on your computer before you can get started. The ComponentWorks setup program does this for you in a process that lasts approximately five minutes.

System Requirements

To use the ComponentWorks ActiveX controls and Analysis Library, you must have the following:

- Microsoft Windows 95 or Windows NT (4.0 required for DAQ controls) operating system
- Personal computer using at least a 33 MHz 80486 or higher microprocessor (National Instruments recommends a 66 MHz 80486 or higher microprocessor)
- VGA resolution (or higher) video adapter

- ActiveX custom control container such as Visual Basic (32-bit version), Visual C++, Delphi (32-bit version)
- NI-DAQ 5.0 or later for Windows 95 or Windows NT (if you are using DAQ controls)
- Minimum of 8 MB of memory
- Minimum of 10 MB of free hard disk space
- Microsoft-compatible mouse

Installation Instructions

Complete the following steps to install ComponentWorks:

☞ **Note:** ***If you are installing ComponentWorks on a Windows NT system, you must be logged in with Administrator privileges to complete the installation.***

Installing the ComponentWorks ActiveX Control

1. Make sure that your computer and monitor are turned on and that you have installed Windows 95 or Windows NT.
2. Insert the ComponentWorks CD into the CD drive of your computer. From the CD startup screen, click on **Install ComponentWorks 1.1**. If the CD startup screen does not appear, use the Windows Explorer or File Manager to run the `SETUP.EXE` program in the `\CWorks\disks\disk1` directory on the CD.

Installing From Floppy Disks

If your computer using ComponentWorks does not have a CD drive, follow these instructions for installing the software:

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

Installing the Instrument Drivers DLLs

If you have purchased the Instrument Driver DLL library or the ComponentWorks Full Development System you will have received a separate CD containing the instrument driver DLLs. Follow these instructions for their installation:

1. Run the `setup.exe` program from the Instrument Driver CD.
2. Following the instructions in the installation program, select whether to install the 16- or 32-bit versions of the instrument drivers. If you would like both versions, you will need to run the installation twice. To do this, move the installed DLL files to a different directory after the first installation, so that they will not be copied over during the second installation.
3. Select the desired support files and continue to the driver selection.
4. When you select the actual drivers to install, click to the left of the driver name (in the white space, not on the name itself) so that a check mark appears next to the selected drivers. In this dialog, also select the pathname for the installation. The default is `\InstrDLL`.
5. Follow the installer instructions to complete the installation.
6. To use the instrument drivers, copy the `INSTRSUP.DLL` and/or `LWSUPP.DLL` files (found in the `\InstrDLL\System` directory) to the `\System` subdirectory under your Windows 95 directory or to the `\System32` subdirectory under your WindowsNT directory.

Installed Files

The ComponentWorks setup program installs the following groups of files on your hard disk:

- ActiveX Controls and Documentation

  Directory: `\Windows\system(32)\`

  Files: `cwdaq.ocx`, `cwui.ocx`, `cwanalysis.ocx`, `cwref.hlp`

- Example Programs and Applications

  Directory: `\ComponentWorks\samples\…`

- Tutorial Programs

  Directory: `\ComponentWorks\tutorial\…`

- Miscellaneous Files

  Directory: `\ComponentWorks\`

- Instrument Driver Files

  Directory: `\InstrDLL\`.

☞ **Note:** ***You select the*** `\ComponentWorks\…` ***and*** `\InstrDLL\…` ***directories during installation.***

About the ComponentWorks Controls

Before learning how to use ComponentWorks, you should be familiar with using ActiveX controls. This section outlines some background information about ActiveX controls, in particular the ComponentWorks controls. If you are not already familiar with the concepts outlined in this section, make sure you understand them before continuing.
You may also want to refer to your programming environment documentation for more information on using Active X (OLE) controls in your particular environment.

Properties, Methods, and Events

ActiveX controls consists of three different parts (properties, events, and methods) 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 controls. Methods are called with respect to a particular control and usually have some affect on the control itself. The operation of most methods is also affected by the current property values of the control.

Events are messages generated by a control in response to some particular occurrence. The events pass to the control container application to execute a particular section in the program (event handler).

For example, the ComponentWorks Graph control has a wide variety of properties that determine how the graph looks and operates. You can set properties for color, axes, scaling, tick marks, cursors, and labels, to name a few, to customize the graph appearance and behavior.

The Graph control also has a series of high-level methods, or functions, that you can invoke to set several properties at once and to perform a

particular operation. For example, you can use the `PlotY` method to pass an array of data to the Graph control.

The Graph control generates events when particular operations take place. For example, when you drag a cursor on the graph, the control passes an event to your program so you can respond to cursor movements. For example, you might want to retrieve the X- and Y-coordinate positions of a cursor as it is being dragged and display the coordinates in text boxes.

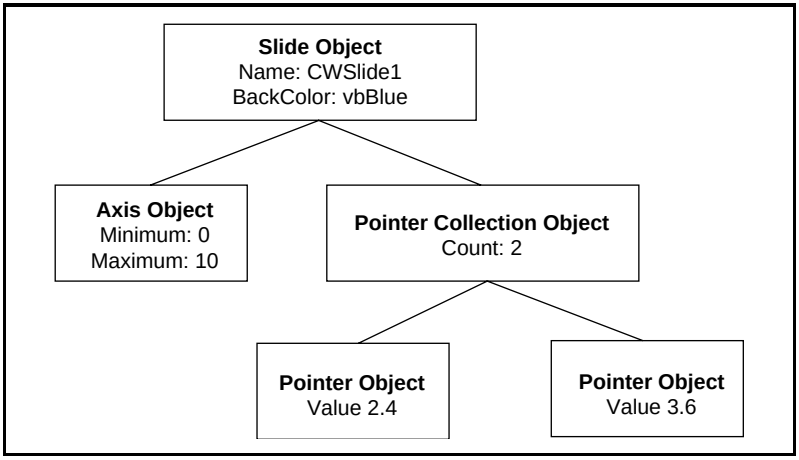
☞ **Note:** ***Use the ComponentWorks online reference for specific information about the properties, methods, and events of the ActiveX controls and the routines of the instrument driver DLLs.***

Object Hierarchy

As described in the previous section, each ActiveX control has properties, methods and events. These three parts are stored in a *software object*, which is the piece of software that makes up the ActiveX control and contains all its parts. Certain ActiveX controls can be become very complex, containing many different parts (properties). For this reason, complex 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 some specific parts (properties) and functionality (methods and event) of the ActiveX control. The relationship between the different objects of a control are maintained in an object hierarchy. At the top of the hierarchy is the actual control itself. This top-level object contains its own properties, methods and events. In addition, the object contains references to other objects that define specific parts of the control. Such references are usually also referred to as properties of the control/object because they describe the state of the object. The number of levels in the hierarchy is not limited, and objects below the top-level object may contain references to other objects of their own.

Another advantage of subdividing controls is the re-use of different objects between different controls. One object may be used at different places in the same object hierarchy, or may be used in several different controls/object hierarchies.

The following diagram shows an example of an object hierarchy using the ComponentWorks slide control:



The Slide 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 from the Slide object. The Axis object contains all the information about the axis used on the slide and has properties such as Maximum and Minimum. The Pointers Collection object contains several Pointer objects of its own, each describing one pointer on the Slide control. Each Pointer object has properties, one of which is `Value`, while the Pointers Collection object has the property `Count`. The Pointers Collection object is a special type of object referred to as a collection, which is described in the *Collection Objects* section.

Collection Objects

In certain cases one object contains several objects of the same type. For example, a Graph object contains several Axis objects, each representing one of the axes on the graph. Additionally, the number of objects in the group of objects may not be defined, and may change while the program is running (that is, you can add or remove axes 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 which contains or stores a varying number of objects of the same type. It can also be regarded 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, a collection of Plot objects as

Plots, and a collection of Axis objects as Axes. In the software, the terms object and collection are removed so that only the type names Pointer and Pointers are used.

Each collection object contains an `Item` method you can use to access any particular object stored in the collection. This method and how to access properties and objects in the object hierarchy are explained in the *Setting the Properties of an ActiveX Control* section.

Setting the Properties of an ActiveX Control

You can set the properties of an ActiveX control from its property sheets or from within your program. Information specific to your programming environment on how to perform these operations is documented in the following chapters. This section discusses the general issues regarding these operations.

Using Property Sheets

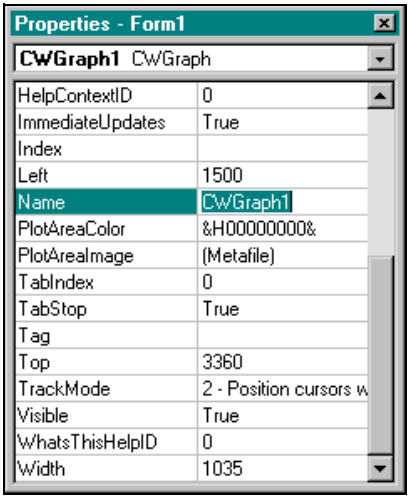
You use property sheets to set the default property values for each ActiveX control. Property sheets are common throughout the Windows 95 and Windows NT interface. When you want to change the appearance or options of a particular object, right-click on the object and select **Properties**. A property sheet 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 drop the control onto a form in your programming environment, right-click on the control and select **Properties…**. You can then set the properties, customizing the appearance and operation of the control.

Use the property sheets to set the default property values for each 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. The property values can also be changed from within your program, as shown in the *Changing Properties Programmatically* section later in this chapter.

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

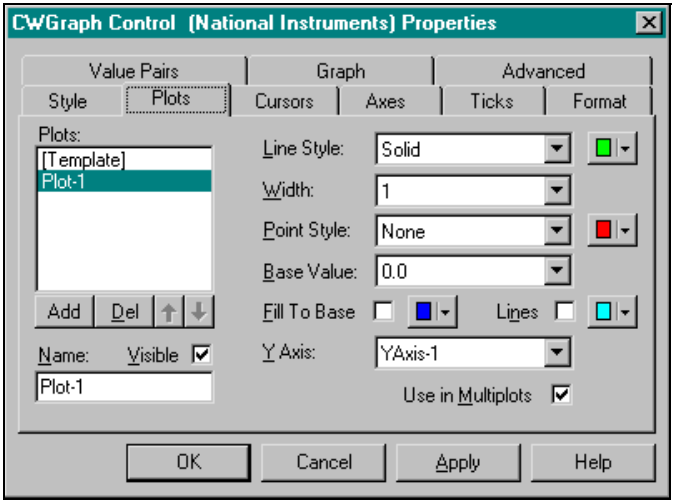
Your programming environment assigns default values for some of the basic properties, such as the control name and the tab order. You must edit these properties by using the default property sheet.

The following illustration shows the Visual Basic default property sheet for the CWGraph control:



The second property sheet is called the custom property page. The layout and functionality of the custom property sheets are defined by the developer of the ActiveX control and varies among different controls.

The following illustration shows the custom property sheet for the CWGraph control:



Changing Properties Programmatically

You can also set or read the properties of your controls programmatically. For example, if you want to change the state of an LED control during program execution, change the *Value* property from `True` to `False`, or `False` to `True`. The exact syntax for reading and writing property values depends on your programming language, so consult the appropriate chapter for using your programming environment. For illustration purposes we will use the Visual Basic syntax here, which is similar to most other implementations.

Each control you create in your program will have a name (like a variable name) which is used to reference the control in your program. Setting the value of a property on a top level object is straightforward.

```
name.property = new_value
```

For example, to change the value property of an LED control to `off`, use the following line of code, where `CWButton1` is the default name of the button/LED control:

```
CWButton1.Value = False
```

To access properties of sub-objects referenced by the top level object, follow the control name with the name of the sub-object followed by the

property name. For example, consider the following code for the ComponentWorks data acquisition analog input (CWAI) control:

```
CWAI1.ScanClock.Frequency = 10000
```

In the above code, `ScanClock` is a property/object of the CWAI control. `Frequency` is a property of the `ScanClock` object. As an object of the CWAI control, `ScanClock` has several additional properties.

You can retrieve the value of control properties from your program in the same way. For example, to print the value of the LED control listed above, use the following line of code:

```
Print CWButton1.Value
```

To display the frequency used by the CWAI control in a Visual Basic text box use the following code.

```
Text1.Text = CWAI1.ScanClock.ActualFrequency
```

Item Method

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

```
CWSlide1.Pointers.Item(2).Value = 5.0
```

The term `CWSlide1.Pointers.Item(2)` refers to the second Pointer object in the Pointers collection of the Slide object. The parameter of the `Item` method is either an integer representing the (one-based) index of the object in the collection or a string with the name of one of the objects in the collection.

```
CWSlide1.Pointers.Item("TemperaturePointer")
```

Because the `Item` method is the most commonly used method on a collection it is referred to as the default method. Therefore, in *some* programming environments, `.Item` can be left out of the code. So,

```
CWSlide1.Pointers(2).Value = 5.0
```

is programmatically the same as

```
CWSlide1.Pointers.Item(2).Value = 5.0
```

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 you pass to the method, and return values that pass information back to your program.

For example, the `PlotY` method for the ComponentWorks Graph control has a required parameter—the array of data to be plotted—that you must include when you call the method. If you want to plot the data returned from an Analog Input control, use the following line of code (the array `Voltages` is automatically generated by the CWAI control).

```
CWGraph1.PlotY Voltages
```

The `PlotY` method has additional parameters that are optional in *some* programming environments. For example, in addition to the first parameter representing the data to be plotted, you can pass a second parameter to represent the initial value for the X axis, a third parameter for an incremental change on the X axis corresponding to each data point, and a fourth parameter that determines how the graph should handle two-dimensional data:

```
CWGraph1.PlotY Voltages, 0.0, 1.0, True
```

Depending on your programming environment, the parameters may be enclosed in parentheses. Visual Basic does not use parentheses to pass parameters if the function or method is not assigned a return variable. The `AcquireData` method in the DAQ Analog Input control has the following form when used with a return variable `lErr`.

```
lErr = CWAI1.AcquireData(Voltages, BinaryCodes, 1#)
```

Developing Event Handler Routines

After you configure your controls on a form, you create event handler routines in your program to respond to events generated by the controls. For example, the DAQ Analog Input control has an `Acquired_Data` event that *fires* (occurs) when the acquired data is ready to be processed, based on the acquisition options you have configured in the control property pages.

You can configure the control to continuously collect 1,000 points of data from a particular channel at a rate of 1,000 points per second. Therefore, once every second, the data buffer is ready and the `Acquired_Data` event is fired. In your `Acquired_Data` event routine, you can write code to analyze the data buffer, plot it, or store it to disk.

To develop the event routine code, most programming environments generate a skeleton function to handle each event. Chapters 2, 3, and 4 of this manual outline how to generate these function skeletons to build your event handler routines. For example, the Visual Basic environment generates the following function skeleton into which you insert the functions to call when the AcquireData event occurs.

```
Private Sub CWAI1_AcquiredData(Voltages As Variant,
  BinaryCodes As Variant)

End Sub
```

In most cases, the event also returns some data to the event handler that can be used in your event handler routine. In the example of the `AcquireData` event above, this includes the `Voltages` and `BinaryCodes` arrays.

Using the Analysis Library and Instrument Driver DLLs

The ComponentWorks Analysis Library is packaged as a set of ActiveX controls, while the instrument drivers are packaged as 32-bit DLLs. You can add analysis functions to your project in the same way you add user interface or data acquisition controls. After adding the Analysis controls to your programming environment, use the analysis functions like any other method on a control. To use any specific function, place the appropriate analysis control on a form. Then, in your program, call the name of the control followed by the name of the analysis function:

```
MeanValue = CWStat1.Mean (Data)
```

Consult Chapter 7, *Using the Analysis Controls and Functions*, and the ComponentWorks online help for more information on the individual analysis functions and their use.

To use the instrument driver DLLs, you must add a reference to the DLL in your project. After you add the appropriate reference to your project, you can use the functions included in the DLLs to control your instruments easily.

The Help File—Learning the Properties, Methods, and Events

The ComponentWorks online help files contain detailed information on each control and its associated properties, methods, and events. Refer to ComponentWorks Help when you are using a control for the first time. Remember that many of the ComponentWorks controls share sub-objects, properties, and more, so when you learn how to use one control, you also learn how to use others. You can open the help file

from within most programming environments by clicking on the **Help** button in the custom property pages, or you can open it from the Windows environment by selecting the ComponentWorks Reference icon in the ComponentWorks folder/program group.

Some programming environments, compatible with ActiveX controls, have built-in mechanisms for detailing the available properties, methods and events for a particular control. Some of these include automatic links into the online help file. These additional tools are highlighted in the chapters of this manual discussing the individual programming environments.

Building ComponentWorks Applications with Visual Basic

Chapter 2

This chapter contains an overview of using the ComponentWorks controls with Visual Basic. At this point you should be familiar with the general structure of ActiveX controls described in Chapter 1, *Introduction to ComponentWorks*. This chapter explains how to insert the controls into the Visual Basic environment, set their properties, and use their methods and events. This chapter discusses how to perform these operations using ActiveX controls in general. The individual ComponentWorks controls are described later in this manual. This chapter also outlines Visual Basic features that simplify working with ActiveX controls.

Developing Visual Basic Applications

You start developing applications in Visual Basic 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 toolbox in Visual Basic contains all of the controls available for developing the form.

After you place each control on the form, the next step is to configure the properties of the control. This is done using the default and custom property pages.

Each control you place on a form has associated code (event handler routines) in your Visual Basic program that is automatically executed when that control is operated by the user. To create this code, double-click on the control and the Visual Basic code editor is opened to a default event handler routine.

Loading the ComponentWorks Controls into the Toolbox

Before you build an application using the ComponentWorks controls and libraries, you must add them to the Visual Basic toolbox. The ComponentWorks ActiveX controls are divided into three groups: user interface controls (`CWUI.OCX`), data acquisition controls (`CWDAQ.OCX`), and analysis library controls and functions

(CWANALYSIS.OCX). When you start a new project in Visual Basic, right-click on the toolbox and select **Custom Controls…**. The ComponentWorks controls should be listed in the **Available Controls** list starting with **National Instruments...**. Select the controls you want to use in your project. If the ComponentWorks controls are not in the list, select the desired control files from the \Windows\System(32) directory by pressing the **Browse** button.

If you plan to use the ComponentWorks controls in several projects, you can configure Visual Basic 4 to include the ComponentWorks controls in the default collection of controls in the toolbox. Do this by loading the project AUTO32LD.VBP from the Visual Basic directory. This file contains the default settings for your Visual Basic projects. Then, add the ComponentWorks controls to the toolbox and save the project. When you create a new project in Visual Basic, the ComponentWorks controls will always appear in the toolbox.

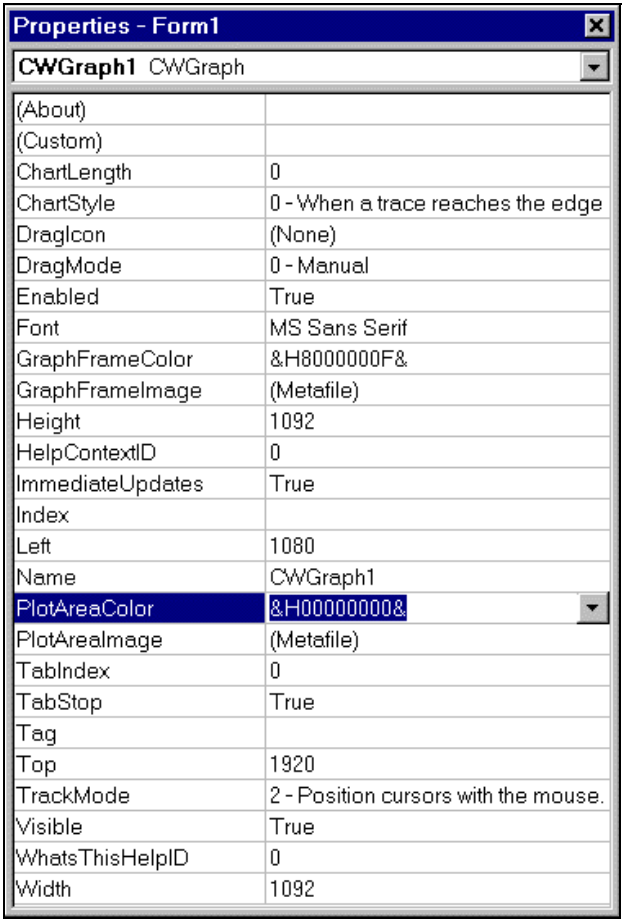
Building the User Interface Using ComponentWorks

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. Notice that the icons for the data acquisition (DAQ) and analysis controls cannot be resized and will not be visible at run time.

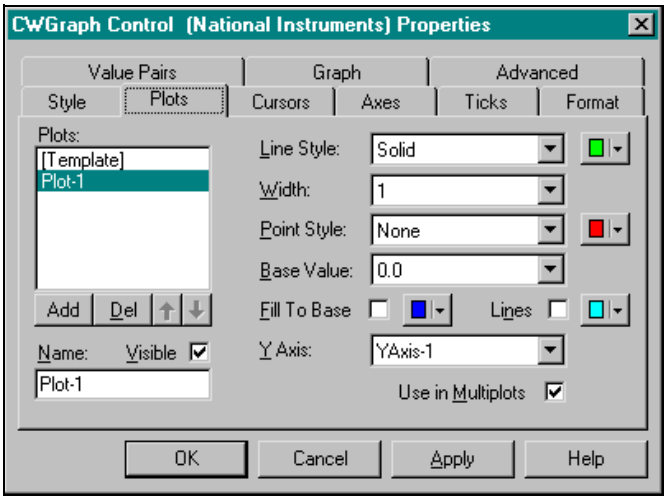
Once ActiveX controls are placed on the form, you can edit their properties using their property sheets. You can also edit the properties from within the Visual Basic program at run time.

Using Property Sheets

After dropping a control on a Visual Basic form, configure the control by setting its properties in the default and custom control property pages. Visual Basic assigns some default properties, such as the control name and the tab order. When you create the control, you can edit these stock properties in the Visual Basic default property sheet. To access this sheet, select a control and select **Properties** from the **View** menu, or press <F4>. To edit a property, highlight the property value on the right side of the property sheet and type in the new value or select it from a pull down menu. The most important property in the default property sheet is the `Name`, which is used to reference the control in the program. The following illustration shows the Visual Basic property sheet for the CWGraph control:



All other properties of an ActiveX control are edited in the custom property sheets. To open the custom property sheets right-click on the control on the form and select **Properties...**. The following illustration shows the custom property sheet for the CWGraph control:



Using Your Program to Edit Properties

You can also set or read the properties of your controls programmatically in Visual Basic. Use the name of the control combined with the name of the property as you would with any other variable in Visual Basic.

For example, if you want to change the state of an LED control during program execution, you change the `Value` property from `True` to `False`, or `False` to `True`. The syntax for setting a property in Visual Basic is `name.property = new value`. For example:

```
CWButton1.Value = False
```

Some properties can be objects that have their own properties. In this case, you string the name of the control, sub-object and property together. For example, consider the following code for the DAQ CWAI control:

```
DAQ.ScanClock.Frequency = 10000
```

In the above code, `ScanClock` is the property of the DAQ control. `Frequency` is a property of the `ScanClock` object. As an object of the DAQ control, `ScanClock` itself has several additional properties.

You can retrieve the value of control properties from your program in the same way. For example, to print the value of an LED control use the following line of code:

```
Print CWButton1.Value
```

In Visual Basic most controls have a default property such as `Value` for the Knob, Button and Slide controls. You can access the default property of a control by using its control name (without the property name attached). For example:

```
CWSlide1 = 5.0
```

is programmatically equivalent to:

```
CWSlide1.Value = 5.0
```

Consult the *How to Set the Properties of a ActiveX Control* section of Chapter 1, *Introduction to ComponentWorks*, for information on how to set properties programmatically, specifically on using objects and properties in collections. One tool in Visual Basic that is helpful with using properties in your code is the Object Browser, which is described in detail in the *Object Browser—Building Your Code in Visual Basic* section of this chapter.

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). To call the `Start` method on the DAQ analog input control, for instance, use the following code:

```
CWAI1.Start
```

Methods can also have parameters that you pass with the method, and return values that pass information back to your program. For example, the `PlotY` method for the ComponentWorks Graph control has a required parameter—the array of data to be plotted—that you must include when you call the method. If you want to plot the data returned from an Analog Input control, use the following line of code.

```
CWGraph1.PlotY Voltages
```

The `PlotY` method has some additional parameters that are optional. These are added after the `Voltages` parameter, separated by commas, if desired.

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:

```
CWAI1.AcquireData Voltages, BinaryCodes, 1.0
```

However, if you assign the return value of a method to a return variable the parameters need to be enclosed in parentheses as in the following example:

```
lErr = CWAI1.AcquireData(Voltages, BinaryCodes, 1.0)
```

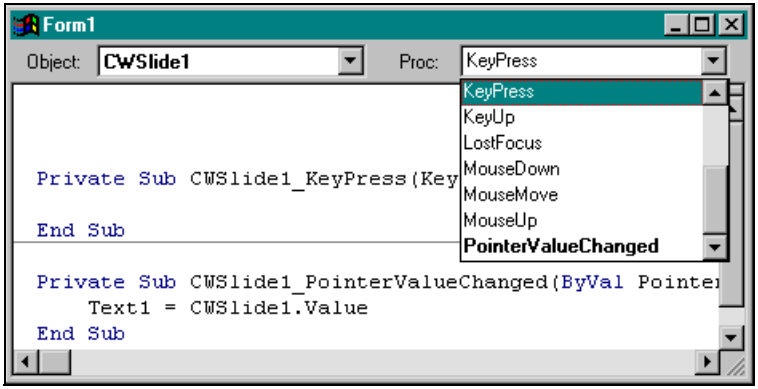
Use the Visual Basic Object Browser to add method calls to your program easily.

Developing Control Event Routines

After you configure your controls in the forms editor, you write Visual Basic code to respond to events on the controls. These events are generated by the controls 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 controls in Visual Basic, double-click on the control to open the code editor. This action automatically generates a default event handler routine for the control. The event handler routine skeleton that is generated 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 Slide control. This event routine (`PointerValueChanged`) is called when the value of the slide is changed by the user or by some other part of the program:

```
Private Sub CWSlide1_PointerValueChanged(ByVal
   Pointer As Long, Value As Variant)
End Sub
```

To generate an event handler for a different event of a control, double-click the control to generate the default handler. Then, select the desired event from the **Proc** (Procedure) field in the code window, as shown in the following illustration:



This generates an event handler routine for the selected event on the same control. Use the **Object** field in the code window to change to another control without going back to the form window.

Using the ComponentWorks Instrument Driver DLLs in Visual Basic

The ComponentWorks Full Development System comes with a library of instrument drivers. An *instrument driver* is software that handles the details of control and communications with a specific instrument.

An instrument driver consists of a set of high-level functions that controls a specific programmable instrument. Each function corresponds to a programmatic operation such as initialization, configuration, and measurement. If you did not purchase the ComponentWorks Full Development Kit and find that you need instrument drivers, contact National Instruments for information on ordering drivers.

The ComponentWorks instrument drivers are packaged as 32-bit DLLs. Before you can use any of the instrument driver functions in your program, you must add a reference to the instrument driver DLL to your project.

To add a instrument driver DLL to your project, select **References…** from the **Tools** or **Project** menu. In the **References** dialog window, press the **Browse** button and then move to the directory in which you installed your instrument drivers (the default is `\InstrDLL\System`). Select the instrument driver DLL you wish to add to your project.

After you have added the reference to the DLL, you can use any of the functions in the instrument driver without having to do any more declarations. An example of a typical instrument driver function is the initialization function for the Fluke 45 multimeter:

```
lerr = fl45_init(2, 1, 1, InstrID)
```

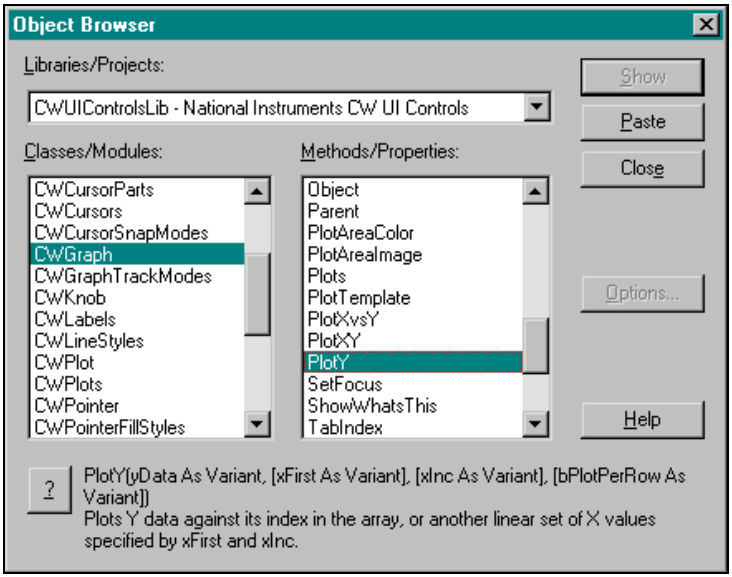
All the functions in each instrument driver are listed and described in the corresponding help file which is installed with each driver. After you add the appropriate reference to your project for using an instrument driver DLL, use the object browser to view the functions and parameters available in each instrument driver. Refer to the *Object Browser—Building Your Code in Visual Basic* section for more information on using the object browser to help you build your program.

Object Browser—Building Your Code in Visual Basic

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

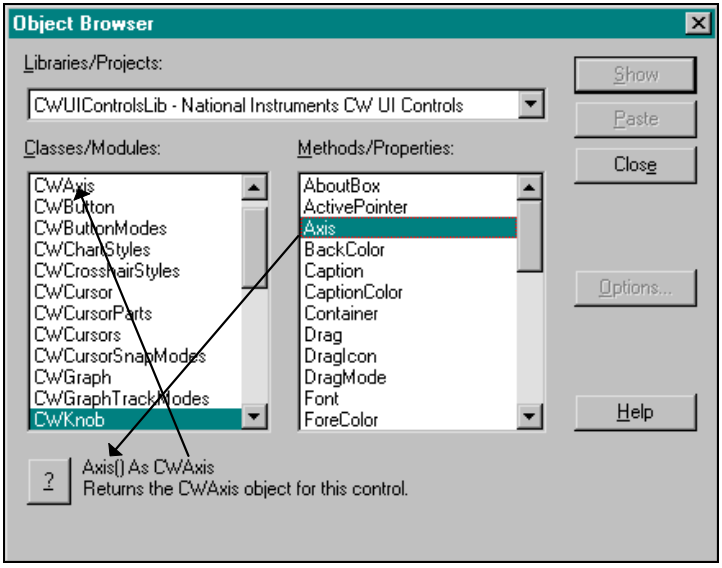
In the object browser, use the **Libraries/Projects** field to select a particular ActiveX control file, library or instrument driver. You can select any of the currently loaded controls or drivers. The **Classes/Modules** window of the object browser shows a list of controls, objects, and function classes available in the selected control file or driver.

The illustration below shows the ComponentWorks User Interface (UI) control file selected. The Classes/Modules list shows all the UI controls and associated object types. Each time you select an item from the **Classes/Modules** window in the object browser, the **Methods/Properties** window displays the properties, methods and functions for the selected object or class.



When you select an item in the **Methods/Properties** window, the prototype and description of the selected property, method or function is displayed at the bottom of the object browser dialog box. For example, in the figure above the `CWGraph` control is selected from the list of available UI objects. For this control the `PlotY` method is selected and the prototype and description of the method appears in the dialog box. The prototype of a method or function includes a list of parameters that can or must be passed.

When you select a property of a control or object in the **Method/Properties** window which is an object in itself, the description of the property includes a reference to the object type of the property. For example, the next figure shows the Knob control (`CWKnob`) selected in the **Classes/Modules** field and its Axis property, selected in the **Methods/Properties** field.



The Axis on the Knob 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 Axis object and can be selected in the Classes/Modules 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 bottom of the object browser dialog window opens the help file to a description of the currently selected item. To find more information about the CWGraph control, select the control in the window and press the **?** button.

Pasting Code into Your Program

If you open the object browser from the code editor in Visual Basic, you can paste a selected property, method, or function directly into your program. To perform this operation, place the cursor at the location in your program where you want to insert the new code. Open the object browser by pressing <F2>. Select the desired property name, method, or function in the window of the object browser, and click on the **Paste** button. This pastes a copy of the selected item into your code. When you paste in a call to a method or function that includes parameters, you must replace the parameter prototypes with actual variables or values.

You can also 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

```
CWGraph1.Axes.Item(1).ValuePairs.Item(3).Name
```

by typing in the name of the control (`CWGraph1`) and then using the object browser to add each section of the property reference. Refer to the *Item Method* section of Chapter 1, *Introduction to ComponentWorks,* for more information on the `Item` method and collections.

Learning to Use Specific ComponentWorks Controls

Each ComponentWorks control and its use is described in more detail in later chapters in this manual. However, these chapters will not discuss every property, method, and feature of each control. The ComponentWorks online help contains detailed information on each control and all its associated properties, events, and methods. Refer to onlineonline help to find descriptions of the different features of a particular control. Remember that many of the ComponentWorks controls share properties, so when you learn how to use one control, you are learning how to use others as well.

Building ComponentWorks Applications with Visual C++

Chapter 3

This chapter contains an overview of using ComponentWorks controls with Visual C++. At this point you should be familiar with the general structure of ActiveX controls described in Chapter 1, *Introduction to ComponentWorks*, as well as C++ programming and the Visual C++ environment. This chapter explains how to insert the controls into the Visual C++ environment and create the necessary wrapper classes. It also 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. This chapter discusses how to perform these general operations using ActiveX controls. The individual ComponentWorks controls are described later in this manual.

Developing Visual C++ Applications

You start developing applications in Visual C++ by creating a new workspace or project. To create a project compatible with the ComponentWorks ActiveX controls, use the Visual C++ MFC AppWizard to create a skeleton project and program. After the skeleton project is built, add any ActiveX controls to be used to the Controls toolbar. From the toolbar, you can add the controls to the application itself. After you add a control to your application, configure its properties using its property pages. While developing your program code, you can use the control properties and methods as well as create event handlers to process different events generated by the control. Creating the necessary code for these different operations is simplified by using the ClassWizard in the Visual C++ environment.

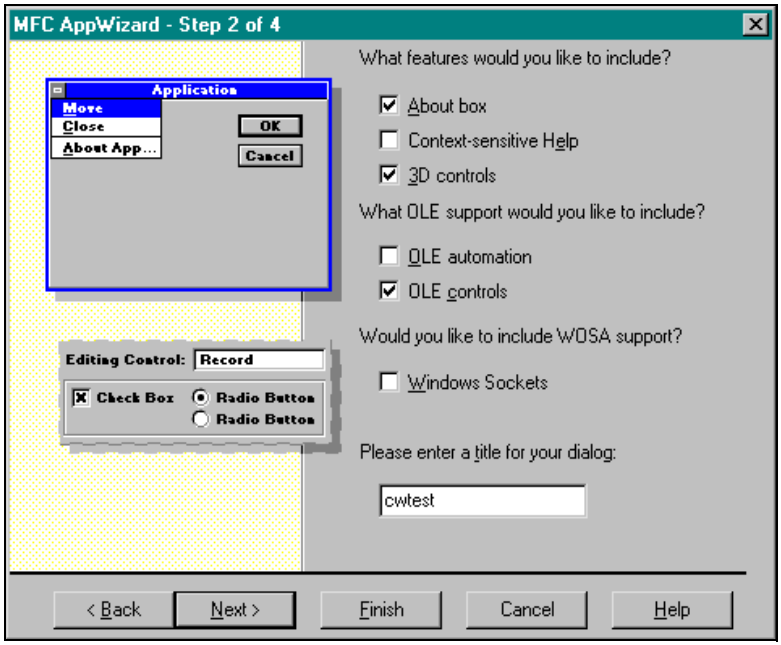
Creating Your Application

When you begin developing a new application, the project must be configured to be compatible with ActiveX controls. This is done by using the MFC AppWizard when you create the new project workspace. The MFC AppWizard creates the project skeleton and adds the

necessary code so that you can add ActiveX controls to your program later on.

Create a new project by selecting **New** from the **File** menu and then select **Project Workspace**. In the next dialog window, select the **MFC AppWizard (exe)** and enter your project name in the **Name** field. Then click on **Create** to setup your project.

In the next series of dialog windows, the MFC AppWizard prompts you for different project options. If you are not familiar with Visual C++ or the MFC AppWizard, use the default options unless otherwise noted here. In the first step, you select the type of application to build. Selecting a **Dialog based** application initially makes it easier to become familiar with the ComponentWorks controls. Click on the **Next>** button.



In the following steps of the MFC AppWizard, make sure you enable OLE/ActiveX controls support. This is done in step 2, as shown above, of the MFC AppWizard, if you select a **Dialog based** application.

Continue to the end of the MFC AppWizard, selecting desired options. The MFC AppWizard then builds a project and program skeleton according to the selected options. The skeleton includes several classes, resources, and files, all of which can be accessed from the Visual C++

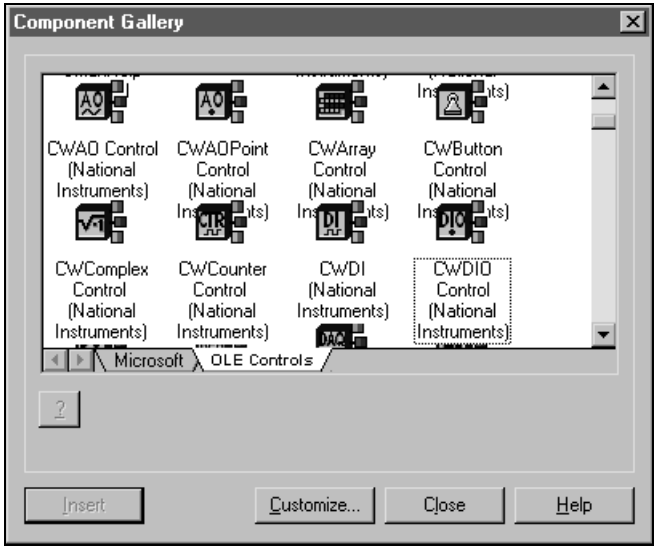
development environment. Use the **Project Workspace** window, which can be selected from the **View** menu, to see these different components in your project.

Adding ComponentWorks Controls to the Visual C++ Controls Toolbar

To use ComponentWorks controls in your application, you must load the controls into the Controls toolbar in Visual C++. This is done from the Component Gallery in the Visual C++ environment. The process of loading the controls using the Component Gallery automatically generates a set of C++ wrapper classes in your project, which are necessary to work with the ComponentWorks controls.

The Controls toolbar is only visible in the Visual C++ environment if the Visual C++ dialog editor is currently active. To open the dialog editor, open the Project Workspace window (select **Project Workspace** from the **View** menu), select the **ResourceView**, and double-click on one of the **Dialog** entries.

To add a new control, select the **Component** option from the **Insert** menu. This opens the Component Gallery in Visual C++ as shown in the following illustration:



Select the **OLE Controls** tab in the gallery and look for the ComponentWorks controls. ComponentWorks control names all start with `CW`.

If the ComponentWorks controls are not shown in the **OLE controls** tab of the Component Gallery, push the **Customize** button, and the **Import** button in the dialog window that appears. Select the OCX file on your hard drive that contains the controls you want to load. The ComponentWorks OCX files are located in the `\Windows\System(32)` directory and have names of the form `CW*.OCX`. Repeat the import process for any additional OCX files. This adds the corresponding ComponentWorks controls to the Component Gallery.

From the Component Gallery, select a control you want to add to the Controls toolbar and push the **Insert** button. The dialog window that appears lists the classes generated for the ActiveX control and the file names used. Click **OK** to continue. This adds the new classes to your project and the new control to the Controls toolbar. Repeat this process for additional controls.

When you have completed adding controls, click on **Close** in the Component Gallery. The new controls should now be visible in the Visual C++ environment Controls toolbar.

Building the User Interface Using ComponentWorks Controls

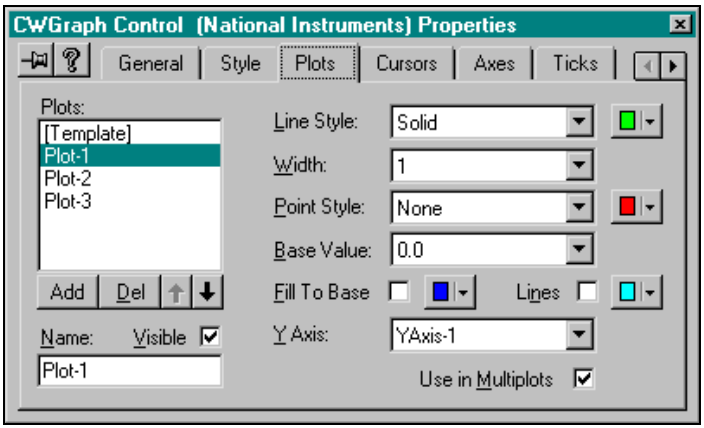
After the controls are added to the Controls toolbar, you can use the controls in the design of the application user interface. Place the controls on the dialog form using the dialog editor. You can size and move individual controls in the form to customize the interface.

Configure controls through their property sheets. This affects both their representation on the user interface as well as their behavior at run-time. Notice that the following sections assume that you are developing a dialog based application.

To add ComponentWorks controls to the form, open the dialog editor by selecting the dialog form from the **Resource View** of the **Project Workspace**. After you open the dialog editor, if the Controls toolbar is not displayed, open it by selecting **Toolbars** from the **View** menu and placing a check mark next to **Controls**.

To place a ComponentWorks control on the dialog form, select the desired control in the Controls toolbar, then click and drag the mouse on the form to create the control. After placing the controls, move and resize them on the form.

When you add a ComponentWorks control to a dialog form, configure the default properties of the control by displaying its custom property sheets. The graph control property sheets are shown below as an example:



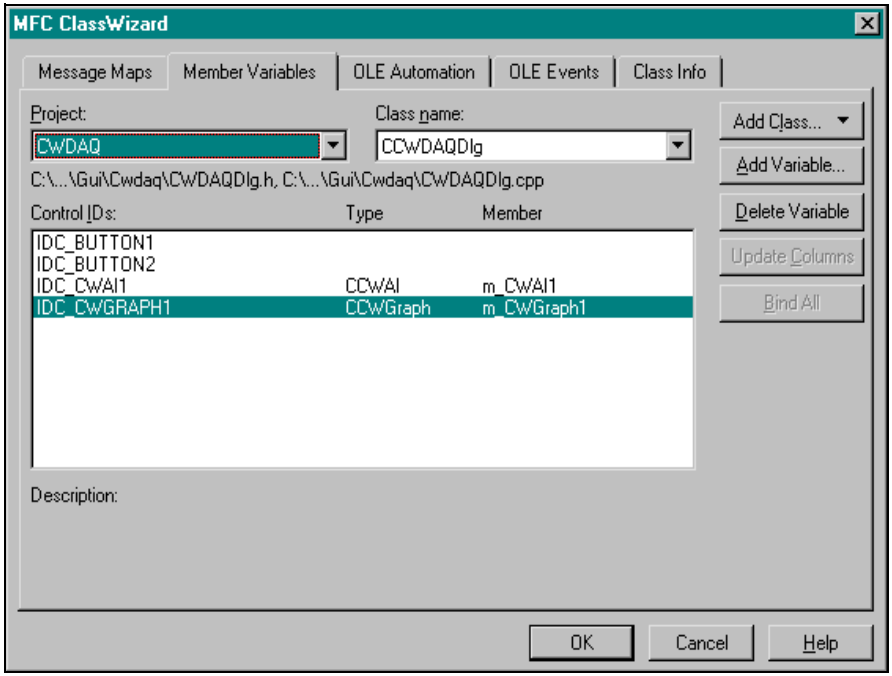
A separate window displays a sample copy of the control that reflects the property changes as you make them in the property sheets, so you can see what the control will look like. To open the property sheets, right click on the control and select **Properties**.

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++. Later chapters in this manual provide more information on the most commonly used properties, methods, and events of the individual controls. All the properties, methods and events of the different controls are described in detail in the ComponentWorks Online Reference, which you can access from the ComponentWorks folder under the **Start** menu. Consult the online documentation for information on how to use specific controls and their properties, methods and events.

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

To create a member variable for a control on the dialog form, right-click on the control and select **ClassWizard**. In the MFC Class Wizard window switch to the **Member Variables** tab.



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**. Usually member variable names start with `m_` and you should use this convention. After you create the member variable, use it to access a control from your source code. The figure above shows the MFC Class Wizard after member variables have been added for a graph and analog input control.

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 slide, use the `SetValue` function of the wrapper class for the slide control. In the source code, the function call is preceded by the member variable name of the control to which it applies. For example:

```
m_Slide.SetValue(COleVariant(5.0));
```

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

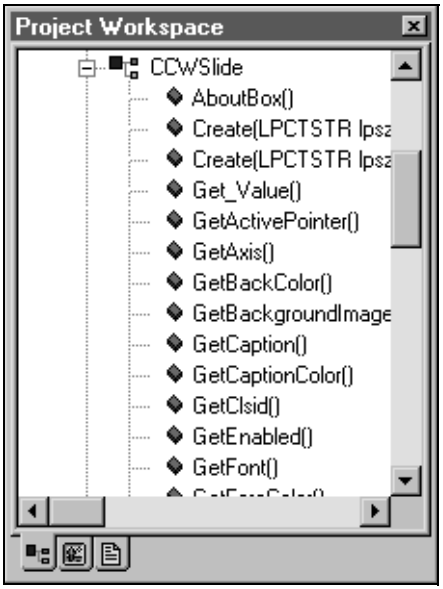
To read the value of a control use the `GetValue()` function. You can also use the `GetValue` function to pass a value of a control to another part of your program. For example, to pass the value of a slide control to a meter control, use the following line of code:

```
m_Meter.SetValue(m_Slide.GetValue());
```

The conversion to a variant type is not necessary in the previous line because the `GetValue` function returns its value as a variant.

You can see the names of all the property functions for a given control in the **ClassView** of the Project Workspace. In the **Project Workspace** window, select **ClassView**, and then select the control to view its property functions, as well as methods. The following figure shows the functions for the slide object as they are listed in the Project Workspace.

These are created automatically when you add a control to the Controls toolbar in you project.



If you need to access a property of a control which is in itself another object, use the appropriate property function to return the sub-object of the control. Then make a call to access the property of the sub-object. You need to include the header file in your program for any new object you use. For example, use the following code to configure the `Axis` object of a slide control:

```
#include“cwaxis.h”
CCWAxis Axis1;
Axis1 = m_Slide.GetAxis();
Axis1.SetMaximum(COleVariant(5.0));
```

You can chain this operation into one function call without the need to declare another variable:

```
#include “cwaxis.h”
m_Slide.GetAxis().SetMaximum (COleVariant(5.0));
```

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, to set the maximum of the first Y axis on a graph use the following code:

```
#include “cwaxes.h”
#include “cwaxis.h”
```

```
m_Graph.GetAxes().Item(COleVariant(2.0)).SetMaximum
  (COleVariant(5.0));
```

Using Methods

All the methods of each control are extracted with its wrapper class. To call a method, append the method name to the member variable name and pass the appropriate parameters. Methods that do not take any parameters should be followed by a pair of empty parentheses:

```
m_CWAI1.Start();
```

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 easily convert most scalar values to a variant by using `COleVariant()`. For example, the `PlotY` method of the graph control requires three scalar values as variants:

```
m_Graph.PlotY (*Voltages, COleVariant(0.0),
  COleVariant(1.0), COleVariant(1.0));
```

Consult the Visual C++ documentation for more information on the variant data type.

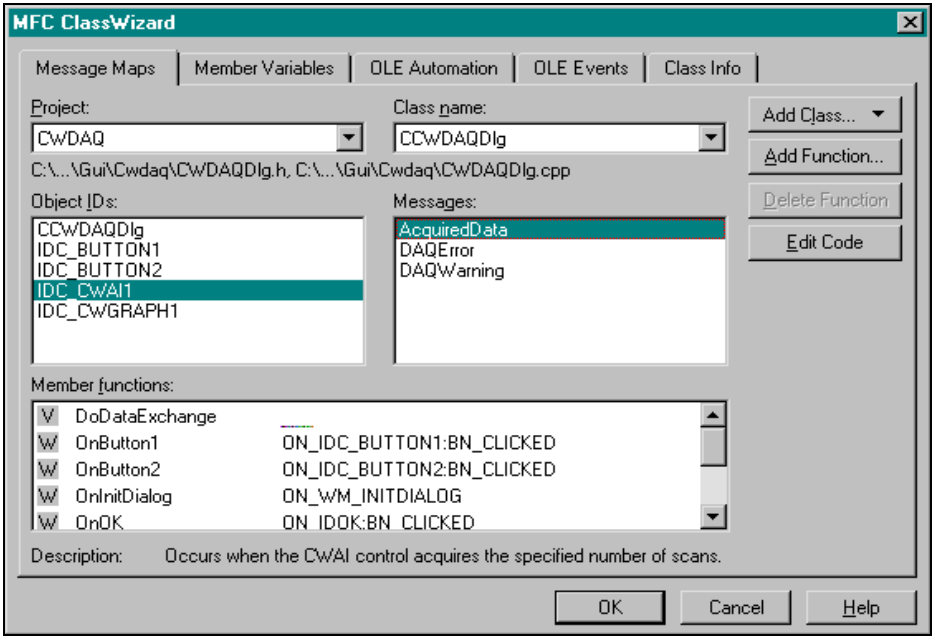
If you need to call a method on a sub-object of a control, follow the conventions outlined in the *Using Properties* section. For example, a single plot on a graph is an object in the Plots collection—which itself is an object in the graph control. To call `PlotY` on one particular plot of your graph use the following line of code:

```
m_Graph.GetPlots().Item(COleVariant(2.0)).PlotY
  (*Voltages, COleVariant(0.0), COleVariant(1.0));
```

Using Events

After you place a control on your form, you can start defining event handler functions for the control in your code. Events are generated automatically at run time by different controls in response to conditions, such as the user clicking a button on the form or the data acquisition process acquiring a specified number of points.

To create an event handler, right-click on a control and select **ClassWizard**. 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. Select the event and press the **Add Function...** button to add the event handler to your code. To switch directly to the source code for the event handler, click on the **Edit Code** button. This places the cursor in the event handler, where you can add the functions to call when the event occurs. You can use the **Edit Code** button at any time by opening the class wizard and selecting the event for the specific control.



The following is an example of an event handler generated for the `PointerValueChanged` event of a knob. Insert your own code in the event handler:

```
void CTestDlg::OnPointerValueChangedCwknob1(long
  Pointer, VARIANT FAR* Value)
{
// TODO: Add your control notification handler code here
}
```

Learning to Use Specific ComponentWorks Controls

Each ComponentWorks control and its use is described in more detail in later chapters in this manual. However, these chapters will not discuss every property, method, and feature of each control. The ComponentWorks online help contains detailed information on each control and all its associated properties, events, and methods. Refer to online help to find descriptions of the different features of a particular control. Remember that many of the ComponentWorks controls share properties, so when you learn how to use one control, you are learning how to use others.

Building ComponentWorks Applications with Delphi

Chapter 4

This chapter contains an overview of using ComponentWorks controls with Delphi. At this point you should be familiar with the general structure of ActiveX controls as described in Chapter 1, *Introduction to ComponentWorks*. This chapter explains how to insert controls into the Delphi environment, how to set their properties, and use their methods and events using ActiveX controls. Individual ComponentWorks controls are described later in the manual. This chapter also outlines Delphi features that simplify working with ActiveX controls.

Developing Delphi Applications

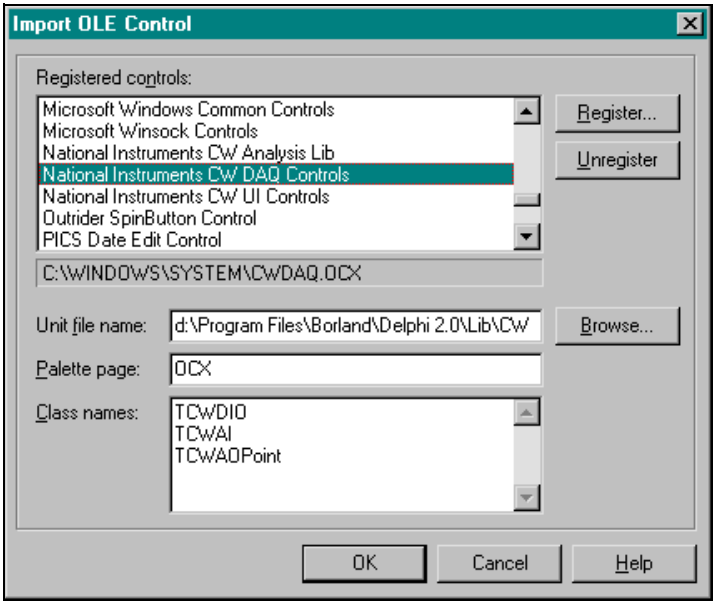
You start developing applications in Delphi using a *form*. A form is a window or area on the screen on which you can place controls and indicators to create the user interface for your programs. The Component palette in Delphi contains all of the controls available for building applications. After you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

Loading the 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. The controls only need to be added once to the palette because they will be available until they are explicitly removed from the Component palette. Adding the controls to the palette also creates a Pascal import unit (header file) that declares all the properties, methods and events of a control. When you use a control on a form, a reference to the import unit is automatically added to the program.

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.

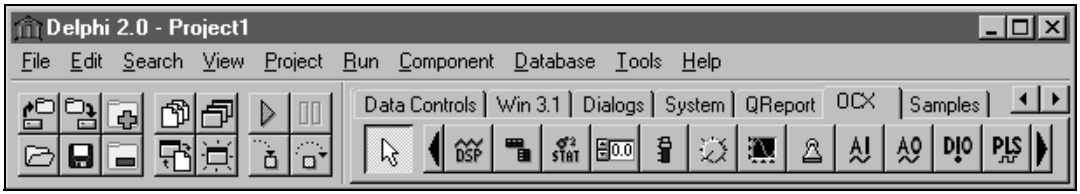
To add ActiveX controls to the Component palette, select **Install** from the **Component** menu in the Delphi environment. In the Install Components window that appears, press the **OCX** button. In the Import OLE Control window, select the desired registered control that appears on the Registered Controls field. The ComponentWorks controls all start with `National Instruments`. After you have selected the proper control, click **OK** to close the window. When you click the **OK** button, Delphi generates a Pascal import unit file for the selected `.OCX` file, which is stored in the `\Lib` directory of Delphi. If you had previously installed the same `.OCX` file, it will prompt you to overwrite the existing import unit file.



If your control does not show in the Import OLE Control window it is not registered with the operating system. In this case click on the **Register...** button to open the Register OLE Control window, find the OCX file that contains the control, and select it. This process registers the control with the operating system. Most OCX files are located in the `\System(32)` directory under Windows.

To load additional controls, return to the Import OLE Controls window and select more controls.

When you have finished selecting controls, click **OK** in the Install Components window to load the new controls and add them to the OCX tab of the Component palette. This step closes any open projects in the Delphi environment, so you will need to reopen any projects you had open.



You can rearrange controls on the Component palette in Delphi by right-clicking on the palette and selecting **Properties**.

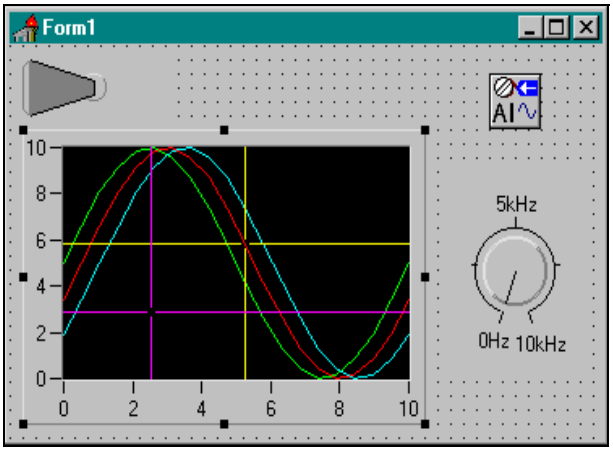
Building the User Interface

After you add the ComponentWorks controls to the Component palette, you can begin building your application. Starting with a new project, place different controls on the form. These controls are part of the user interface of the program, and also add specific functionality to the application. After you place controls on the form, configure their default property values through the stock and custom property sheets.

Placing Controls

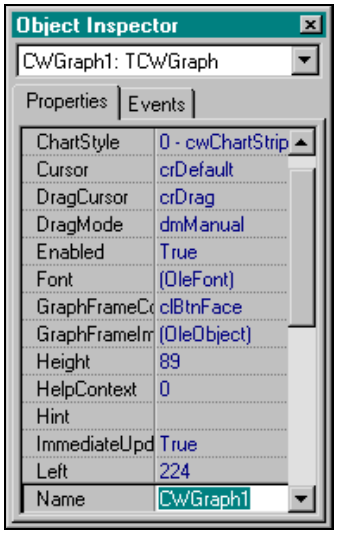
To place a control on the form, select the control from the Component palette and click and drag the mouse on the form. Select controls from the different tabs available in the Component palette. You can move and resize controls using the mouse to customize the interface. After controls are placed, you can change their default property values by

using the default property sheet (Object Inspector) and custom property sheets.

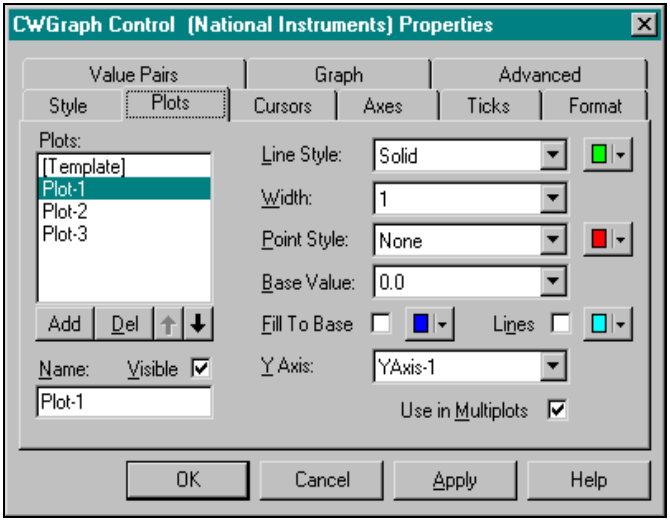


Using Property Sheets

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



To open the custom property pages of a control, double click on the control or right-click on the control and select **Properties...**. Most of the properties of a control can be edited here. The specific properties of the controls are explained later in this manual and in the online reference. The following illustration is an example of the ComponentWorks graph control property sheet:



Using Your Program to Edit Properties

You can set or read the properties of your controls programmatically from within your Delphi code. This is done by using the name of the control (set in the object inspector) combined with the name of the property, as you would a variable in Delphi.

For example, if you want to change the state of an LED control during program execution, you change the `Value` property from `True` to `False`, or from `False` to `True`. The syntax for setting the value property in Delphi is `name.property: = new_value`. For example:

```
CWButton1.Value := True;
```

Some properties can be objects that have their own properties. In this case you combine the name of the control, sub-object and property. For example, consider the following code for the DAQ CWAI control:

```
CWAI1.ScanClock.Frequency := 10000;
```

In the above code, `ScanClock` is both a property of the DAQ control, and an object itself. `Frequency` is a property of the `ScanClock` object. As an object of the DAQ control, `ScanClock` itself has several additional properties.

You can retrieve the property value of a control from your program in the same way. For example, to assign the value of a knob control to the scan rate of a CWAI control, use the following code:

```
CWAI1.ScanClock.Frequency := CWKnob1.Value;
```

To use the properties or methods of an object in a collection, use the `Item` method to extract the object from the collection. Once the object is extracted, use its properties and methods as you normally do:

```
CWGraph1.Axes.Item(2).Maximum := 5;
```

In some cases an object may be assigned as a property to another object. For example, the following code assigns a plot object of a graph to a cursor object in order to specify which plot the cursor is tracking:

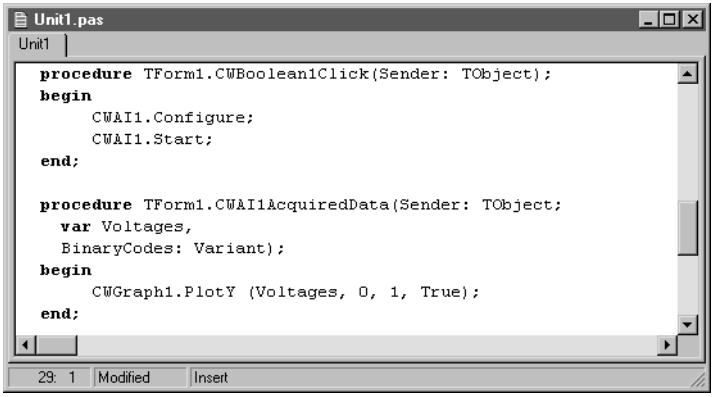
```
CWGraph1.Cursors.Item(1).Plot :=
  CWGraph1.Plots.Item(2);
```

Consult the *Setting the Properties of an ActiveX Control* section in Chapter 1, *Introduction to ComponentWorks*, for more information on setting properties programmatically.

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>. When you have placed controls on the form, you

can use their methods in your code and create event handler routines to process events generated by the controls at run-time.



Using Methods

Each control has a number of methods defined that you can use in your program. Some of these may require some parameters. To call a method in your program, use the control name followed by the method name:

```
CWAI1.Start;
```

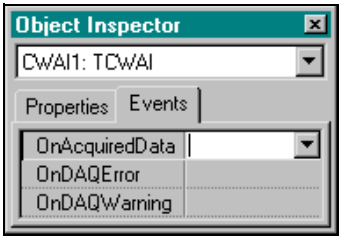
Parameters passed to a method are of type variant in most cases. Simple scalar values can automatically be converted into variant type value, and may therefore be passed to methods. Arrays, however, must be explicitly declared as variant arrays. The following example shows how to plot some data using the `PlotY` method. The data acquisition control returns its data as a variant array so it can be plotted directly on the graph control. Consult the Delphi documentation for more information on the variant data type:

```
var
     vData:Variant;
begin
     // Create array in Variant
     vData := VarArrayCreate([0, 99], varDouble);
     for i := 0 to 99 do
     begin
          vData[i] := Random;
     end;
     // Plot Variant Array
     CWGraph1.PlotY (vData, 0.0, 1.0, True);
end;
```

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 in response to user interaction with an object such as a knob, but also by other controls (such as the DAQ controls) in response to internal conditions (for example, acquisition completed or error). You can easily create a skeleton for an event handler routine using the object inspector in the Delphi environment.

To open the object inspector, press <F11> or select **Object Inspector** from the **View** menu. In the object inspector, select the **Events** tab. This lists all the events of the currently selected control. To use a specific event and create its skeleton function in your code window, double-click on the empty field next to the desired event name. Delphi generates the event handler routine in the code window using the default name for the event handler.



To specify your own event handler name, single-click in the empty field in the object inspector next to the event, and enter the desired function name. After the event handler function is created, insert the desired code in the event handler.

Learning to Use Specific ComponentWorks Controls

Each ComponentWorks control and its use is described in more detail in later chapters in this manual. However, these chapters will not discuss every property, method, and feature of each control. The ComponentWorks online help contains detailed information on each control and all its associated properties, events, and methods. Refer to online help to find descriptions of the different features of a particular control. Remember that many of the ComponentWorks controls share properties, so when you learn how to use one control, you are learning how to use others.

Using the Graphical User Interface Controls

Chapter 5

This chapter shows how to use the ComponentWorks Graphical User Interface (GUI) controls to customize your application interface to suit your needs. The GUI controls include features commonly used in instrumentation and data acquisition. They can be used to create the front end for virtually any type of application, including finance, systems management, and many others. The individual controls and most commonly used properties, methods and events are explained in this chapter. Additional information is found in the online help file.

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

What are the GUI Controls?

The user interface file `CWUI.OCX` contains five separate ActiveX controls with instrumentation-style interface controls and indicators. Each of the ActiveX controls represents a family of individual control styles. The following table lists the controls and their associated styles. You set the style of an individual control from the property sheets at design time, or it can be set through properties and methods at run-time.

Control	Control Style
CWGraph	Graph Strip Chart Scope Chart
CWButton	Slide Switch Toggle Switch Push Button Command Button Custom Bitmap Button LED
CWSlide	Horizontal and Vertical Slide Horizontal and Vertical Fill Thermometer Tank
CWKnob	Knob Dial Horizontal and Vertical Meter
CWNumEdit	Numeric Edit Box

This chapter discusses the most commonly used properties, methods and events for each of the different controls and how they are applied in typical applications. To become familiar with a specific control, read the section discussing and describing that control and work through the tutorial that implements the control in a simple program.

Object Hierarchy and Common Objects

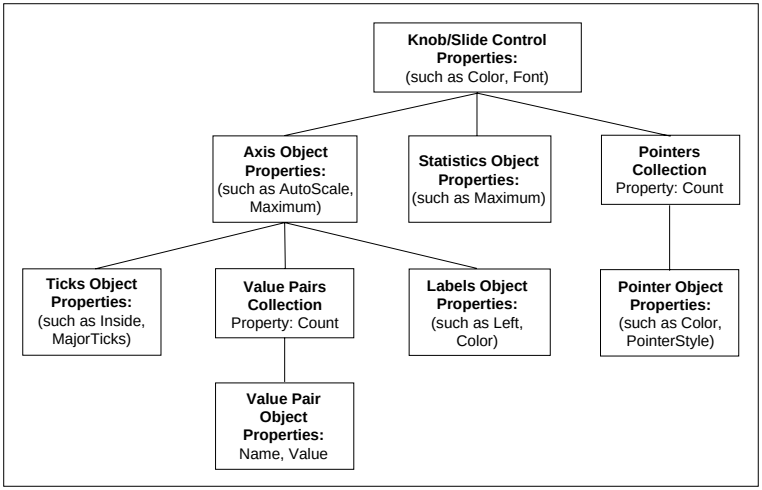
Most of the ComponentWorks user interface controls are made up of a hierarchy of less-complex objects. This chapter outlines the object hierarchy for each of the controls and describes the individual objects. Understanding how these different objects fit together to create one control is the key to properly understanding and programming the individual controls. By breaking a control down into individual components, it actually becomes much simpler to work with, because each individual component has fewer parts that you need to use at one time.

Several objects in the object hierarchy are reused throughout different controls. This further simplifies the process of becoming familiar with different controls because as you learn the parts of one control, you are learning parts of other controls at the same time. This chapter describes such common controls the first time they are encountered and will refer to that explanation from other sections.

The Knob and Slide Controls

The knob and slide controls are similar to each other. The knob control represents different types of circular displays, such as a knob, gauge, or different types of meters. The slide control represents different types of linear displays, such as thermometers and tank displays. The purpose of the knob and slide is to allow the user to input or output (display) individual or multiple scalar values. A knob or slide can have multiple pointers on the control, each pointer representing one scalar value.

Like other controls, the knob and slide are made up of a hierarchy of objects, illustrated in the following diagram, that simplifies the use of the controls:



The following sections outline the structure of the controls, followed by descriptions of the individual objects and their use. For most of the objects, the section outlines the most common properties and methods, followed by a description of the events generated by the control.

Knob and Slide Object

The knob or slide object maintains the basic attributes of the control such as background color and the caption. Its most important property, though, is the `Value` property, that 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 knob or slide 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

```
CWKnob1.Value = 5.0
```

```
x = CWSlide1.Value
```

Pointers Collection

The pointers collection of the knob or slide 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 = CWSlide1.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:

```
CWKnob1.Pointers.Item(2)
```

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

```
CWSlide1.Pointers.Item("BoilerPressure")
```

Pointer Object

The Pointer object is stored in the pointers collection, and represents one value displayed on either a knob or a slide control. It contains properties such as `Style` and `FillStyle` which affect the display of the pointer. These properties are usually set through the property sheets at design time, and not changed 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. For example:

```
MaxLimit = CWKnob1.Pointers.Item(3).Value
```

```
CWSlide1.Pointers.Item("BoilerPressure").Value =
  AcquiredPressure
```

Axis Object

The Axis object contains the information about the axis scale used on the control if it is a knob (circular scale) or a slide (straight scale). The axis object is also used in the axes collection as part of the graph control. The axis object has a number of simple properties like `AutoScale`, `Maximum`, and `Minimum` that can be set and read directly. For example:

```
CWKnob1.Axis.AutoScale = True
```

```
MaxValue = CWKnob1.Axis.Maximum
```

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

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

```
CWSlide1.Axis.SetMinMax newMin, newMax
```

Ticks and Labels Objects

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

```
CWSlide1.Axis.Ticks.AutoDivision = False
```

```
CWKnob1.Axis.Ticks.MinorUnitsInterval = 2.0
```

```
CWGraph1.Axes.Item(1).Ticks.MajorGrid = True
```

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

```
CWSlide1.Axis.Labels.Color = vbBlue
```

```
CWKnob1.Axis.Lables.Radial = True
```

```
CWGraph1.Axes.Item(1).Labels.Above = True
```

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 = CWSlide1.Axis.ValuePairs.Count
```

```
CWKnob1.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:

```
CWSlide1.Axis.ValuePairs.Item(2)
```

```
CWKnob1.Axis.ValuePairs.RemoveAll
```

```
CWGraph1.Axes.Item(2).ValuePairs.Remove 2
```

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 knob, slide, or graph control as custom ticks, labels, and grid lines. You can use value pairs on the knob and slide control to implement a *Value Pairs Only* control that limits the valid values of the control to the control's value pairs. Specify the `Name` and `Value` property of a value pair in the property sheets or at run-time. For example, to create a new value pair and set its properties, use the following code:

```
CWSlide1.Axis.ValuePairs.Add
```

```
n = CWSlide1.Axis.ValuePairs.Count
```

```
CWSlide1.Axis.ValuePairs.Item(n).Name = “Max”
```

```
CWSlide1.Axis.ValuePairs.Item(n).Value = 7.0
```

Statistics Object

The statistics object provides access to the statistical value stored by the Knob and Slide controls. 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 since the last reset while the mean is the average of the last ten values:

```
AverageMeasurement = CWSlide1.Statistics.Mean
```

```
CWKnob1.Statistics.Reset
```

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

Events

The main event on the knob and slide controls is `PointerValueChanged`. It is fired when the value of a pointer on the control is changed from the user interface or the program. This is normally used 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 slide and synchronize the two controls, use the following event handler:

```
Private Sub CWSlide1_PointerValueChanged(ByVal Pointer
  As Long, Value As Variant)
    NumEdit1.Value = CWSlide1.Value
End Sub
```

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

Consult the online help file for more information on the individual properties, methods or events.

The Numeric Edit Box Control

Use the numeric edit box control to display numbers in a manner similar to a text box. The control includes increment and decrement buttons to change the value of the control by using a mouse or touch screen. The control includes range checking, so you can preset a valid range for the control and the application will be notified if the value is set outside of the limits. The control has no other objects in its hierarchy and all properties and methods are contained in the control itself.

The most commonly used property is the `Value` property used to read and set the value of the numeric edit box control:

```
CWNumEdit1.Value = 5.0
x = CWNumEdit1.Value
```

The `Minimum`, `Maximum`, and `RangeChecking` properties allow you to configure the range checking process:

```
CWNumEdit1.Maximum = 5.0
CWNumEdit1.RangeChecking = True
```

The `SetMinMax` method can be used to set both the upper and lower limit of the range at once:

```
CWNumEdit1.SetMinMax -10,10
```

Events

The numeric edit box control has three key events, `ValueChanged`, `ValueChanging`, and `IncDecButtonClicked`.

`ValueChanged` is fired every time the value of the control has been changed from the program or user interface.

`ValueChanging` is fired any time the value of the control is changed, but before the new value is set in the control. The event returns parameters for the new value with range checking in place, the attempted value, and the previous value. `NewValue` is passed by reference so the code in the event

routine is able to revise this value before it is set in the control. You need only update `NewValue` and the changed value will be stored in the control:

```
Private Sub CWNumEdit1_ValueChanging(NewValue As
  Variant, ByVal AttemptedValue As Variant, ByVal
  PreviousValue As Variant, ByVal OutOfRange As Boolean)
    If NewValue > 100 Then NewValue = NewValue + 10
End Sub
```

`IncDecButtonClicked` is fired when either the increment or decrement button on the numeric edit box control is pressed on the user interface. The event returns a boolean parameter to indicate which of the two buttons was pressed.

Consult the online help file for more information on the individual properties, methods or events.

Tutorial: Knob, Slide, and Numeric Edit Box Controls

This tutorial shows how to use the knob, slide and numeric edit box in an application. Normally these control are used to display information or to input simple data into your application. The tutorial goes through all the steps necessary to integrate the controls with the program.

The knob and slide control each have several different display styles, such as the meter and dial for the knob, and the fill, thermometer, and tank for the slide. Although each of these styles changes the display of the control, the programmatic functionality of the control remains constant, and property sheets, event functions, and code interface are used the same way.

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 on 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 `SimpleUI`.
2. Load the ComponentWorks user interface controls (specifically, the numeric edit box, knob, and slide) into your programming

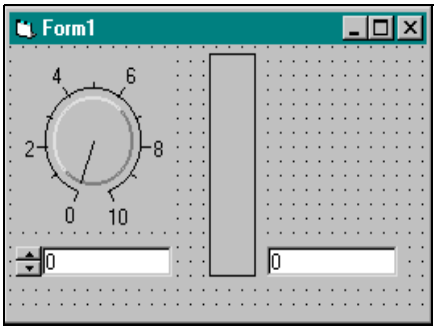
environment. Consult the chapter discussing your environment if you are not familiar with this operation.

3. From the toolbox or toolbar, place a CWKnob (knob) control on the form. Keep its default name, `CWKnob1`.

4. Place a CWSlide (slide) control on the form. Keep its default name, `CWSlide1`. Open its property sheet and select the `Vertical Fill` style. You can also change other properties such as the fill color.

5. Place a CWNumEdit (numerical edit box) control near the knob on the form. Keep its default name, `CWNumEdit1`. Keep its default property values.
6. Place another CWNumEdit (numerical edit box) control near the slide on the form. Change its name from `CWNumEdit2` to `CWSlideDisplay`. To change the name, in Visual Basic, use the default property sheet (press <F4>). In Visual C++, open the custom property sheets. In Delphi, use the object inspector (press <F11>). Open its custom property sheet, under the **Style** tab, select the **Indicator** Control Mode and unselect the **Visible** property of the Inc/Dec Button. You can also change other properties such as the font used in the display.

Your form should look similar to the one shown below.



Developing the Program Code

This program uses the numeric edit box next to the slide (without the increment or decrement arrows) to display the value of the slide control. The knob is used to change the value of the slide, and the other numeric edit box is used to change the value of the knob, thereby changing the value of the slide.

To have your program respond when the slide value changes, add the `PointerValueChanged` event for the slide. Use the Value property to retrieve or set the current value of the controls.

1. Create a skeleton event handler for the `PointerValueChanged` event of CWSlide1.
   a. In Visual Basic, double-click on the slide control on the form to create the `CWKnob1_PointerValueChanged` subroutine.
   b. In Visual C++, use the MFC ClassWizard to create the event handler routine. Right-click on the slide control and select **ClassWizard**.
   c. In Delphi, use the object inspector to create the event handler routine. Select the slide control, press <F11> to open the object inspector, select the Events tab and double-click the empty field next to the `PointerValueChanged` event.
2. Add the following code inside the event handler routine. If you are working Visual C++, first add a member variable for each control to the application dialog class.
   a. Visual Basic:

      ```
      CWSlideDisplay.Value = CWSlide1.Value
      ```

   b. Visual C++:

      ```
      m_CWSlideDisplay.SetValue(m_CWSlide1.GetValue());
      ```

   c. Delphi:

      ```
      CWSlideDisplay.Value:= CWSlide1.Value;
      ```

3. Repeat step 1 for the knob control.
4. Add the following code to the `CWKnob1_PointerValueChanged` event routine, adjusting for your programming language:

   ```
   CWSlide1.Value = CWKnob1.Value
   ```

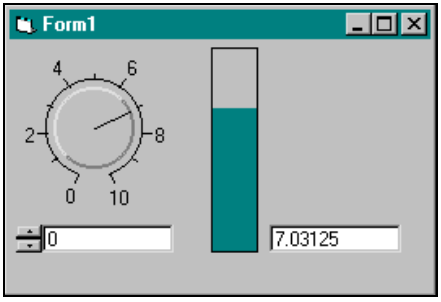
5. Repeat step 1 for the numeric edit box (`CWNumEdit1`) control.
6. Add the following code to the `CWNumEdit1_PointerValueChanged` event routine, adjusting for your programming language:

   ```
   CWKnob1.Value = CWNumEdit1.Value
   ```

7. Save the project and associated files as `SimpleUI`.

Testing Your Program

Run the program. Notice that the slide display and associated numeric edit box change as you turn the knob. Notice that when you change the value of the other numeric edit box (with the increment or decrement arrows), both the knob and slide value change.

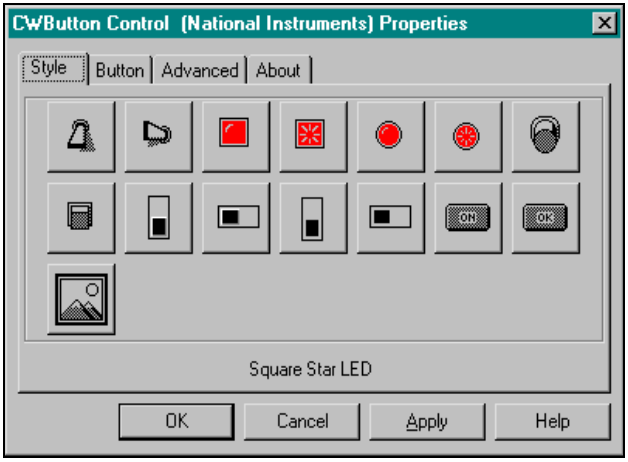


The program calls the `CWKnob1_PointerValueChanged` function and updates the slide control every time the value of the knob changes while the mouse button is pressed. Because the slide control has its own `PointerValueChanged` routine, the associated numeric edit box is always updated when the value of the slide control changes. Finally, when you change the value of the other numeric edit box, its own `PointerValueChanged` routine updates the value of the knob, calling the `PointerValueChanged` routine of the knob, and so on.

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

The Button Control

The button control is a simple control you use to input or output boolean information from your application or to initiate some action in your program. Because of its simplicity, it is made of only one object. The different styles of the button include toggle switches, LEDs, push buttons, slides, on/off buttons and custom bitmap buttons. The mode property of the button allows the button, regardless of style, to act as a command button that switches state only while the mouse button is pressed down. This mode is used to initiate action in your program without changing the state of the button permanently.



The most commonly used property on the button control is `Value`. It is used to set the state of the control, such as for an LED or to read the state of the control:

```
CWAlarmLED.Value = AlarmState
```

```
If (CWButton1.Value = True) Then...
```

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

There are no methods on the button control.

Events

The most important event generated by the button control is `ValueChanged` which notifies the application that the button value has changed. This event is generated if the button is in switch mode (switch value when clicked on) or in command mode (switch value until released):

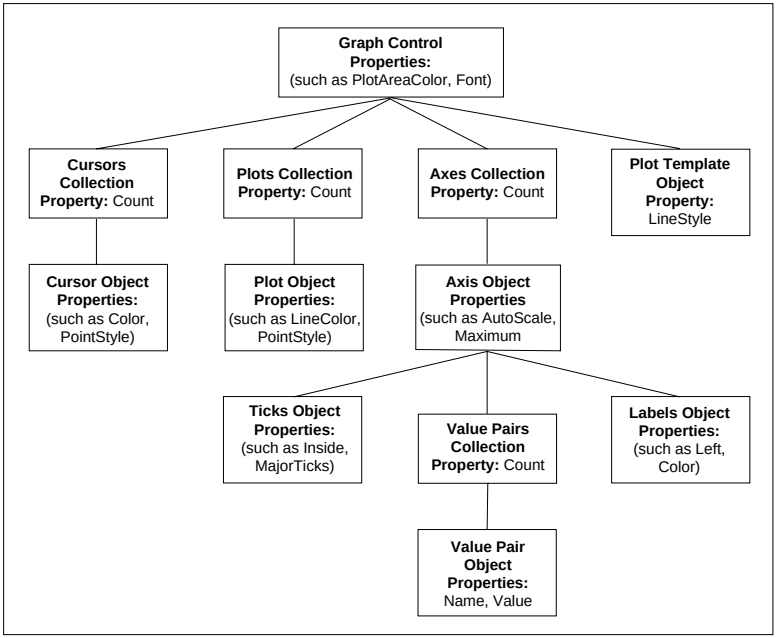
```
Private Sub CWButton1_ValueChanged(ByVal Value As
  Boolean)
'insert code to run when button is pressed
End Sub
```

The Graph Control

The graph control is a complex control used for plotting and charting data. It can be used to display multiple traces and supports multiple cursors and Y axes. *Plotting* data refers to the process of taking a large number of points and updating one plot on the graph with new data. The old plot is replaced with the new plot. *Charting* data appends new data points to an existing plot over time. It is used with slow processes where only few data points per second are added to the graph. When more data points are added than can be displayed on the graph, it starts to scroll so that new points are added to the right side of the graph while old points disappear on the left. The same graph control can be used both for charting and plotting. Select between the two operations by using different methods for displaying the data.

The graph control is made of a hierarchy of objects used to interact with the control programmatically. At design time, you can manipulate properties of the individual objects through the property sheets. The following sections outline the structure of the control followed by descriptions of the individual objects and their use. For most of the

objects, the section outlines the common properties and methods, followed by a description of the events generated by the control.



The objects in the graph control hierarchy represent the different parts displayed on the physical representation of the graph. The three main parts are the axes collection and axes objects, plots collection and plot objects, and cursors collection and cursor objects. Additionally, the plot template object serves as a template for new plot objects created in the plots collection.

The graph object contains the basic properties of the control such as name, graph frame color, plot area color, and track mode.

The axes collection and axis objects control the different axes on the graph. The graph contains one X axis and a varying number of Y axes, all contained in the axes collection.

The cursors collection and cursor objects control the cursors on the graph. Cursors are normally created at design time by using the property sheets. You can use cursors to mark a specific point or region on the graph, or to highlight something programmatically.

Graph Object

The graph object has several simple properties, such as its name and colors, that are usually set in the property sheets during design time. Other important properties that affect the behavior of the graph, generation of events, and some of its parts, such as the cursors, are `TrackMode`, `ChartStyle` and `ChartLength`. The `TrackMode` property specifically determines how mouse interaction with the graph is interpreted and is used to implement cursors, zooming, and panning.

There are several methods in the graph object which are called directly on the graph control. These are the `Plot` and `Chart` methods. These methods are called on the graph object to send data to multiple plots at once and can also be called on individual plot objects to send new data to one specific plot at a time. Use the `Plot` methods to update and replace all the data on the plots, and `Chart` methods to append new data to the plots.

Plot Methods

The following three plot methods accept data in a slightly different format:

- `PlotY` (`yData`, `xFirst`, `xInc`, `bPlotPerRow`) plots Y data evenly spaced on the X axis relative to the index in the array. Using the `xFirst` and `xInc` parameters, you can specify the X value at the first data point and the incremental X value between data points. `yData` can be a one dimensional array which updates the first plot on the graph or a two dimensional array which updates the first *n* plots on the graph. The `bPlotPerRow` parameter determines whether a two dimensional data array is plotted by row or by column.
- `PlotXY` (`xyData`, `bPlotPerRow`) plots a two dimensional array of data. Depending on the `bPlotPerRow` parameter, the first row or column in the data array contains the X data. Subsequent rows or columns contain the Y data for different plots.
- `PlotXvsY` (`xData`, `yData`, `bPlotPerRow`) plots a one dimensional or two dimensional array of Y data against a one dimensional array of X data. Depending on the `bPlotPerRow` parameter, each row or column of Y data generates one plot.

In some programming environments, some of these parameters are optional. These parameters use a default value if not explicitly specified:

Visual Basic (some parameters optional):

```
CWGraph1.PlotY Voltages
```

Visual C++ (all parameters required):

```
m_CWGraph1.PlotY (VariantArray, COleVariant(0.0),
 COleVariant(1.0), COleVariant(1.0));
```

Delphi (all parameters required):

```
CWGraph1.PlotY (Voltages, 0.0, 1.0, True);
```

Chart Methods

The following three chart methods accept data in a slightly different format:

- `ChartY` (`yData`, `xInc`, `bChartPerRow`) charts Y data on one or more plots relative to the index of the data. The `xInc` parameter determines the X spacing between points passed to a plot. `yData` can be a scalar value adding one point to the first plot, a one dimensional array adding *n* points to the first plot, or one point to *n* plots, or a two dimension array adding multiple points to multiple plots. The `bPlotPerRow` parameter determines if plots in arrays are stored by row or column. This is applicable to one dimensional and two dimensional arrays.
- `ChartXY` (`xyData`, `bChartPerRow`) charts a two dimensional array of data. Depending on the `bPlotPerRow` parameter, the first row or column in the data array contains the X data. The subsequent rows or columns contain the Y data for different plots.
- `ChartXvsY` (`xData`, `yData`, `bChartPerRow`) charts a one dimensional or two dimensional array of Y data against a one dimensional array of X data. Depending on the `bPlotPerRow` parameter, each row or column of Y data generates one plot.

In some programming environments, some of these parameters are optional. These parameters use a default value if not explicitly specified:

```
CWGraph1.ChartY VariantArray,,False
```

Plots Collection

The Plots collection is a standard collection containing plot objects. The collection contains one property `Count` that returns the number of plot objects in the collection.

```
NumPlots = CWGraph1.Plots.Count
```

Normally all plots and their properties are defined at design time in the property sheets. You can use the `Add`, `Remove`, and `RemoveAll` method to programmatically change the number of plots on the graph. When you add a plot to the collection, the new plot has the properties of the plot template object (see *PlotTemplate Object* section). The `Remove` method requires the index of the plot to be removed.

```
CWGraph1.Plots.Add
```

```
CWGraph1.Plots.Remove 3
```

Use the `Item` method of the plots collection access any particular plot object in the collection.

```
Dim Plot1 as CWPlot
```

```
Set Plot1 = CWGraph1.Plots.Item(1)
```

Plot Object

The Plot object represents an individual plot on the graph. The object contains a number of different properties that determine the display of the plot. These properties include `LineColor`, `LineStyle`, `PointColor`, `FillToBase`, and others. These properties can be set during design in the property sheets and can also be changed programmatically.

```
CWGraph1.Plots.Item(1).LineColor = vbBlue
```

```
CWGraph1.Plots.Item(1).PointStyle = cwPointAsterisk
```

The following code fills the space between the first and second plot on the graph red.

```
CWGraph1.Plots.Item(1).FillToBase = True
```

```
Set CWGraph1.Plots.Item(1).BasePlot =
  CWGraph1.Plots.Item(2)
```

```
CWGraph1.Plots.Item(1).FillColor = vbRed
```

Each plot object has a set of `Plot` and `Chart` methods similar to the methods on the graph object. See the *Plot Methods* and *Chart Methods* sections for detailed descriptions of these methods. Calling these methods on the plot object directly allows you to update one individual plot on the graph without affecting the other plots. For the `PlotY`, `PlotXvsY`, `ChartY` and `ChartXvsY` methods, the only difference to the graph object is that the `bplotPerRow` parameter falls away and all data arrays are one dimensional only:

```
CWGraph1.Plots.Item(4).PlotY Voltages
```

```
CWGraph1.Plots.Item(2).ChartXvsY xData, yData
```

The new protoypes of the `PlotXY` and `ChartXY` methods are:

```
ChartXY (xyData, bXInFirstRow)
```

```
PlotXY (xyData, bXInFirstRow))
```

With these methods, you use a two dimensional data array with exactly two rows or two columns. The `bXInFirstRow` parameter specifies whether the x and y data is stored in rows or columns.

```
CWGraph1.Plots.Item(4).PlotXY xyData, True
```

PlotTemplate Object

The PlotTemplate object of the graph object is a special copy of the plot object. It is used to specify the default property values of new plots. The plot template object properties are the same as of the plot object and are set through the property sheets or programmatically.

The plot template property values are used as the default property values for a newly created plot when the `Add` method is called on the Plots collection. New plots are created dynamically if one of the `Chart` or `Plot` methods is called on the graph control and data for more plots is passed to the method than are defined in the plots collection. For example, if only one plot is defined, but the PlotY method is called with data for five plots, the defined plot is used for the first plot and four new plots are dynamically created to display the additional data. The new plots use the plot template property values.

```
CWGraph1.PlotTemplate.LineColor = vbRed
```

Cursors Collection

The cursors collection is a standard collection containing cursor objects. To move the cursors using the mouse while running an application, the `TrackMode` property must be set to a value that supports this operation.

The cursors collection contains one property, `Count`, that returns the number of cursor objects in the collection:

```
NumCursors = CWGraph1.Cursors.Count
```

Normally all cursors and their properties are defined at design time in the property sheets. If necessary, you can use the `Add`, `Remove` and `RemoveAll` method to programmatically change the number of cursors on the graph. The `Remove` method requires the index of the cursor to be removed:

```
CWGraph1.Cursors.Add
```

```
CWGraph1.Cursors.Remove 3
```

Use the `Item` method of the cursors collection access any particular cursor object in the collection:

```
Dim FirstCursor as CWCursor
```

```
Set FirstCursor = CWGraph1.Cursors.Item(1)
```

Cursor Object

The cursor object controls the position and other attributes of the individual cursors on the graph. Two of the most frequently used cursor object properties are `XPosition` and `YPosition`, used to read or set the position of the cursor on the graph:

```
x = CWGraph1.Cursors.Item(2).XPosition
```

```
CWGraph1.Cursors.Item(1).YPosition = YLimit
```

A cursor can also be associated with a specific plot on a graph. Set this association in the property sheets or programmatically using the `SnapMode` and `Plot` properties of the cursor. If a cursor is associated with a specific plot, the cursor `PointIndex` property sets the cursor at any specific index on the plot or returns the cursor position on the plot. The `SnapMode` property is set by using a predefined ComponentWorks

constant. In Visual C++ and Delphi, these constants are defined in a separate header file which must be included in your program:

```
CWGraph1.Cursors.Item(1).SnapMode = cwCSnapPointsOnPlot
```

```
Set CWGraph1.Cursors.Item(1).Plot =
  CWGraph1.Plots.Item(1)
```

```
ptIndex = CWGraph1.Cursors.Item(1).PointIndex
```

Axes Collection

The Axes collection is a standard collection containing all the axis objects of the graph. A graph has one X axis and a varying number of Y axes. The number of Y axes is determined by the developer at design time and can be changed programmatically at run time. These different axis objects are all contained in the axes collection and can be referenced by index. Normally the X axis is at index one and the Y axes are at subsequent indices.

The axes collection contains the property `Count`, which returns the number of axis objects in the collection:

```
NumAxes = CWGraph1.Axes.Count
```

Normally all axes and their properties are defined at design time in the property sheets. If necessary you can use the `Add`, `Remove`, and `RemoveAll` method to change programmatically the number of axes on the graph. The `Remove` method requires the index of the axis to be removed.

```
CWGraph1.Axes.Add
```

```
CWGraph1.Axes.Remove 3
```

Use the `Item` method of the axes collection access any particular axis object in the collection.

```
Dim xAxis as CWAxis
```

```
Set xAxis = CWGraph1.Axes.Item(1)
```

Axis Object

The axis object contains all the properties of the individual axes on the graph, and is identical to the axis object used on the knob and slide controls. In addition, the axis object contains three other objects, Labels, Ticks, and the Value Pairs collection that define different parts of an individual axis. The axis object and its parts are described in detail in the *Knob and Slide Control* section.

Events

The graph generates a number of different events used to enable your application to react to user interaction with the graph. The graph processes certain mouse interactions automatically without the need to develop any event handler routines.

The type of events generated and other automatic processing is determined by the `TrackMode` property on the graph object which is set through the property sheets or programmatically. Some of the common modes on the graph generate events for mouse interaction with cursors, plots, and the plot area, as well as moving cursors and panning and zooming the graph.

To move the cursors with the mouse during program execution set the `TrackMode` property to a compatible value using either the property sheets or your application.

```
CWGraph1.TrackMode = cwGTrackDragCursor
```

In this mode, the graph generates the `CursorChange` event when any of the cursors move. This event can initiate some action in your application in response:

```
Private Sub CWGraph1_CursorChange(CursorIndex As Long,
  XPos As Variant, YPos As Variant, bTracking As Boolean)
    xDisplay = XPos
    yDisplay = YPos
End Sub
```

Panning and Zooming

The `TrackMode` allows you to specify the panning and zooming modes on the graph.

Panning moves the displayed plot area by using the mouse on the graph. You can scroll through a time based data set displayed on the graph. Panning can be enabled for both the X and Y axes or limited to one of the axes.

Zooming selects an area of the graph and zooms in on the selection. Zooming can also be enabled for both axes or limited to one.

Using programmatic changes of the `TrackMode` property, you can allow your users to select the different modes on the graph by clicking on a button or setting a switch. For example, to use a slide with defined value pairs:

```
Private Sub CWSlide1_PointerValueChanged(ByVal Pointer
  As Long, Value As Variant)
    Select Case CWSlide1.ValuePairIndex
        Case 1
            CWGraph1.TrackMode = cwGTrackZoomRectXY
        Case 2
            CWGraph1.TrackMode = cwGTrackPanPlotAreaXY
        Case 3
            CWGraph1.TrackMode = cwGTrackDragCursor
    End Select
End Sub
```

Consult the online help file for more information on the graph control and its individual properties, methods or events.

Tutorial: Graph and Button Controls

This tutorial shows you how to integrate the button and graph controls in a simple application.

The button control, similar to the slide and knob, has several different display styles while maintaining one set of property sheets, event functions, and one style of interaction with the program. You can use the button control as an input or as an output. That is, as an input you can create a push button or a switch to initiate an action or switch between actions. For an output, you can create an LED to indicate a boolean condition.

The graph is the most complex of the user interface objects. You can use it in two basic modes—*Plot* and *Chart*—which you select by using different methods in your program to pass data to the graph.

This tutorial uses Visual Basic syntax, but the discussion is in general terms so that 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 on 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 `ButtonGraphExample`.
2. Load the ComponentWorks user interface controls (specifically the button and graph) into your programming environment. Consult the chapter discussing your programming environment if you are not familiar with this operation.

3. Place a ComponentWorks graph control (shown at left) on the form. Keep its default name, `CWGraph1`. Open the graph control property sheet and, in the **Axes** tab, change the X-axis range to 0 to 20 and disable autoscaling. Examine the remaining tabs.

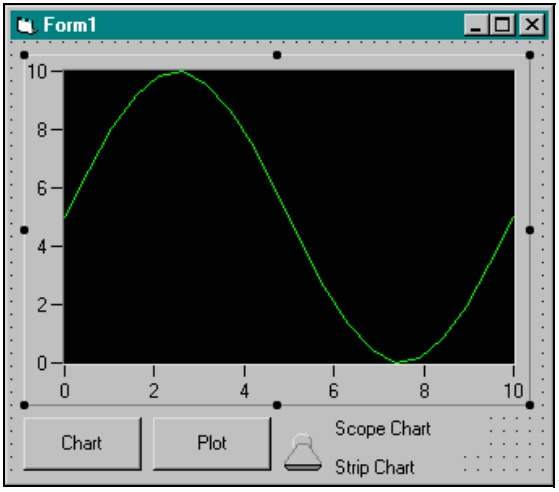
   Most of the settings in the property sheet should be self-explanatory. For more information about some of the advanced features, refer to Chapter 8, *Building Advanced Applications,* and to the online help.

4. Place two ComponentWorks buttons (shown at left) on the form. Change their name property to `Chart` and `Plot`. This step is done in the default property sheets in Visual Basic and Delphi Object Inspector or the custom property sheets in Visual C++.

   In the custom property sheets, change each of their styles to `Command Button`. Also, in the **Button** tab of the property sheets, change the **On Text** and **Off Text** to `Chart` for the first button and `Plot` for the second button.

5. Place another ComponentWorks button on the form. Change its name to *ChartSelect*. Leave its style as Vertical Toggle.
6. Place two Visual Basic labels next to the toggle button, changing their label property so that the up state of the switch is labeled `Scope Chart` and the down state `Strip Chart`.

Your form should now look similar to the one shown below.



Developing the Code

Develop the code so that data is either plotted or charted on the graph in response to the different buttons.

1. Define an event handler routine for the **Plot** button to be called when the button is pressed. In the event handler the program will create an array of 20 points and plot it on the graph.

   Generate the event handler routine for the `Click` event of the Plot button. Add the following code to the `Plot_Click` subroutine. In Visual C++, remember to generate member variables for any controls referenced in the program. See the online tutorial programs for Visual C++ and Delphi code examples.

```
Private Sub Plot_Click()
    Dim data(0 To 20) As Double
    CWGraph1.Axes.Item(1).Maximum = 20
    CWGraph1.Axes.Item(1).Minimum = 0
    For i = 0 To 20
        data(i) = Rnd * 10#
    Next i
    CWGraph1.PlotY data, 0, 1, True
End Sub
```

   This code generates an array of 20 random numbers. The `PlotY` method then replaces any data on the plot and plots the new data starting at zero on the X axis. To ensure that the new data appears on the graph, the minimum and maximum values of the X axis (`CWGraph1.Axes.Item(1)`) are reset to 0 and 20. This routine uses a one-dimensional array with the `PlotY` method to generate one trace. You can also use a two-dimensional array to generate multiple traces. For more information about the parameters on the `PlotY` method, see the description of the graph control.

2. The Chart button when pressed will chart some random data on the graph. Generate the event handler routine for the `Click` event of the Chart button. Add the following code to the `Chart_Click` subroutine:

```
Private Sub Chart_Click()
    Dim data As Double
    For i = 0 To 59
        data = Rnd * 10#
        CWGraph1.ChartY data, 1, True
    Next i
End Sub
```

   The `Chart_Click` subroutine performs a similar action to the `Plot_Click` routine, except that the random points are generated and charted individually. When the `ChartY` method is called, it appends the new data to the data already on the graph. If you only add one point at a time, you can use a scalar value, or you may use a one-dimensional array to pass multiple points to the trace. Use a two-dimensional array to chart multiple traces.

   The `ChartStyle` property on the graph sets the style of charting used. This example uses the toggle switch to switch between the two charting styles. To have your program respond when the user switches styles, add the `ChartSelect_ValueChanged` event handler routine for the switch. Use the `Value` property of the switch to retrieve or set the current Value of the control.

3. Generate the event handler routine for the `ValueChanged` event of the switch. Add the following code. In Visual C++ and Delphi, you also must include the appropriate header files to define the constant values:

```
Private Sub ChartSelect_ValueChanged(ByVal Value As
  Variant)
    If ChartSelect.Value = True Then
        CWGraph1.ChartStyle = cwChartScope
    Else
        CWGraph1.ChartStyle = cwChartStrip
    End If
End Sub
```

4. Save the project and form as `ButtonGraphExample`.

Testing Your Program

Run and test the program. Notice the difference between the Plot, Strip Chart, and Scope Chart options.

`PlotY` and `ChartY` are the two most common methods for passing data to the graph. There are two more Plot (`PlotXY` and `PlotXvsY`) and two more Chart (`ChartXY` and `ChartXvsY`) methods that change how data passes to the graph. For more information on these methods, see the description of the graph control or the online help file. Refer to Chapter 8, *Building Advanced Applications*, for more information about other advanced graph features, such as cursors and multiple axes.

Using the Data Acquisition Controls

Chapter 6

This chapter shows how to use the ComponentWorks data acquisition (DAQ) controls in your application to perform input and output operations using your DAQ hardware. The DAQ controls are used to program your DAQ hardware and integrate these operations into the rest of your application. The individual controls and their most commonly used properties, methods and events are explained in this chapter. Additional information is found in the online help file.

This chapter also includes tutorials that give step-by-step instructions on using the controls in simple programs. While the code listed with the tutorials uses the Visual Basic syntax, the steps can be applied to any programming environment. Consult the appropriate chapters in this manual for information on using the ComponentWorks controls in another environment.

The data acquisition file `CWDAQ.OCX` contains nine separate ActiveX controls for performing DAQ operations as well as a utility control that contains miscellaneous DAQ support functions. Each control is used for one specific type of operation such as analog input, analog output, and so on. The following is a list of the DAQ controls:

- CWAIPoint — Single Point Analog Input
- CWAI — Waveform Analog Input
- CWAOPoint — Single Point Analog Output
- CWAO — Waveform Analog Output
- CWDIO — Single Point Digital Input/Output
- CWDI — Waveform Digital Input
- CWDO — Waveform Digital Output
- CWCounter — Data Acquisition Counter Functions
- CWPulse — Data Acquisition Pulse Generation Functions
- CWDAQTools — Data Acquisition Utilities

This chapter discusses the most commonly used properties, methods and events for each of the different controls, and how they are applied in typical applications. To become familiar with a specific control, read the section discussing the control and work through the tutorial that shows you how to implement the control in a simple program.

Most properties are normally set through property pages as you design and create the program. This is also the best place to become familiar with the different properties of a control. In certain cases you need to change the value of one or more properties in your program code. The following sections give many examples of how to change property values programmatically.

Data Acquisition Configuration

Before you can use your National Instruments data acquisition hardware with the ComponentWorks DAQ controls you must configure your DAQ device using the NI-DAQ driver configuration utility. Make sure that you follow the directions in the NI-DAQ driver documentation (online) for properly configuring the hardware. The configuration utility also allows you to test the hardware and perform simple input/output operations. Once a data acquisition device is configured it is assigned a device number that is used to reference the device in your application. Select the desired device and device number in the property sheets of each control.

Object Hierarchy and Common Properties

Some of the ComponentWorks data acquisition controls are made up of a hierarchy of less complex individual objects. This chapter outlines the object hierarchy for each of the controls and describes the individual objects. Understanding how these different objects fit together to create one control is the key to properly programming with the individual controls. Breaking a control down into individual objects makes it easier to work with a control, because each individual component has fewer parts. Additionally, more complex features are part of lower level objects with which a beginning user does not need to interact. You can also set all properties directly from the property pages of the controls.

Some objects in the hierarchy are re-used by different controls. This simplifies the process of becoming familiar with different controls, because as you learn the parts of one control, you are also learning parts

of others. This chapter describes such common controls the first time they are encountered, and refers to their description from other sections.

The main objects of each DAQ control share certain common properties described in the following section.

Device, DeviceName, and DeviceType

Each control has a `Device` property you use to select which hardware device is used by the control. You can set this property from a pull down menu in the property sheets of each control, or set it programmatically as in the following example:

```
CWAI1.Device = 2
```

`DeviceName` and `DeviceType` are read-only properties that return the name and type number of the selected device. The name of a device is its descriptive name such as `AT-MIO-64E-3`. The type number is a unique number assigned to each hardware device type in the NI-DAQ driver. Use these properties to control the execution of your application depending on the device used. You can also build a list of available devices in your system. For example:

```
If CWAI1.DeviceType = 16 then…
```

Channel Strings

Most DAQ controls have some kind of channel string property in their object hierarchy. Use the channel string to specify what channels on a data acquisition device are used by a particular operation. If you use only one channel, enter the channel number in the string. For example:

```
CWAOPoint1.ChannelString = "1"
```

In many cases, you need to specify more than one channel in a channel string. There are several conventions you can use to do this. If you want to specify a series of consecutive channels, specify the first and last channel in your list separated by a colon:

```
CWAI1.Channels.Item(1).ChannelString = "1:4"
‘ specifies channel 1, 2, 3, and 4
```

You can also specify a reverse list of consecutive channels:

```
CWAI1.Channels.Item(1).ChannelString = "6:3"
' specifies channels 6, 5, 4, and 3
```

☞ **Note:** ***Certain DAQ devices require that a multiple channel acquisition use a reverse list of consecutive channels ending with channel 0. These devices include all 500-, 700-, and 1200-series devices, as well as the Lab and LPM series cards:***

```
CWAI1.Channels(1).Item.ChannelString = "3:0"
```

You can specify non-consecutive channels in a channel string by listing each channel separated by commas:

```
CWAIPoint1.Channels.Item(1).ChannelString = "0,1,3,5"
```

SCXI Channel Strings

You can use the ComponentWorks DAQ control with SCXI signal condition hardware by using a different channel string to specify your channels.

To configure channels on a SCXI module set the `Device` property of the control to the number of the DAQ board directly or indirectly connected to the desired SCXI module. The channel string(s) of your controls include information about the DAQ device channel, SCXI chassis number, SCXI module number and SCXI channel number. The string has the following format:

```
oba!scx!mdy!z
```

In the SCXI channel string, `a` represents the DAQ device (onboard) channel used for the acquisition, `x` represents the chassis number, `y` the module number, and `z` the channel number on the SCXI module. The onboard channel number is usually one less than the chassis number:

```
CWAIPoint1.Channels(1).Item.ChannelString =
"ob0!sc1!md1!0"
' specifies channel 0 on module 1 in chassis 1
```

Multiple channels on a SCXI module can be specified in a consecutive list:

```
CWAI1.Channels(1).ChannelString = "ob0!sc1!md3!1:5"
' specifies channel 1 through 5 on module 3 in chassis 1
```

Other combinations are listed in the following table:

String Syntax	Description
ob0!sc1!md2!5	channel 5 on module 2 of SCXI chassis 1 read through onboard channel 0
ob0!sc1!md2!0:7	channels 0 through 7 on module 2 read through onboard channel 0
ob1!sc2!md1!20:24	channels 20 through 24 of module 1 on chassis 2 read through onboard channel 1

ExceptionOnError and ErrorEventMask

DAQ controls handle error checking in a number of different ways. By default, each DAQ control generates an exception that is handled by your programming environment when an error occurs. You can disable the generation of exceptions using the `ExceptionOnError` property of each DAQ control. If exceptions are disabled, each call to a DAQ control method returns an error code. If the code is equal to zero, everything completed normally. If the value is non-zero, either a warning or error occurred and the condition should be handled by the application. Consult the section on error handling in Chapter 8, *Building Advanced Applications* for more information.

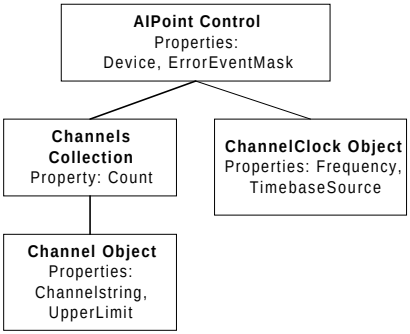
Another type of error notification is the generation of error and warning events in response to error conditions. Each event calls a corresponding event handler routine that processes the error information. You use the `ErrorEventMask` property on each DAQ control to limit the error and warning event generation to specific operations (contexts) of the DAQ controls. For example, by default the AI control generates an error event only during the following contexts: `cwaiReadingData`, `cwaiReadingDataContinuous`, `cwaiReadingDataSWAnalog`. These three are asynchronous contexts, which means the AI control is in the process of acquiring data and returning it to the application. Other contexts, such as `cwaiStartingAcquisition` or `cwaiConfiguringChannels` do not generate error events by default. To select which contexts generate error events, add the values of the `CWAIErrorContexts` constants and assign the sum to the `ErrorEventMask` property. For example:

```
CWAI1.ErrorEventMask = cwaiReadingData +
cwaiReadingDataContinuous + cwaiReadingDataSWAnalog +
cwaiStartingAcquisition
```

Error handling is discussed in more detail in Chapter 8, *Building Advanced Applications.*

AIPoint Control—Single Point Analog Input

Use the AIPoint control to acquire one point of data from one or more analog input channels at a time. This control is used for monitoring slowly changing processes such as temperature. After you set the properties of the control, the application can acquire a single scan of data using a simple method call to the AIPoint control. A scan is defined as an acquisition of one point from each channel in the channel list.



The object hierarchy of the AIPoint control is fairly simple, containing a channels collection with channel objects and a channel clock object.

AIPoint Object

In addition to the default properties of each DAQ control, the AIPoint object has one more property, `ReturnDataType`, which determines whether the acquired data is returned to the application as voltage data, binary values, or both.

The AIPoint object has two methods, `SingleRead` and `Reset`. The `SingleRead` method performs a single scan using the values set in the control properties. The `SingleRead` method has three variant parameters: `data` and an optional parameter, `TimeLimit`, which you use to specify the time limit for the acquisition:

```
Dim data as Variant
CWAIPoint1.SingleRead data, 1.0
```

The variable passed to `data` must be passed as a variant and will be assigned the values to return. The data is returned in the format specified by the `ReturnDataType` property as a one or two dimensional array.

When the `SingleRead` method is called, the hardware is configured using the values set in the control properties. This configuration is only done when necessary, such as calling `SingleRead` the first time or after changing any of the properties. You can also manually unconfigure the control using the `Reset` method. The control then configures the hardware on the next acquisition:

```
CWAIPoint1.Reset
```

Channels Collection

Use the channels collection on analog input and analog output DAQ controls. This collection contains the individual channel objects of a control that determine which channels on the hardware are used by the control. The collection has a read-only property `Count`, which returns the number of channel objects in the collection:

```
NumChanObjects = CWAIPoint1.Channels.Count
```

You can read the read-only property `NChannels` after a control has been configured. This returns the total number of channels used by the control. The value returned from this property is only valid when the control is configured, which you do using the `Configure` method or `SingleRead` method of the respective control, without calling `Reset`:

```
NumChannels = CWAIPoint1.Channels.NChannels
```

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

```
CWAIPoint1.Channels.Item(2)
```

In this way you can programmatically change properties of individual channel objects. There are also several other methods on the channels collection that you can use to modify the number of channel objects in the collection. The `RemoveAll` method clears the collection of all

channel objects. Use the `Remove` method to delete individual channel objects. The `Add` method adds a new channel object to the collection:

```
CWAIPoint1.Channels.RemoveAll
CWAIPoint1.Channels.Remove 1
CWAIPoint1.Channels.Add "1", 10, -10, cwaiDIFF, cwaiDC
```

Channel Object

Each channel object contains information about one or more channels used by a DAQ control. The individual channel object contains properties such as `ChannelString`, `InputMode`, `UpperLimit`, and `LowerLimit`. For example, the `ChannelString` property specifies which channels are affected by the channel object, while the remaining properties determine how the channels are used. You can read and set these properties programmatically:

```
CWAIPoint1.Channels.Item(1).ChannelString = “0,1”
MaxVolts = CWAIPoint1.Channels.Item(1).UpperLimit
```

ChannelClock Object

The channel clock object determines the timing the AIPoint control uses in the actual analog-to-digital conversions within a scan. Use it to increase the delay between the acquisitions of different channels or to synchronize conversion(s) with an external signal. It directly affects the conversions by allowing you to select either an internal source and frequency or an external source and exact description of the signal source, such as an I/O pin or RTSI pin.

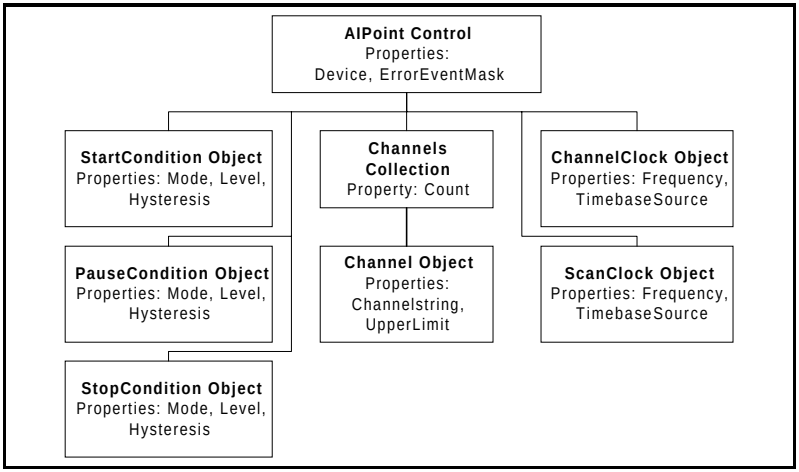
By default ComponentWorks picks your channel clock settings, which should work for most applications. You usually configure the channel clock in the property pages, but you may also change its settings from your program. For example, to switch to frequency mode and change the frequency setting use the following code:

```
CWAIPoint1.ChannelClock.ClockSourceType =
  cwaiInternalCS
CWAIPoint1.ChannelClock.Frequency = 10000
```

Consult the online reference for other properties of the channel clock object.

AI Control—Waveform Analog Input

Use the AI control to perform waveform analog input operations including single shot and continuous acquisitions. With it you can acquire data from one or more channels at a time and configure for many different modes, such as start and stop triggers, pause conditions, and different channel and scan clocks. Use this control for any application that requires fast acquisition of multiple points per channel, such as frequency analysis. After the properties of the control are set, the application can perform acquisitions using a number of simple method calls.



The object hierarchy of the AI control separates the different functionality of the control into individual objects. The channels collection and channel objects specify the channels and channel parameters used for the acquisition process. The condition objects control when an acquisition is performed, such as triggering. The clock objects control the rate of the acquisition.

AI Object

Along with the default properties of the DAQ controls, the AI object has other properties. One is `ReturnDataType`, which determines whether the acquired data is returned to the application as voltage data, binary values, or both. The `NScans` property (number of scans to acquire) specifies the number of scans acquired in a single-shot acquisition, or the number of scans returned at a time in a multiple channel acquisition:

```
CWAI1.NScans = 5000
```

If the `UseDefaultBufferSize` property is set to `False`, the `NScansPerBuffer` property determines the size of the acquisition buffer; otherwise ComponentWorks automatically selects the buffer size.

The AI control uses the *Channels* collection and *Channel* object in the same manner as the AIPoint control. See the *AIPoint* section earlier in this chapter for more information on these objects. Consult the online help file for more information on the individual properties, methods or events of the AI objects or any of its underlying objects.

Methods and Events

The AI control has a number of simple methods for running the different acquisition processes. The normal and recommended acquisition type is an asynchronous acquisition which is controlled using four different method calls and gives you the most flexibility and control over the acquisition. Alternatively, in environments that do not support event handler routines, you can use an additional method to perform a synchronous acquisition.

Asynchronous Acquisition

The methods to perform an asynchronous acquisition are `Configure`, `Start`, `Stop` and `Reset`. Use these methods to the control the acquisition. These methods do not require any parameters, as acquisition parameters are set through the control properties. Use the `Configure` method to configure the DAQ driver and hardware with the acquisition parameters. `Configure` must be called before the `Start` method. Use the `Start` method to start the acquisition that proceeds according to the start condition object described later. Use the `Stop` method only during a continuous acquisition to stop such an acquisition. Use the `Reset` method to unconfigure the AI control and free any resources reserved during configuration. You must also call the

`Configure` method after you change any of the control properties before they will take effect, and after you call the `Reset` method before restarting an acquisition:

```
Private Sub StartAcquisition_Click()
    CWAI1.Configure
    CWAI1.Start
End Sub
Private Sub StopAcquisition_Click()
    CWAI1.Stop
End Sub
Private Sub ResetAcquisition_Click()
    CWAI1.Reset
End Sub
```

The AI control returns data from an asynchronous acquisition by generating an `AcquiredData` event. The acquired data is returned as arrays, `Voltages` and/or `BinaryCodes`, passed to the event handler. You can process the data inside the event handler by displaying it on a graph or writing it to file:

```
Private Sub CWAI1_AcquiredData(Voltages As Variant,
  BinaryCodes As Variant)
    CWGraph1.PlotY Voltages
End Sub
```

Synchronous Acquisition

Certain programming environments do not support event handler functions, and are therefore not suited for running an asynchronous acquisition. In such cases the AI control can perform a synchronous acquisition using the `AcquireData` method. The `AcquireData` method requires you to pass in two variables for the voltage and binary data which are used to return the acquired data at the completion of the acquisition. You must call the `Configure` method before calling `AcquireData`, and you cannot run a continuous acquisition using this method. Because the `AcquireData` method takes control of the program until the acquisition is completed, you can also specify a timeout parameter in seconds that forces the method to return in the time limit specified:

```
Private Sub RunAcquisition_Click()
   Dim Voltages As Variant
   Dim BinaryCodes As Variant
   CWAI1.Configure
   CWAI1.AcquireData Voltages, BinaryCodes, 5
   'timeout is 5 seconds
   CWGraph1.PlotY Voltages
End Sub
```

Error Handling

The AI control also has `DAQError` and `DAQWarning` events which can be used for error handling:

```
Private Sub CWAI1_DAQError(ByVal StatusCode As Long,
  ByVal ContextID As Long, ByVal ContextDescription As
  String)
   MsgBox "DAQ Error: " + CStr(StatusCode)
End Sub
```

ScanClock and ChannelClock Objects

The AI control contains both a scan clock and channel clock to specify the scan rate and interchannel delay. These two settings apply if you acquire multiple points of data from more than one channel. In this type of operation, the data acquisition device performs repeated scans, in which one scan is an acquisition of one data point from each channel in the channel list. The timing within one scan is called the *interchannel delay* and is set in the channel clock object. The rate at which scans are acquired is called the *scan rate*, and is set in the scan clock object. The effective acquisition rate per channel (the rate at which points on one channel are acquired) is also the scan rate.

The channel clock object is described in the AIPoint control section. This object normally selects the interchannel delay automatically.

The scan clock object is critical to the operation of the AI control and you must set it for most applications. In common operations, specify an internal frequency to use and set the acquisition (scan) rate in the property pages. More complex operations can include specifying an external source for the scan clock to synchronize the acquisition with another process. You can make these settings in the property pages or

programmatically. After the control is configured, you can read back the actual frequency or period used for the acquisition:

```
CWAI1.ScanClock.Frequency = 200
CWAI1.Configure
ScanPeriod = CWAI1.ScanClock.ActualPeriod
```

Consult the online reference manual for the other properties of the scan clock object.

StartCondition, PauseCondition and StopCondition Objects

The start, pause and stop condition objects control when an acquisition starts, pauses and stops. The main property of the condition object is `Type`, which sets the overall operation of the object. The value of the `Type` property determines which of the remaining properties on the condition object are used.

Certain condition types are only supported by specific hardware. Check that your data acquisition device supports the desired operation. For example, all hardware analog conditions require specific analog trigger circuitry on the acquisition device.

The start condition object controls when an acquisition is started. By default (`Type` set to `cwaiNoActiveCondition`) the acquisition is started immediately after the corresponding method call:

```
CWAI1.StartCondition.Type = cwaiNoActiveCondition
```

You can also set the start condition to start the acquisition on a digital or analog trigger. In such cases, the hardware is set to start on the corresponding software call, but actual conversions do not start until the digital or analog trigger arrives. You can set trigger conditions either in the property pages or programmatically:

```
CWAI1.StartCondition.Type = cwaiHWAnalog
CWAI1.StartCondition.Level = 5
CWAI1.StartCondition.Hysteresis = 0.1
CWAI1.StartCondition.Source = 1
CWAI1.StartCondition.Mode = cwaiRising
```

The pause condition object controls when an ongoing acquisition is paused. This may be done in response to an external digital or analog signal, and can be used with a limited number of data acquisition

devices. The `Type` property selects among `None`, `Hardware Digital Gate`, and `Hardware Analog Gate` settings. The remaining properties of the pause condition are similar to the start condition and select the specific pause conditions. You can pause the acquisition while above or below a specific analog level (high and low for digital) or while inside or outside a specific analog window:

```
CWAI1.PauseCondition.Type = cwaiHWAnalog
CWAI1.PauseCondition.Mode = cwaiInside
```

The stop condition object controls when the acquisition is stopped. The default mode is to stop after the acquisition buffer, set by the `NScans` property on the AI control, has been filled once. You can also select to run a continuous acquisition so the acquisition stops only on a user command or when an error occurs. Other advanced options are stopping on a hardware digital or analog signal, or a software analog condition (single shot or continuous). These last three types also support pretrigger scans, which means you can specify to acquire a number of points before the stop condition and the remainder after. The remaining properties are similar to the start and pause condition objects.

The software analog trigger type on the stop condition is added to support analog triggering on devices that do not have an explicit hardware analog trigger circuit. In this mode data is continuously acquired from the data acquisition device, but returned only when it matches the specified conditions. This mode behaves similarly to a hardware analog start trigger, and you can run it as either a one-shot or continuous acquisition. The continuous software analog trigger makes it easy for you to duplicate the operation of an oscilloscope in your application.

Tutorial: Using the AIPoint and AI DAQ controls

This tutorial shows you an example of using the AIPoint and AI controls in a simple program to acquire some data. New data acquisition programmers can easily incorporate these controls, which also offer advanced features for more experienced programmers. There are a lot of similarities between these and all other DAQ controls, so as you become familiar with each DAQ control, you should be more at ease using any of them.

This example shows how to acquire one scan of data from several channels using the AIPoint controls, as well as how to perform a simple

waveform acquisition using the AI control, and display the data on a graph.

This tutorial uses Visual Basic syntax, but the discussion is 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 on implementing any particular step. You can also consult the Tutorial examples installed with ComponentWorks for a completed version of this example in several different programming environments.

Designing the Form

1. Open a new project and form. If you are working in Visual C++, select a dialog based application and name your project `AIExample`.
2. Load the ComponentWorks user interface controls (specifically, the graph, button, and numeric edit box) and the data acquisition controls (specifically, the AIPoint and AI controls) into your programming environment. Consult the chapter discussing your environment if you are not familiar with this operation.

3. Place an AIPoint and AI control on your form. You will configure their properties in the next section.
4. Place a ComponentWorks graph control on the form. Keep its default name `CWGraph1`.

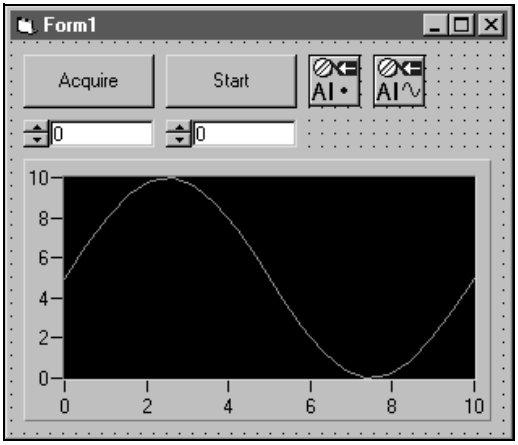
   Most of the settings of the graph in its property sheet should be self-explanatory. For more information about the advanced features, refer to Chapter 8, *Building Advanced Applications*, and the online help manual.

5. Place two ComponentWorks numeric edit boxes on the form. Keep their default names, `CWNumEdit1` and `CWNumEdit2`.

6. Place two ComponentWorks buttons on the form (toolbar icon shown at left; modify in properties sheet to make a button on your form). Change their `Name` property to `Acquire` and `Start`. This is done in the default property sheets in Visual Basic and Delphi or the custom property sheets in Visual C++.

In the custom property sheets, change each of their style to `Command Button`. Also, in the **Button** tab of the property sheets, change the **On Text** and **Off Text** to `Acquire` for the first button and `Start` for the second button.

Your form should look similar to the one shown below:



Setting the DAQ Properties

You normally configure the default property values of the different controls before you develop your program code. When using the DAQ controls, most or all properties are set at design-time and are not changed during program execution. The program is only used to start and stop the acquisition processes. If necessary, you can edit the properties of the DAQ controls at run-time.

1. Open the custom property pages for the AIPoint control on the form by right-clicking on the control and selecting **Properties…**. In the **General** tab select your data acquisition device from the **Device** combobox. In the **Channel List** tab, add channels `0` and `1` to the channel list. Do this by entering `0,1` in the **Channels** field and then clicking the **Add>>** button. The new channels will be added to the channel list on the right side. Close the AIPoint property pages. If necessary, reverse the order of the channels for your DAQ device. See the *Channel String* section for more information.

2. Open the custom property pages for the AI control on the form. In the **General** tab select your data acquisition device from the **Device** combobox. In the **Channel List** tab, add channel 1 to the channel list. Do this by entering `1` in the **Channels** field and then clicking the **Add**>> button. The new channels are added to the channel list on the right side. In the **Clocks** tab, change the scan rate (Scans/second) to `1000`. Close the AI property pages.

Developing the Code

Next, you develop the code so that data is acquired and displayed in response to the buttons.

1. For the **Acquire** button, define an event handler routine that will be called when the button is pressed. In the event handler, acquire one scan of data (one point each from channel 0 and 1) and display the two points in the numeric edit boxes.

   Generate the event handler routine for the `Click` event of the `Acquire` button. In the event handler declares a variable as a `Variant`. Pass this variable to the `SingleRead` method of the AIPoint control. Then display the data returned in the first variable in the numeric edit boxes. Add the following code to the `Acquire_Click` subroutine. In Visual C++ remember to generate member variables for any controls referenced in the program. See the `Tutorial` folder for Visual C++ and Delphi code examples.

   ```
   Private Sub Acquire_Click()
      Dim Volts As Variant
      CWAIPoint1.SingleRead Volts, 1
      CWNumEdit1.Value = Volts(0)
      CWNumEdit2.Value = Volts(1)
   End Sub
   ```

   This code acquires a scan from channels 0 and 1 using the `SingleRead` method and returns the data in a one dimensional array (`Volts`). The two values are copied to the numeric edit boxes to be displayed.

2. For the **Start** button, define an event handler routine that will be called when the button is pressed. In the event handler, start an asynchronous acquisition on the AI control. Because the acquisition is asynchronous, the program regains control immediately after the acquisition is started, while the acquisition continues to run in the background.

Generate the event handler routine for the `Click` event of the **Start** button. In the event handler, call the `Configure` and `Start` methods of the AI control. Add the following code to the `Start_Click` subroutine. In Visual C++, remember to generate member variables for any controls referenced in the program. See the `Tutorial` folder for Visual C++ and Delphi code examples.

```
Private Sub Start_Click
    CWAI1.Configure
    CWAI1.Start
End Sub
```

This code starts the acquisition and then returns control to the program.

3. The AI control fires an `AcquiredData` event when it is ready to return the data it has acquired. You need to generate the corresponding event handler to receive and process the data. In this example, plot the data on a graph. Use the `PlotY` method of the Graph control to display the returned `Voltages` array.

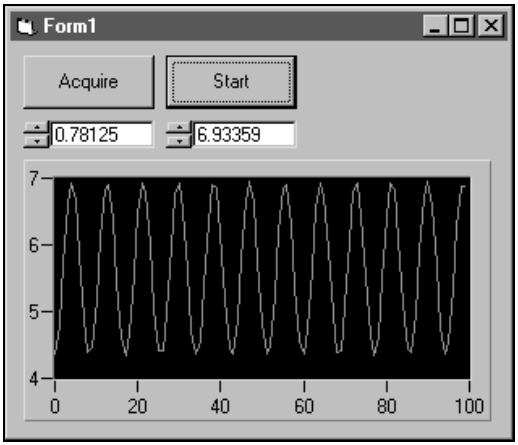
   Generate the event handler for the `AcquiredData` event of the AI control and add the following line of code:

```
Private Sub CWAI1_AcquiredData(Voltages As Variant,
BinaryCodes As Variant)
    CWGraph1.PlotY Voltages, 0, 1, 1
    CWAI1.Reset
End Sub
```

4. Save the project and form as `AIExample`.

Testing Your Program

Run and test the program. Click on the `Acquire` and `Start` buttons to perform the acquisitions. Your application should look similar to the following figure. The data displayed on the graph will depend on the signal connected to your data acquisition board.

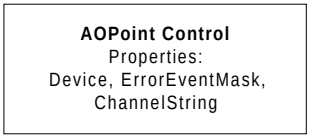


You can further enhance the waveform acquisition performed in this example by defining more properties of the AI control. For example, you can perform a continuous acquisition by setting the **Type** property of the AI control stop condition object to `Continuous`. Do this in the **Conditions** tab of the AI control property pages. In continuous mode, the acquisition continues and repeatedly returns the specified number of points in the `DataAcquired` event. The AI control has a `Stop` method you can call to stop a continuous acquisition.

AOPoint Control—Single Point Analog Output

Use the AOPoint control to update one or more analog output channels with new values. Use the control in applications such as slow process control systems for setting a control output such as the power of a heater or throughput of a valve. After you set the properties of the control, the

application can update the configured channels using a simple method call to the AOPoint control.



The AOPoint control contains no other objects and therefore has no hierarchy. All properties of the control are part of the top level object.

AOPoint Object

In addition to the default properties of each DAQ control, the AOPoint object has several unique properties that control the operation of the control.

The `ChannelString` property, together with the `UpperLimit`, `LowerLimit`, `Reference Source` and `ChannelType` properties, control the output from the AOPoint control. You normally configure these properties through the property pages but they may also be set programmatically:

```
CWAOPoint1.ChannelString = "0,1"
CWAOPoint1.UpperLimit = 10.0
```

Methods

The AOPoint object has two methods, `SingleWrite` and `Reset`. The `SingleWrite` method performs a single update on all the channels configured for the control. The `SingleWrite` method has two variant parameters, `Values` and `Scaled`. The `Values` parameter is used to pass the analog values to the method that will be generated by the analog output channels. The optional `Scaled` parameter allows you to specify whether the analog values are passed as voltage or binary data. By default, the information is interpreted as voltage data:

```
‘ update one channel
CWAOPoint1.ChannelString = "0"
CWAOPoint1.SingleWrite 5.0, True
‘ update two channels
Dim VoltsArrayData(0 to 1)
CWAOPoint2.ChannelString = "0,1"
```

```
VoltsArrayData(0) = 3.2
VoltsArrayData(1) = 6.5
CWAOPoint2.SingleWrite VoltsArrayData, True
```

You can send the output data to the `SingleWrite` method as a scalar value for updating one channel, or as a one dimensional array of values for more than one channel.

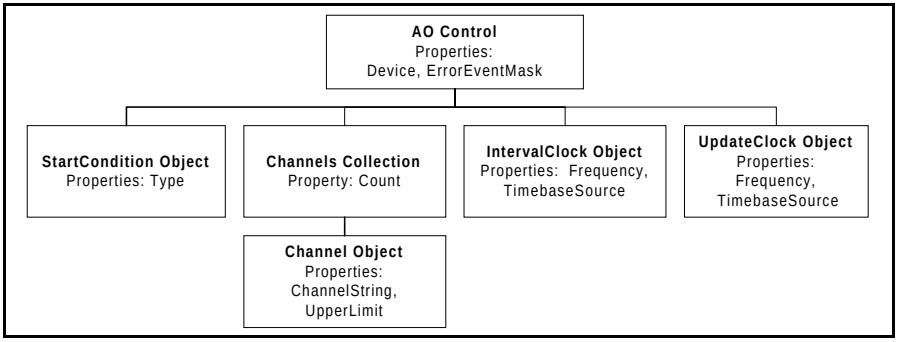
When you call the `SingleRead` method, the hardware is configured using the values set in the control properties. This configuration is done only when necessary, such as when `SingleRead` is called the first time or after changing any of the properties. You can also manually unconfigure the control using the `Reset` method, which causes the control to configure the hardware on the next output:

```
CWAOPoint1.Reset
```

Consult the online reference manual for more information on the AOPoint control.

AO Control—Waveform Analog Output

Use the AO control to perform waveform generation operations from one or more analog output channels on a data acquisition device. The waveform generation can be run in a continuous or finite mode. You can configure properties on the control such as the channels used for the waveform generation, the frequency (update clock) used, and the start condition or trigger. This control is used for different types of applications that require dynamic analog signals such as testing of analog devices. After you set the properties of the control, the application can perform output operations using a number of simple method calls.



The object hierarchy of the AO control separates the different functionality of the control into individual objects. The channels collection and channel objects specify the channels and channel parameters used for the output process. The start condition object controls the start of the operation, while the clock objects control the rate of the output.

AO Object

In addition to the default properties of the DAQ controls, the AO Object has several other properties. The generated waveform data is stored in a buffer in memory. The `NUpdates` property specifies how many updates will be stored in the buffer. The `Infinite` property is a boolean that select whether a waveform generation runs continuously or stops after a finite number of buffer outputs. In a finite generation the `NIterations` property specifies how often the data in the buffer is generated:

```
CWAO1.Infinite = False
CWAO1.NIterations = 10
```

The data buffer is usually stored in the memory of the computer. If you are using some specific data acquisition devices, you may also be able to store the data in memory on the DAQ device itself, which enables faster output of the waveform. You can select this type of operation using the `AllocationMode` property.

The AO control can generate events to notify you of the progress of the output operation while a waveform generation is running. The frequency at which these events are generated is set by the `ProgressInterval` property, and is specified in number of updates.

The AO control uses the `Channels` collection and `Channel` object in the same manner as the AIPoint control. See the AIPoint description for information on these objects. Consult the online help file for detailed information on all the properties of the AO channel object. The reference file also contains more information on the individual properties, methods, and events of the AO objects or any of its other underlying objects.

Methods and Events

The AO control has a number of methods for performing waveform generation operations. The methods used with the AO controls are `Configure`, `Write`, `Start`, and `Reset`. All are used to control the output operations. Only the `Write` method requires parameters.

Use the `Configure` method to configure the DAQ driver and hardware with the operation parameters. Use the `Write` method to write an array of voltage data to the buffer in memory before it can be generated from the analog output channels. All other parameters are set through the properties of the control.

`Configure` must be called before the `Write` and `Start` methods. Use the `Start` method to start the waveform generation which proceeds indefinitely or stops after the specified number of buffer generations. Use the `Reset` method to stop a continuous generation, unconfigure the AO control, and free any resources reserved during configuration.

You must call the `Configure` method again after any of the control properties are changed before they will take effect, and after the `Reset` method is used, before restarting a waveform generation:

```
Private Sub Start_Click()
    Dim WaveData(0 to 99) As Double
    For i = 0 To 99
        WaveData(i) = Sin(i / 100 * 6.28)
    Next i
    CWAO1.Configure
    CWAO1.Write WaveData
    CWAO1.Start
End Sub
Private Sub Stop_Click()
    CWAO1.Reset
End Sub
```

The AO control fires the `Progress` event while the output operation is running. The event notifies your application that a specific number of updates has been performed on the output. The frequency at which the event is generated is set in the `ProgressInterval` property. You can

use the event to update your front panel as to the progress of the waveform generation, or for several other purposes:

```
Private Sub CWAO1_Progress(ByVal ScansGenerated As Long)
   Text1.Text = ScansGenerated
End Sub
```

The AO control also has `DAQError` and `DAQWarning` events which can be used for error handling:

```
Private Sub CWAO1_DAQError(ByVal StatusCode As Long,
  ByVal ContextID As Long, ByVal ContextDescription As
  String)
   MsgBox "DAQ Error: " + CStr(StatusCode)
End Sub
```

UpdateClock and IntervalClock Objects

The AO control contains both an update clock and interval clock to specify the update rate and optional interval delay. Only the update clock is commonly used. It specifies the rate at which data points are generated by the boards. Because each output channel has its own digital-to-analog (DAC) converter, there is no delay between updates from different channels and all channels are updated simultaneously.

You can set the update clock either in the property pages or programmatically to use an internal frequency, a signal from another component on the DAQ device or an external signal. The choices of update clock sources will depend on the specific DAQ device used:

```
CWAO1.UpdateClock.ClockSourceType = cwaoInternalCS
CWAO1.UpdateClock.Frequency = 10000
```

The property page for the update clock gives a good overview of the different possible settings for the properties of the update clock object. You can also select any of these settings programmatically.

The interval clock is used in a very limited number of applications and is supported on only a small number of DAQ devices. It is used in a waveform generation, when the data is stored completely in FIFO memory on the DAQ device, to add a time delay in the output between generations of the waveform data stored in the buffer. For example, this allows you to store one cycle of a sine wave in the buffer and generate repeated cycles of a sine wave with delays between the different cycles.

You can examine the properties of the interval clock object and their possible settings in its property page.

Consult the online reference manual for more information on the update and interval clock objects.

StartCondition Object

Use the start condition object to specify when the waveform generation is started. In most cases, the generation starts immediately after the `Start` method is called. You can use the start condition object on some DAQ devices to trigger the generation in response to an external signal or a signal coming from another component on the device. You can use this functionality to synchronize a waveform generation with an input operation or other external process.

The main property of the start condition object is `Type`, which selects the overall operation of the object. The value of the `Type` property determines which of the remaining properties on the condition object are used. If the start condition object is set to use another signal as the start trigger, the `Source` property specifies the source of the signal. Other properties may be used to specify trigger parameters, such as the slope of the signal to trigger on, or the conditions of an analog trigger.

You can also set the values of the different start condition properties programmatically:

```
CWAO1.StartCondition.Type = cwaiHWAnalog
CWAO1.StartCondition.Source = "PFI0"
CWAO1.StartCondition.Level = 5
CWAO1.StartCondition.Mode = cwaoRising
```

If you use a start trigger, the hardware is set to start the waveform generation after the `Start` method is called. The actual output conversions do not start until the specified trigger signal arrives.

Tutorial: Using the AOPoint control

This tutorial develops a simple program using the AOPoint control. To use this example, your DAQ device must have one or more analog output channels. In addition, you will use an analog input channel of your DAQ device to read the output voltage. If your device does not have analog inputs, you can use an external voltmeter or oscilloscope.

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 `AOPoint`.
2. Load the ComponentWorks user interface controls (specifically, the graph and slide) and the data acquisition controls (specifically, the AOPoint and AI controls) into your programming environment. Consult the chapter discussing your environment if you are not familiar with this operation.

3. Place a ComponentWorks button on the form. In the custom property sheet change the button style to `Toggle Button`, its `On Text` to `Stop`, and its `Off Text` to `Start`. In the default property sheet, change the `Name` to `Start`.

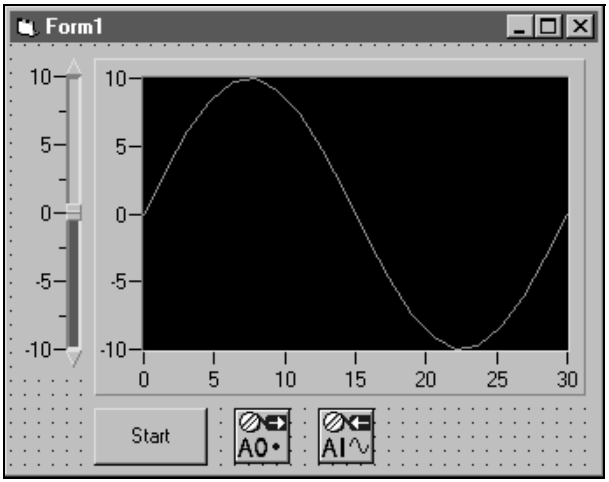
4. Place a ComponentWorks slide control on the form and change its name to `UpdateValue`. In the custom property sheet under the **Numeric** tab, change its `Min` and `Max` properties to `-10` and `10`.

5. Place a ComponentWorks graph on the form. In the custom property sheet **Axis** tab, set the X-axis range to `0` to `30`, and the Y-axis range to `-10` to `10`.

6. If your DAQ device has analog input channels, place a DAQ AI control on the form. In the property sheet, select the device and channel you want to use. Also set the **Number of scans to acquire** to `1`, the **Scan Clock** to `Internal 10 scans/second`, and the **Stop Condition** to `Continuous`.

7. Place a DAQ AOPoint control on the form. In the custom property sheet of the AOPoint control, select the device and output channel you want to use.

8. Your form should now look similar to the one shown below:



Developing the Code

The next step is to add the necessary code to generate the analog output and acquire the signal using your analog input channel. If you do not have an analog input channel on your DAQ device, leave out all the calls relating to the analog input (starting with CWAI) and use an external voltmeter to measure your analog output when you run the program.

The program updates the selected analog output channel with the value of the slide when the slide pointer is moved. Use the **Start**/**Stop** button to start and stop a continuous analog input operation that measures the voltage being generated.

1. Use the `PointerValueChanged` event of the slide control to detect any changes in the slide control and update the analog output value. Use the `SingleWrite` method of the AOPoint control to update the output.

   Create the event handler routine for the `PointerValueChanged` event of the slide control and add the following code to it. In Visual C++, you first need to create a member variable for the AOPoint control using the ClassWizard.

```
Private Sub UpdateValue_PointerValueChanged(ByVal
  Pointer As Long, ByVal Value As Variant)
    lerr = AOPoint1.SingleWrite(UpdateValue.Value,
    True)
End Sub
```

2. To monitor the voltage being generated, run a continuous analog input operation and chart the acquired voltage on a graph. Start and stop the acquisition using the **Start**/**Stop** button in response to its `ValueChanged` event.

   Create the event handler routine for the `ValueChanged` event of the button and add the following code to it. In Visual C++ you first need to create a member variable for the AI control using the ClassWizard. Depending on the state (`Value`) of the button, the event handler either starts or stops the acquisition.

```
Private Sub Start_ValueChanged(ByVal Value As
Boolean)
    If Value Then
        CWAI1.Configure
        CWAI1.Start
    Else
        CWAI1.Stop
    End If
End Sub
```

3. While the acquisition is running continuously, it returns data through the `AcquiredData` event of the AI control. In this event handler routine, chart the data on the graph. The `ChartY` method of the graph acts similar to the `PlotY` method, except that data is appended to the data already displayed on the graph.

   Create the event handler routine for the `AcquiredData` event of the AI control and add the following code to it. In Visual C++ you first need to create a member variable for the graph control using the ClassWizard.

```
Private Sub CWAI1_AcquiredData(Voltages As Variant,
  BinaryCodes As Variant)
    CWGraph1.ChartY Voltages, 1, True
End Sub
```

4. You normally reset hardware operations before quitting the application. This means stopping any ongoing acquisition and resetting the analog output to 0 volts.

   Add the following code to an event handler routine that you call when your application is terminated. The exact name of the event varies depending on your programming environment. In Visual Basic, you can use the `Terminate` event of the `Form` object.

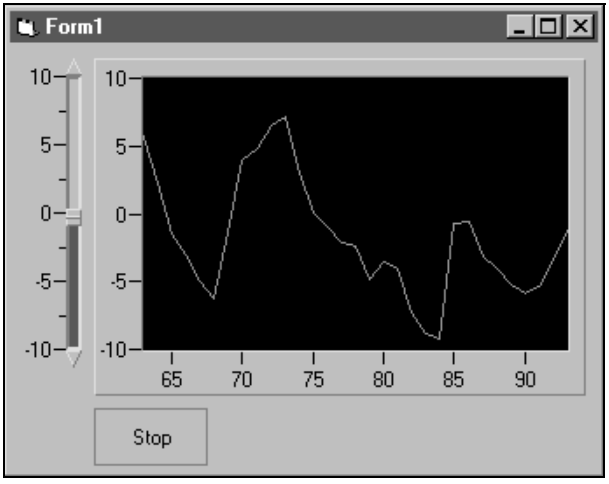
```
Private Sub Form_Terminate()
    CWAOPoint1.SingleWrite 0, True
End Sub
```

5. Save your project and form as `AOPoint`.

Testing Your Program

1. Run the program. Remember to physically connect the analog output to your analog input. When you toggle the Start button, the continuous input operation starts and display its measurement on the graph.
2. Change the value of the analog output by moving the slider. When you move the slider, the analog output is updated and you can see the change on the graph.

   Your completed running program should look similar to the one shown below.



3. When you end the program, the analog output is automatically reset to zero volts.

Digital Controls and Hardware

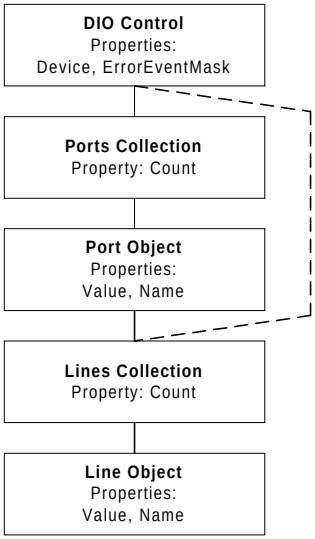
There are three ComponentWorks DAQ controls for performing digital input and output operations. Use the DIO control for both input and output single point operations to update the state of any output lines or read the state of any input lines. Use the DI control to perform buffered digital waveform input operations, and the DO control to perform buffered digital waveform output (pattern generation) operations.

The digital I/O lines on each data acquisition device are grouped into logical units called ports. There are eight lines per port on most devices, although there can be as few as two and up to 32 lines. When referencing digital lines on the different controls, you always specify a port number (starting with zero per device) to select the lines you wish to use. On the DIO control you can also select individual lines of a given port to update or read.

DIO Control—Single Point Digital Input and Output

Use the DIO control to perform single-point updates or reads on the digital lines of a data acquisition device. Typical applications using the DIO control include controlling the state of a physical device such as a valve, relay or LED; or reading the current state of a similar device such as a switch or light gate. You can also use the DIO control to generate slow pulses to activate other parts of your system. After you set the

properties of the DIO control, your program can perform the different operations using simple method calls to the DIO control.



The DIO control consist of a top level object and a set of Ports and Lines collections and objects. The Ports collection contains a number of Port objects which represent the logical ports on the DAQ device selected in the DIO object. Once you select a device in the DIO object, all ports of the device are represented by the control. The Lines collections contains a Line object for each physical digital line on the device. You normally access the Lines collection through one of the port objects, but you also may access it directly from the DIO object. See the following sections for more information on how to reference specific digital lines using these two methods.

Depending on the device you are using, you configure all lines in a given port for input or output operations, or you may be able to configure individual lines of a digital port for input and output operations. Devices that allow for line configuration include all E-Series devices (except the extended ports on the MIO-16DE-10), the PC-TIO-10, and DIO-32HS devices.

DIO Object

In addition to the default properties of each DAQ control, the DIO object contains a `SCXIChannelString` property. If you use the DIO

control with one of the SCXI-116x digital modules, enter a SCXI channel string for this property to select the SCXI module. The string must follow the convention for SCXI channel strings listed at the beginning of this chapter. Each SCXI digital module has only one logical port (port 0) which contains all the digital lines of the module. Therefore the port number is always zero, such as in the following example:

```
CWDIO1.SCXIChannelString = “sc1!md3!0”
```

The DIO object also contains the `Ports` collection, which contains individual `Port` objects. Use the port object to configure the individual ports on a device including programmatic configuration of the direction of the digital lines. Each port object contains a `Lines` collection with `Line` objects that allow you to access the individual digital lines of the DAQ device. You can also to access the lines collection directly from the DIO control.

Normally you configure the direction of individual ports and lines using the property pages of the DIO control. You can do this through an intuitive interface that presents the individual ports and lines to you. In most cases, this is all you do to configure the control.

The DIO, port, and line objects contain a set of common methods to perform operations using the DIO control. These methods are described after the sections outlining the port and line objects.

Ports Collection and Port Object

The Ports collection contains the common properties and methods of a collection such as `Count` and `Item`. Use the `Item` method to access the individual Port objects.

```
CWDIO1.Ports.Item(1)
```

Use the port object to configure the individual digital ports on a DAQ device. Port objects include properties such as `Assignment` and `LineDirection`. Use these two properties to programmatically configure the direction of the port or the lines in the port. The `Assignment` property specifies whether a port is configured for input or output operations or is line configured. In line configuration, use the `LineDirection` property to specify the direction of each individual line in the port. Each bit in the `LineDirection` property corresponds to a digital line. For example, bit 0 corresponds to line 0. A bit value of 1 indicates an output line and a 0 indicates an input line.

```
' Configure port 2 for output
CWDIO1.Ports.Item(2).Assignment = cwdioPortOutput

' Configure port 0, lines 0-3 for output, lines 4-7 for
input, binary 00001111 = decimal 15
CWDIO1.Ports.Item(0).Assignment =
  cwdioPortLineConfigured
CWDIO1.Ports.Item(0).LineDirection = 15
```

☞ **Note:** ***The MIO-16DE-10 is a hybrid DIO board:***

- ***Port 0 is 8 bits wide, and is line configurable.***
- ***Port 1 does not exist.***
- ***Ports 2, 3, and 4 are each 8 bits, and are port configurable.***

The port object has another property `Value` and several methods, all of which are described in the section, *Common Properties and Methods*. Each port object also contains a reference to a `Lines` collection containing individual `Line` objects. Use these to configure and operate the individual digital lines of the data acquisition device.

Lines Collection and Line Object

The `Lines` collection has the same properties as the `Ports` collection, and allows you to select individual `Line` objects. The line object has a `Name` property which allows you to identify individual lines by name:

```
CWDIO1.Ports.Item(0).Lines.Item(0).Name = "Switch1"
```

The `Assignment` property of the line object is read-only. Use it to check the current direction of a particular line:

```
Line0isOutput =
  CWDIO1.Ports.Item(0).Lines.Item(0).Assignment
```

See the following section for information on the `Value` property and the methods of the line object.

Common Properties and Methods

The DIO, Port and Line objects have a set of common methods (SingleRead, SingleWrite, and Update) to perform the actual input and output operations. In addition the port and line objects include a Value property which you use in conjunction with the Update method.

Use either the SingleRead or SingleWrite method to read or write the current state of the digital I/O lines. SingleRead requires as a parameter a variant which returns the value or values from the read operation. Performing SingleRead on the DIO object returns an array of integers where each array element represents the state of one port. The integer represents the state of the digital lines in a port by bit, where the lowest bit in the integer represents the state of line 0 in the port and so on. For example an integer of value 25 (binary 00011001) would mean the state of lines 0, 3 and 4 are high and all the remaining are low. Performing SingleRead on a port object returns a single integer representing the state of the port, and SingleRead on the line object return a simple boolean indicating the state of the line:

```
Dim vData As Variant
CWDIO1.SingleRead vData
‘ returns an array of integers
CWDIO1.Ports.Item(0).SingleRead vData
‘ returns an integer
CWDIO1.Ports.Item(0).Lines.Item(0).SingleRead vData
‘ returns a boolean
```

You can also access the Lines collection directly from the DIO object. If you have assigned a name to a specific digital line you can use it as reference:

```
CWDIO1.Lines.Item("Switch1").SingleRead bState
CWDIO1.Lines.Item(14).SingleRead bState
```

To write to the digital output lines, use the SingleWrite method on either the DIO, Port or Line object. This method requires you to write a parameter containing the data to the digital lines. The form of the parameters is the same as it would be returned from a corresponding SingleRead call:

```
Dim vData As Variant
' an array of integers to write to the DIO object
vData = Array(0, 0, 0)
CWDIO1.SingleWrite vData
' a single integer to write to a port
vData = 0
CWDIO1.Ports.Item(0).SingleWrite vData
' a boolean to write to a line
vData = False
CWDIO1.Ports.Item(0).Lines.Item(0).SingleWrite vData
```

The alternative to reading and writing using the digital lines is using the `Value` property of the port or line object and the `Update` method on the `DIO`, `Port` or `Line` object.

In the output direction, use the `Value` property to specify the value of a port or line, which will be written to the hardware the next time you call the `Update` method. On the input side the `Value` property represents the state of the hardware lines the last time the `Update` method was called.

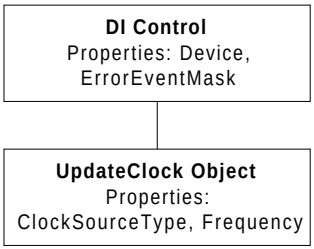
☞ **Note:** ***The*** `Value` ***property does not represent the current state of the digital lines.***

The `Value` property represents a cached state of either the last input operation or the data that will be written on the next output operation. Use the `Update` method to synchronize the cached values with the hardware. This allows you to assign new values to individual output lines, and update all of them at once. You can cache the current state of a set of input lines once and then read the values from the individual lines:

```
' Input Operation
CWDIO1.Update
portVal = CWDIO1.Ports.Item(0).Value
lineVal = CWDIO1.Ports.Item(0).Lines.Item(4).Value
' Output Operation
CWDIO1.Ports.Item(0).Lines.Item(6).Value = False
CWDIO1.Ports.Item(0).Value = newPortVal
CWDIO1.Update
```

DI Control—Buffered Waveform Digital Input

Use the DI control to perform buffered waveform digital input operations. This allows you to acquire data from your digital inputs quickly, at a rate specified by an external signal or internal frequency. Typical applications using the DI control include transferring digital data from an external device or monitoring of a quickly changing system. Advanced applications may include network analyzers. After the properties of the DI control are set, your program can perform the digital acquisition using a couple simple method calls to the DI control.



The DI control consists of a top level object and `UpdateClock` object. Most properties that determine the actions of the DI control are set on the control itself, while the update clock object determines the source for the rate of acquisition.

The capabilities of the DI object depend on the data acquisition device you are using. Many DAQ devices do not support buffered digital acquisition, and the DI control will not work with these devices. Other devices support only single buffered acquisition. For these devices you will not be able to use the continuous mode. Most of these devices also require an external signal to set the acquisition rate and you can not just specify an internal frequency. Devices in the DIO-32 series support both continuous acquisitions and internal frequency sources, and can take advantage of the full range of the DI control features. Check the hardware manual of your data acquisition device for detailed information on its capabilities.

DI Object

After the default DAQ properties of the DI object (such as `Device`) are set, the most important property to configure is `ChannelString`. You specify what digital ports are used by the DI control using the `ChannelString` property. Enter the number of the port to be used, or

enter a list of ports separated by commas. Make sure the ports you specify support the desired operation. Not all ports on a device may necessarily support buffered digital input. For example of the three ports on the DIO-24 devices, only ports 0 and 1 can be used with the DI control. Consult your hardware manual for more information.

You can run the DI control in either continuous or single buffer mode. In continuous mode, data is acquired until the operation is explicitly stopped, while the single buffered operation acquires a preset number of points and stops. Use the `Continuous` property to select between these two different modes:

```
CWDI1.Continuous = True
```

The `NPatterns` property specifies the size of the acquisition buffer, which also equals the number of patterns acquired per port in a single buffer acquisition. A *pattern* is a value acquired from a digital port that is a numeric representation of the state of the digital lines in the port. `NPatterns` is specified in number of patterns per port in the channel list. For example, if `NPatterns` is set to `1000` and there are two ports in `ChannelString`, the buffer contains 2000 pattern values. When the control is ready to return data to the application, it fires the `AcquiredData` event and returns the patterns in an array.

While the acquisition is running you have the option of receiving progress events from the DI control. Set the frequency of the `Progress` event in the `ProgressInterval` property in number of patterns acquired; `ProgressInterval` must be less than or equal to `NPatterns` and a zero value disables the `Progress` event generation. You can select to have the acquired patterns returned with the event using the `ProgressReturnData` boolean property:

```
CWDI1.NPatterns = 1000
CWDI1.ProgressInterval = 100
CWDI1.ProgressReturnData = False
```

UpdateClock Object

Use the `UpdateClock` object to set the rate used to acquired the digital pattern into the buffer. Depending on the data acquisition device you are using, you may need to supply a clock signal to the device; or you may be able to select an internal rate by frequency or period. Check the hardware manual to see what clock sources are available on your device.

The main property of update clock is `ClockSourceType`, which sets the source for the clock signal. This may be the I/O connector of the device, the RTSI bus, or an internal clock. Depending on this value, different other properties are used to further specify the clock. If you are using an internal clock you can use the `InternalClockMode` and `Frequency` or `Period` properties to set the acquisition rate:

```
CWDI1.UpdateClock.ClockSourceType =
  cwdioCSInternalClock
CWDI1.UpdateClock.InternalClockMode = cwdioFrequency
CWDI1.UpdateClock.Frequency = 1000
```

You do not need to set any other properties if you select to use the I/O connector or RTSI bus.

Methods and Events

You normally operate the DI control using its `Configure`, `Start`, and `Reset` methods. None of these methods require any parameters. After you set the properties of the control, call the `Configure` method to program the driver and hardware with the current property values. Next, call the `Start` method to initiate the acquisition. Use the `Reset` method during a continuous acquisition to stop the acquisition and to release any resource assigned in the `Configure` call. You may perform another acquisition without calling `Configure` again if the control was not reset since the last acquisition. However, if any of the property values were changed, you need to call `Configure` to implement those changes:

```
Private Sub DigStart_Click()
   CWDI1.Configure
   CWDI1.Start
End Sub
Private Sub DigStop_Click()
   CWDI1.Reset
End Sub
```

☞ **Note:** ***If your digital acquisition requires an external signal to use as a clock, you must physically connect this signal to the proper line of the I/O connector of your data acquisition card. Consult your hardware manual to determine where to connect this signal.***

The acquisition can generate two types of events. The `AcquiredData` event is generated at the completion of a single buffer acquisition or at `NPatterns/2` intervals of a continuous acquisition. The `AcquiredData` event is the main mechanism for returning the acquired data to the application. Data from a one port acquisition is returned in a one dimensional array; data from a multiport acquisition is returned in a two dimensional array.

```
Private Sub CWDI1_AcquiredData(Waveform As Variant)
    CWGraph1.PlotY Waveform, 0, 1, True
End Sub
```

The `Progress` event is generated if the `ProgressInterval` property has a value other than zero, and it returns data if you set the `ProgressReturnData` property to `True`. Use the `Progress` event to retrieve data during an acquisition or to show the progress of an acquisition on the user interface:

```
Private Sub CWDI1_Progress(ByVal TotalPatterns As Long,
  Waveform As Variant)
    ' show percent complete
    CWSlide1.Value = TotalPatterns / CWDI1.NPatterns * 100
    Text1.Text = TotalPatterns
End Sub
```

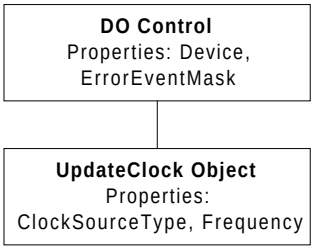
For programming environments that do not support event handler routines, you can call the `AcquireData` method to perform a synchronous acquisition. You can only do this with a single buffered operation:

```
Private Sub SynchAcq_Click()
    Dim waveform As Variant
    CWDI1.Configure
    CWDI1.AcquireData waveform, 5
    CWGraph1.PlotY waveform, 0, 1, True
End Sub
```

See the online reference manual, for more information on the DI control and its object, properties, methods and events.

DO Control—Buffered Waveform Digital Output

Use the DO control to perform buffered waveform digital output operations. The DO control is similar to the DI control. DO allows you to generate a digital stream from the digital outputs at a rate specified by an external signal or internal frequency. Typical applications using the DO control include generating digital test signals for testing of electronic devices, systems and networks. You can also use the DO control to transfer binary data from a computer to another computer or other device. By building your application after the properties of the DO control are set, your program can perform the digital waveform generation using a couple of simple method calls to the DO control.



The DO control consists of a top level object and `UpdateClock` object. You set most of the properties that determine the actions of the DO control on the control itself, while the update clock object determines the source for the rate of generation.

The capabilities of the DO object depend on the data acquisition device you are using. Many DAQ devices do not support buffered digital generation and the DO control will not work with these devices. Other devices support only single buffered generation. you will not be able to use the continuous mode in the DO control. Most of these devices also require an external signal to set the update rate and you can not simply specify an internal frequency. Devices in the DIO-32 series support both continuous acquisitions and internal frequency sources, and can take advantage of the full range of the DO control features. Check the hardware manual of your data acquisition device for detailed information on its capabilities.

DO Object

After you set the default DAQ properties of the DO object (such as `Device`, the most important property to configure is `ChannelString`. You specify what digital ports are used by the DO control using the

ChannelString property. Enter the number of the port to be used, or enter a list of ports separated by commas. Make sure the ports you specify support the desired operation. Not all ports on a device may support buffered digital input. For example, of the three ports on the DIO-24 devices, only ports 0 and 1 can be used with the DO control. Port 2 is used to connect the timing signals required for these operations. Consult your hardware manual for more information.

You can run the DO control in either continuous or single buffer mode. In the continuous mode, data is generated until you explicitly stop the operation; while the single buffer operation generates a preset number of data values stored in the control buffer once and stops. Use the Continuous property to select between these two different modes:

```
CWDO1.Continuous = True
```

The NPatterns property specifies the size of the control buffer used to store the output data. A pattern is a value generated from a digital port numerically representing the state of the digital lines in the port. NPatterns is specified in number of patterns per port in the channel list. For example, if NPatterns is set to 1000 and there are two ports in ChannelString, the buffer contains 2000 pattern values.

While the generation occurs, the control fires the Progress event to indicate the status of the operation. By default the progress event is only generated at the completion of a single buffer operation or at the end of every completed half-buffer in a continuous operation. You can set the frequency of the Progress event explicitly by disabling the AutoSelectProgressInterval property and specifying the ProgressInterval property in number of patterns acquired; ProgressInterval must be less than NPatterns and a zero value disables the progress event generation:

```
CWDO1.NPatterns = 1000
CWDO1.AutoSelectProgressInterval = False
CWDO1.ProgressInterval = 100
```

UpdateClock Object

Use the UpdateClock object to set the update rate of the digital waveform generation. Depending on the data acquisition device you are using, you may need to supply a clock signal to the device or you may be able to select an internal rate by frequency or period. Check the hardware manual to see what clock sources are available on your device.

The main property of the scan clock object is `ClockSourceType`, which sets the source for the clock signal. This may be the I/O connector of the device, the RTSI bus or an internal clock. Depending on this value, other properties are used to further specify the clock. If you are using an internal clock, you can use the `InternalClockMode` and `Frequency`, `Period`, or `Timebase` properties to set the update rate:

```
CWDI1.UpdateClock.ClockSourceType = cwdioCSInternalClock
CWDI1.UpdateClock.InternalClockMode = cwdioFrequency
CWDI1.UpdateClock.Frequency = 1000
```

If you use the I/O connector or RTSI bus, no other properties need to be set.

Methods and Events

You normally operate the DO control using its `Configure`, `Write`, `Start`, and `Reset` methods. Only the `Write` method requires any parameters to be passed. After you set the properties of the control, call the `Configure` method to program the driver and hardware with the current property values. Next you must call the `Write` method to send data to the control buffer for output. Data should be passed as a one dimensional array of pattern values for a single port output, and a two dimensional array of patterns for a multiport output. You then call the `Start` method to initiate the output operation. In a single buffer output, the generation stops after it completes the buffer. In a continuous generation, use the `Reset` method to stop the output and to release any resource assigned in the `Configure` call. You may perform another output without calling `Configure` again if the control was not reset since the last generation. You must call `Configure` again to implement any changes you make to the property values:

```
Private Sub DigStart_Click()
    Dim data(0 to 1, 0 to 99) ‘ 2-port output
    For i = 0 To 99
        data(0, i) = i ‘ port 1 data
        data(1, i) = Int(Rnd * 256) ‘ port 2 data
    Next i
    CWDO1.Configure
    CWDO1.Write data
    CWDO1.Start
```

```
End Sub
Private Sub DigStop_Click()
    CWDO1.Reset
End Sub
```

☞ **Note:** ***If your digital waveform generation requires an external signal to use as a clock, you must physically connect this signal to the proper line of the I/O connector of your data acquisition card. Consult your hardware manual to determine where to connect this signal.***

You also use the `Write` method during a continuous output to send new data to the buffer so that it can be output from the digital lines. This new data is usually written in half buffer sizes to the control. Use the `Progress` event described next to determine when to write new data to the DO control. In continuous mode, you can set the `AllowRegeneration` property to `False` to prevent the DO control from generating the same data in the buffer twice.

During the digital output operation, the DO control generates a `Progress` event to notify you of the completion of an output or to allow you to monitor its progress. In a continuous generation, the `Progress` event is normally set to fire after half the buffer is generated to prompt you to send new data to the control. Data is written to the control using the `Write` method as shown before although only half the buffer is normally updated at a time.

```
Private Sub CWDO1_Progress(ByVal TotalPatterns As Long)
    Dim data(0 to 1, 0 to 49)
    For i = 0 To 49
        data(0, i) = i
        data(1, i) = Int(Rnd * 256)
    Next i
    CWDO1.Write data
End Sub
```

See the online reference manual, for more information on the DO control and its object, properties, methods and events.

Tutorial: Using the DIO control

This tutorial develops a simple program that uses the DIO control to perform single point digital inputs and outputs. Your DAQ device must have one or more digital ports to use this example. You also need a way to apply signals to the digital lines if you want to perform an input, or a way to display the state of any output lines. If you have two digital ports you can connect all the lines from one to the other so that the input can read the state of the output lines.

This tutorial uses Visual Basic syntax, but the discussion is in general terms so that 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 on implementing any particular step. You can also consult the Tutorial examples installed with ComponentWorks for a completed version of this example in several different programming environments.

Designing the Form

1. Open a new project and form. If you are working in Visual C++, select a dialog based application and name your project `DIO`.
2. Load the ComponentWorks user interface controls (specifically, the numeric edit box **CWNumEdit**) and the data acquisition controls (specifically, the DIO control) into your programming environment. Consult the chapter discussing your environment if you are not familiar with this operation.

3. Place two buttons on the form. Change their captions and names to the ones listed in the following table:

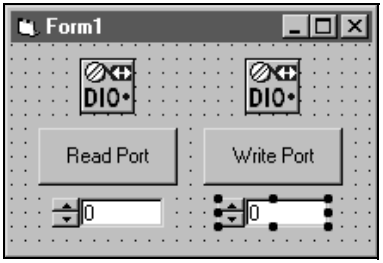
Name	Caption
Input	Read Port
Output	Write Port

4. Place two ComponentWorks numeric edit box controls on the form and change their names to `InputNum` and `OutputNum`.

5. Place two DIO controls on the form. Use the first one (CWDIO1) for your input operations, and the second (CWDIO2) for any output

operation. You will configure the DIO controls in the next section of this tutorial.

Your form should now look similar to the one shown below:



Developing the Code

Next, configure the DIO controls and add the necessary code to write an update to the output lines or read the state of the input lines. If you only use either the input or output control, follow the directions for the appropriate task. Most digital ports can be configured for either input or output, although there are ports on some boards which can only be configured as one or the other. Check your hardware manual for information on such limitations.

1. Configure one DIO control for input and one for output.

   Input: In the custom property page of the first DIO control `CWDIO1`, select the appropriate DAQ device. Then select (highlight) one digital port in the list and set its direction (Port assignment) to `Input`. Set the port assignment for all other ports to `Unused`.

   Output: In the custom property page of the second DIO control `CWDIO2`, select the appropriate DAQ device. Then select (highlight) one digital port in the list and set its direction (Port assignment) to `Output`. Set the port assignment for all other ports to `Unused`.

   Depending on your device, you may be able to select the same port for both controls. You can then write to the port using one control and read back a value with the other control without making any external connections.

2. Use the **Read Input** button to read the state of the digital input port and display the pattern value in the numeric edit control. You can read the state of all the ports at once or you can select to read a specific port using the DIO control. In this example, read only the

selected port. This makes the code more independent of the hardware you are using.

Create an event handler for the `Click` event of the **Input** button and add the following code. In the `Item` method, use the number of the digital port you configured for Input. In this example, port 1 is used. The first port on a board is normally port 0.

```
Private Sub Input_Click()
    Dim data As Variant
    CWDIO1.Ports.Item(1).SingleRead data
    InputNum.Value = data
End Sub
```

☞ **Note:** ***If you read all the ports at once, the data is retuned as an array and you need to access individual array elements for the information. The following lines of code show how to do this. Do not add them to your example.***

```
Dim data As Variant
CWDIO1.SingleRead data
InputNum.Value = data(1)
```

3. Use the **Write Output** button to set the state of the digital output port and display the pattern value in the numeric edit control. You can write the state of all the ports at once or you can select to write to a specific port using the DIO control. In this example, write only to the selected port. This makes the code more independent of the hardware you are using.

   Create an event handler for the `Click` event of the **Output** button and add the following code. In the `Item` method, use the number of the digital port you configured for Output. In this example, port 0 is used, which is the first port on most boards.

```
    Private Sub Output_Click()
    Dim data As Variant
    data = OutputNum.Value
    CWDIO2.Ports.Item(0).SingleWrite data
End Sub
```

☞ **Note:** ***If you want to write to all the ports at once, you need to pass in an array of data with a value for each port on the device you are using. The following lines of code show how to do this. Do not add them to your example:***

```
Dim data As Variant
' for a device with one port
data = OutputNum.Value
' for a device with three ports
data = Array(OutputNum.Value, 0, 0)
CWDIO2.Ports(0).SingleWrite data
```

4. Save your project and form as `DIO`.

Testing Your Program

Before you can run your program, you should connect a signal source to your input port and a display or sensor to the output port. As an option, you can connect the output lines to the input lines and use the input to measure the output.

1. Run the program. Write a value to the output port by setting a value in the corresponding numeric edit box and pressing the **Write Output** button. The value in the numeric edit box represents the pattern written to the port. The pattern is a numeric (integer) representation of the bits in the digital port. A value of zero means all lines are written low. A value of 22 (binary 00010110) means lines 1, 2 and 4 are written high. Most ports have 8 digital lines and a value of 255 corresponds to all lines high.
2. Read the value of the input port by pressing the **Read Input** button. The pattern value is read from the port and displayed in the corresponding numeric edit box. If you have connected all lines of the output port to the input port, you should read the same value you wrote out. If you read input lines that are not connected to anything (floating), their values will not necessarily be low and therefore an open port may read any value and not always 0.
3. Quit the program.

Instead of using the `SingleRead` and `SingleWrite` methods, you can also use the `Value` property and `Update` method of the DIO control to implement the functionality of this tutorial. The `Update` method can be called on the DIO control as a whole or on the individual port. The following are examples of the event handlers using this alternative:

```
Private Sub Input_Click()
    Dim data As Variant
    CWDIO1.Update
    data = CWDIO1.Ports.Item(1).Value
    InputNum.Value = data
End Sub
Private Sub Output_Click()
    Dim data As Variant
    data = OutputNum.Value
    CWDIO2.Ports.Item(0).Value data
    CWDIO2.Ports.Item(0).Update
End Sub
```

DAQTools—Data Acquisition Utility Functions

- The ComponentWorks DAQ controls include a DAQTools control that contains a set of utility function for data acquisition. These utility functions are all methods of the DAQTools control, while the control itself has no properties or events. They can be used with many different data acquisition devices to implement specific functionality which is not part of any of the other DAQ controls. The function groups in the DAQ Tools control include:
- The `GetErrorText` function converts a ComponentWorks error number into a descriptive string. Use this in error handling to take the return code from a DAQ method call and convert it to an error message.
- *Configure* functions configure specific devices or parts of devices such as the hardware analog trigger circuit on specific E-Series devices.
- *Conversion* functions convert measurement units into physical units for certain transducers such as thermocouples, thermistors, RTDs, and strain gauges.

- *Get* and *Set* functions read and set different properties of data acquisition and SCXI (signal conditioning) devices.
- *Reset* functions reset DAQ and SCXI devices.
- *ICtr* functions perform operations using the interval counter (ICounter) on some data acquisition devices. Devices that include this counter are the 500, 700, 1200, LPM-16 and Lab-PC series devices. The functions are `ResetICtr`, `StartICtr`, and `ReadICtr`.
- *FOUT* functions generate a simple continuous pulse train from the FOUT pin of different DAQ devices. The functions are `StartFOUT` and `ResetFOUT`.
- *Calibration* functions perform software calibration on different data acquisition devices.

☞ **Note:** ***In most cases you do not need to calibrate your device. All devices are calibrated before being shipped and they do not need to be calibrated before using them for the first time. It is very important that you read the hardware documentation regarding calibration and the description of the calibration functions before attempting to perform any calibration.***

Using DAQ Tools functions

You must drop the DAQTools control on your form to use the DAQTools functions. Once you do this you can use any of its functions in your program. To call a function use the regular convention of calling any method of a control, i.e. prepend the name of the control to the function name. The following example converts an error code number to a text description and displays it in a message box:

```
MsgBox "DAQ Error: " +
  CWDAQTools1.GetErrorText(ErrorCode)
```

To configure the hardware analog trigger circuit you can use the `ConfigureATCOut` function:

```
' Prototype: ConfigureATCOut(Device:=, Enable:=,
  TriggerMode:=, Level:=, Hysteresis:=, strSource:=,
  ActualLevel:=, ActualHysteresis:=)

Dim Level as Variant, Hysteresis as Variant

CWDAQTools1.ConfigureATCOut 1, True, 1, 0#, 0.1, "0",
  Level, Hysteresis
```

See the online reference manual for a complete description of all the functions in the DAQTools control.

Counter/Timer Hardware

The hardware component on the DAQ device used by the Counter and Pulse controls is called a counter/timer. You can use this component to count or measure incoming digital pulses as well generate digital pulses and pulse trains on its output.

Each counter/timer has two inputs labeled *Source* and *Gate* and one output labeled *Out*. The Source is also referred to as *Clock*. The basic operation of the counter is to count the number of digital pulses coming in on the source input. The counting operation can be gated by a digital signal applied to the gate input. The output of the counter generates a pulse when the counter reaches its maximum count or zero, depending on whether it is counting up or down. By varying parameters such as the signals applied to the source and gate, as well as the initial count of the counter and the mode used in gating, this simple component can be used for a variety of different applications.

The ComponentWorks Counter and Pulse control simplifies the use of the counter/timer by allowing you to select from a number of standard operations and specifying applicable properties. The properties in the controls are grouped by the I/O point of the counter that they affect, such as source, gate and output.

Counter Control—Counting and Measurement Operations

Use the Counter control to perform counting and other measurement operations using the counter/timer components on a data acquisition device. Typical operations include counting a number of events, measuring the period of an unknown pulse, or measuring the frequency of a signal. After the properties of the control are set, the application can perform the operation using a simple method call to the counter control. The operation of the counter control is measurement-oriented,

and the behavior of the control depends on the type of measurement selected.

Counter Control Properties:
e.g. Device, Counter, Measurement Type

The counter control consists of only one object. The counter object contains all the properties and methods necessary to use the control

Counter Object

Besides the default properties of each DAQ control, the counter object contains all the properties necessary to a configure counting or measurement operation. There are a three properties that affect the counter control as a whole, while the remaining properties apply specifically to the source, gate or output of the counter.

The two main properties of the control are `Counter` and `MeasurementType`. The `Counter` property specifies what counter/timer on the DAQ device will be used by the control. The available counter numbers depends on the type of DAQ device you are using. The `MeasurementType` property selects the type of operation the control will perform, and affects the interpretation of the remaining properties. Another property that affects the general operation is `InitialCount`, which sets the value of the counter at the start of an operation.

The following table describes the different measurement types:

Measurement Type	Description	Units	One Point	Buffered
Event Count	Counts the number of pulses on the source input.	Count	X	X
Time	Measures time by counting a known clock input.	Sec	X	X
Frequency	Measure frequency on source input by counting number of pulse in a known period of time.	Hertz	X	

Measurement Type	Description	Units	One Point	Buffered
Pulse Width, High	AM9513: Measure length of high pulse until next falling edge (partial measurement possible if pulse is initially high). DAQ-STC: Measure first pulse after next falling edge.	Sec	X	X
Pulse Width, Low	AM9513: Measure length of low pulse until next rising edge (partial measurement possible if pulse is initially low). DAQ-STC: Measure first pulse after next rising edge.	Sec	X	X
Pulse Width, High Single-Shot	AM9513: Measure length of high pulse until next falling edge (partial measurement possible if pulse is initially high). DAQ-STC: Measure first complete pulse, generates error if pulse is in high state at start.	Sec	X	
Pulse Width, Low Single-Shot	AM9513: Measure length of low pulse until next rising edge (partial measurement possible if pulse is initially low). DAQ-STC: Measure first commplete pulse, generates error if pulse is in low state at start.	Sec	X	
Period, Rising Edge	Measure period between two rising edges on gate input by counting a known clock	Sec	X	X
Period, Falling Edge	Measure period between two falling edges on gate input by counting a known clock	Sec	X	X
Semi-Periods	Measure of length of consecutive high and low pulses	Sec		X

Frequency and Single Shot Pulse Width measurements can only be performed as one point (unbuffered) operations, but the Semi-Period measurement must always be buffered. See the section on Buffered Measurements for more information.

The remaining properties of the counter object directly affect one of the inputs or outputs of the control. The properties affecting the source input include `TimebaseSource`, `TimebaseSignal`, `SourceEdge` and `CountDirection`. Depending on the TimebaseSource property, TimebaseSignal selects either a specific input pin of your board or the internal frequency used for the source of the counter. `SourceEdge` selects whether rising or falling edges on the source are counted and `CountDirection` determines whether the counter counts up or down.

The gate properties are `GateMode, GateSource`, `GateSignal`, and `GateWidth`. `GateMode` selects the type of gating used in an operation such as *No Gating*, *High Level Gating*, and so on. `GateSource` and `GateSignal` select the source of the gate signal. `GateWidth` is only

used in Frequency measurement to set the sample width of the measurement.

The output properties are `OutputMode` and `Polarity`. `OutputMode` determines whether the counter output pulses or toggles when the counter reaches its limit, and `Polarity` determines if the output is high (active high) or low (active low) polarity.

To become familiar with the different types of measurements and properties, browse through the custom property pages of the counter control and study the different settings of the properties. Some properties may be disabled depending on the state of other properties. Consult the online reference for complete information on all properties.

Methods and Events

The counter control has a set of simple methods to control the operation of the counter. Call the `Configure` method to configure the data acquisition driver and hardware with the current properties before starting any operation. Call the `Start` method to start the measurement. If you want, you can call the `Stop` method to stop the counter and the `Reset` method to unconfigure the counter and free up its resources for other operations. After stopping a counter, you can restart it using just the `Start` method if you have not called the `Reset` method. After resetting the counter you need to call `Configure` again before restarting.

```
Private Sub CounterStart_Click()
    CWCounter1.Configure
    CWCounter1.Start
End Sub
Private Sub CounterStop_Click()
    CWCounter1.Stop
    CWCounter1.Reset
End Sub
```

Depending on the measurement, the control may fire an `AcquiredData` event to indicate the completion of a measurement and return the data. The data is returned in the `Measurement` parameter that is passed to the `AcquiredData` event, and scaled to the units indicated in the previous table. Measurements that fire an `AcquireData` event are Frequency, Pulse Width, Period and Semi-Period.

```
Private Sub CWCounter1_AcquiredData(Measurement As
  Variant, ByVal Overflow As Boolean)
    txtPulseWidth.Text = Measurement
End
```

Event counting and time measurement do not fire an event to return data. Therefore, you use the `ReadCounter` and `ReadMeasurement` methods to read the value of your measurement. `ReadCounter` returns the actual count value of the counter and is usually used in event counting operations. `ReadMeasurement` returns the value scaled to the appropriate units. Both methods require one parameter to return the measurement and another to indicate an overflow condition in the counter.

```
Private Sub Timer1_Timer()
    Dim lVal As Long
    Dim vVal As Variant
    Dim bOverflow As Boolean
    CWCounter1.ReadCounter lVal, bOverflow
    txtCount = lVal
    CWCounter1.ReadMeasurement vVal, bOverflow
    txtMeas = vVal
End Sub
```

Buffered Measurements

Normally all measurements acquire and return a single value of the selected type. On specific data acquisition devices, such as the E-Series devices, you can select to perform a buffered measurement for most of the measurement types. A buffered measurement acquires multiple values and stores the individual values in a buffer for later analysis and processing. After a measurement is started, measurements are stored in the buffer asynchronously. Once all the data points have been acquired, the `AcquiredData` event is fired and the buffer is returned to the application. Measurements are stored in the buffer at the completion of a period or pulse for period and pulse measurements, or at a conversion signal on the gate input for event counting and time measurements.

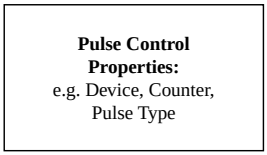
The counter object has two properties to enable buffered measurements. `UseBuffering` is a boolean property that enables the buffered mode. `NMeasurements` specifies the number of measurements to be made.

While a buffered measurement is running, no data can be read from the counter or the buffer and all data is returned when the measurement is complete. The same methods used for a one point operation are also used to control buffered measurements. When data is returned in the `Measurement` parameter of the `AcquiredData` event, it is formatted as an array.

```
Private Sub BufferedStart_Click()
    CWCounter1.MeasurementType = cwctrHiPulseWidth
    CWCounter1.UseBuffering = True
    CWCounter1.NMeasurements = 10
    CWCounter1.Configure
    CWCounter1.Start
End Sub
Private Sub CWCounter1_AcquiredData(Measurement As
  Variant, ByVal Overflow As Boolean)
    CWGraph1.PlotY Measurement, 0, 1, True
End Sub
```

Pulse Control—Digital Pulse and Pulsetrain Generation

Use the Pulse control to generate individual pulses as well as pulse trains with the counter/timer component on a data acquisition device. Typical operations using the pulse control include generating a digital test signal or driving a stepper motor. After the properties of the control are set, the application can perform operations using simple method calls to the pulse control. The operation of the pulse control is task-oriented and the behavior of the control will depend on the type of task selected in the properties.



The pulse control consists of only one object. The pulse object contains all the properties and methods necessary to use the control

Pulse Object

Besides the default properties of each DAQ control, the pulse object contains all the properties necessary to configure the pulse operations. Two properties affect the pulse control as a whole, while the remaining properties apply specifically to the pulse specifications, and the gate or output of the counter/timer.

The two main properties of the control are `Counter` and `PulseType`. The `Counter` property specifies what counter/timer on the DAQ device will be used by the control. The available counter numbers depend on the type of DAQ device you are using. The `PulseType` property selects the type of operation the control performs. This selection also affects the interpretation of the remaining properties.

The following table describes the different pulse type operations:

Pulse Type	Description
Single Pulse	Generate one pulse according to the specifications.
Continuous Pulse Chain	Generate a continuous pulse train according to the specifications.
Finite Pulse Chain	Generate a finite set of pulses. This requires two counters.
FSK Pulse	Frequency Shift Keying, generate one of two different pulse trains dependent on a digital input, selector is applied to gate of counter.
Incremental Delay Pulse (ETS)	Equivalent time sampling pulse, generate series of individual pulses with increasing offset from digital trigger, trigger is applied to gate of counter.
Retriggerable Pulse, Rising Edge	Retriggered single pulse, trigger is applied to gate, triggered off rising edge.

Pulse Type	Description
Retriggerable Pulse, Falling Edge	Retriggered single pulse, trigger is applied to gate, triggered off falling edge.

The remaining properties of the counter object directly affect the pulse generation or the gate of the counter. The properties affecting the pulse generation include `ClockMode`, `CountDirection`, `DutyCycle`, `Frequency`, `Period`, `Period2`, `Phase1`, `Phase2`, `Frequency2`, `Count`, `Phase1Inc`, `PulseDelay`, `PulseWidth`, `SourceEdge`, `TimebaseSource`, and `TimebaseSignal`. The parameters which are actually used for the pulse generation depend on the settings of the `PulseType` and `ClockMode` properties. The `PulseType` settings select the general operation of the pulse control, such as single pulse or continuous pulse generation, while the `ClockMode` settings determine how the output is characterized. Settings for ClockMode include `Frequency` and `Period`, which means the output is characterized by its frequency or by the period of the pulse. Consult the property page of the pulse control for more information on which property is used in a specific operation.

The gate properties are of the pulse control are `GateMode`, `GateSource`, `GateSignal`. `GateMode` selects the type of gating used in an operation such as *No Gating*, *High Level Gating*, and so on. `GateSource` and `GateSignal` select the source of the gate signal.

To become familiar with the different types of pulse generation and their corresponding properties, browse through the custom property pages of the counter control and study the different settings of the properties. Some properties may be disabled depending on the state of other properties. Consult the online reference for complete information on all properties.

Methods

The pulse control has a set of simple methods to control the operation of the pulse generation. Use the `Configure` method to configure the data acquisition driver and hardware with the current properties before starting any operation. Use the `Start` method to start the generation. If you want, you can call the `Stop` method to stop the counter and the `Reset` method to unconfigure the counter and free up its resources for other operations. After stopping a pulse generation, you can restart it using just the `Start` method if you have not called the `Reset` method. After resetting the counter you need to call `Configure` again before restarting.

```
Private Sub PulseStart_Click()
    CWCounter1.Configure
    CWCounter1.Start
End Sub
Private Sub PulseStop_Click()
    CWCounter1.Stop
    CWCounter1.Reset
End Sub
```

While a pulse generation operation is running you can also change some of its parameters without stopping the operation. For example, you can change the frequency of continuous pulse generation in mid-stream. To do this, update the relevant property of the pulse control. Then call the Reconfigure method. `Reconfigure` can only be called while an operation is running, and updates all possible properties that have changed since the last call to `Configure` or `Reconfigure`. You can use this method easily with the PointerValueChanged event of a slide control to set the pulse frequency to the value of the slide.

```
Private Sub CWSlide1_PointerValueChanged(ByVal Pointer
  As Long, Value As Variant)
    CWPulse1.Frequency = Value
    If OutputRunning Then CWPulse1.Reconfigure
End Sub
```

FSK and ETS Pulse Generation

The pulse control supports two specialized pulse generation modes on the E series data acquisition devices that you can select in the `PulseType` property. These modes can be used in conjunction with analog input operations to perform special acquisitions. You may also use them on their own for applicable situations.

Use FSK (frequency shift keying) to generate a continuous pulse with one of two different frequencies depending on a separate digital signal. Apply the digital signal that selects between the two different frequencies to the gate input of the counter used by the pulse control. Using the `Frequency` and `Frequency2`, `Period` and `Period2`, or `Phase1` and `Phase2` properties, you can select the two different frequencies to generate. This type of pulse generation can be combined with an analog input to build a Rate-Change-on-the-Fly acquisition.

In this case, the output of the counter is used as the scan clock of the acquisition, so you can change the acquisition rate dynamically using a digital signal. If you have a hardware analog trigger on your device, you can further enhance this application by using the analog trigger to convert an analog signal, such as the one being acquired, into a digital signal, which you then use to control the pulse generation. This means you can use two different acquisition rates on your analog signal and the acquisition will automatically change rates above and below the specified analog voltage value.

ETS (equivalent time sampling) is an acquisition technique that combines an analog input operation, hardware analog trigger and pulse generation on a repetitive signal to achieve an effective acquisition rate which is significantly higher than the actual acquisition rate of the device.

In this operation the analog trigger is used to generate a repeating digital trigger signal off the repetitive analog signal. This digital signal is routed to the gate of a counter that generates another digital pulse with an increasing amount of delay in response to the trigger. This pulse is used as the scan clock of the acquisition. By triggering off the same point of the analog signal with an exact and varying delay, you can sample the entire signal waveform using many cycles. Because the change in the delay is much smaller than the minimum acquisition period of the device, the effective acquisition rate is significantly increased.

In the pulse control, specify the input of the gate to be `AITrigger` and set the change in delay using the `ETSIncrement` property. The ETS increment is always specified in number of cycles of the clock source, and can range between 0 and 255.

Tutorial: Using the Counter and Pulse controls

This tutorial develops a simple program that uses the Counter and Pulse controls. The example generates a continuous pulse of varying frequency using the pulse control, and counts the number of pulses generated using the counter control. To use this example, your DAQ device must have two available counters. You may also use just one counter in this example with either the counter or pulse control. In this case you will need an external mechanism to display the output from the pulse control or apply an input to the counter control.

This tutorial uses syntax from the Visual Basic environment, but the discussion is in general terms so that 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 on implementing any particular step. You can also consult the Tutorial examples installed with ComponentWorks for a completed version of this example in several different programming environments.

Designing the Form

1. Open a new project and form. If you are working in Visual C++, select a dialog based application and name your project `Counters`.
2. Load the ComponentWorks user interface controls (specifically the slide) and the data acquisition controls (specifically the Counter and Pulse controls) into your programming environment. Consult the chapter discussing your environment if you are not familiar with this operation.

3. Place five buttons on the form. Change their captions and names to the ones listed in the following table.

Name	Caption
ConfigurePulse	Configure Pulse
StartPulse	Start Pulse
StopPulse	Stop Pulse
StartCounter	Start Counter
ReadCounter	Read Counter

4. Place a ComponentWorks slide control on the form and change its name to `Frequency`. In the custom property sheet under the *Numeric* tab, change its *Min* and *Max* properties to 1 and 100 and set the Log scale option.

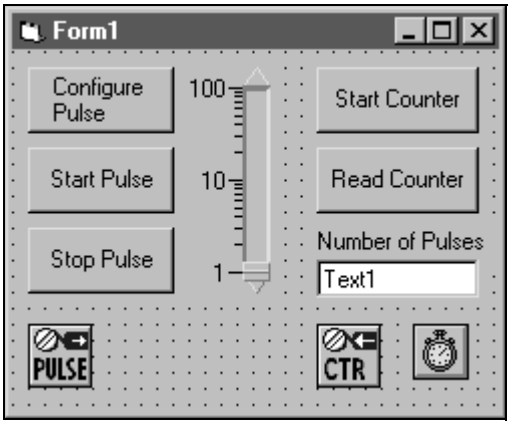
5. Place a text box on the form.

6. Place a DAQ Pulse control on the form. In the custom property sheet of the Pulse control, select the device and counter you want to use. If you have an E-Series device use counter 0; with an older MIO device use counter 5, otherwise use an available counter. You will set the remaining properties in the following section.

7. Place a DAQ Counter control on the form. In the property sheet, select the device and counter you want to use. If you have an E-Series or older MIO device use counter 1, otherwise use an available counter.

8. If your environment has a timer control, place one on the form. In the property sheet, set its `Enabled` property to `False` and its `Interval` property to 100 (ms).

Your form should now look similar to the one shown below.



Developing the Code

Next configure the DAQ controls and add the necessary code to generate the continuous pulse train and count the pulses. If you will only use one counter, follow the directions for either the counter or pulse control. The next section will describe how you should connect your signals in either case.

1. Configure the Pulse control to generate a continuous pulse train. In the property page of the pulse control under the General tab set the Pulse Mode to `Continuous pulse chain`. Under the Clock tab set the Clock Mode to `Use frequency settings`, the Frequency to 1.0 and the duty cycle to 0.50 (50%). Under the Gate tab set the Gate Mode to `Ungated`.
2. There are three buttons on the form which control the operation of the pulse control. Generate event handlers for the `Click` event of each button and add the following code. You also need to declare a boolean variable, `Running`, which is global to the module containing the code for the form. Depending on your programming language this declaration may vary. In Visual C++ you will also need to create a member variable for the pulse control.

```
Dim Running as Boolean
Private Sub ConfigurePulse_Click()
   CWPulse1.Configure
End Sub
Private Sub StartPulse_Click()
```

```
    CWPulse1.Start
    Running = True
End Sub
Private Sub StopPulse_Click()
    CWPulse1.Stop
    Running = False
End Sub
```

Using the Stop and Start buttons you can halt the pulse generation and restart it without having to reconfigure the pulse control.

3. The slide control is used to change the frequency of the pulse train. If the pulse generation is running the output will be updated immediately. Create an event handler for the `PointerValueChanged` event of the slide. Add the following code.

```
Private Sub Frequency_PointerValueChanged(ByVal
  Pointer As Long, Value As Variant)
    CWPulse1.Frequency = Value
    If Running Then
        CWPulse1.Reconfigure
    Else
        CWPulse1.Configure
    End If
End Sub
```

If the pulse generation is running the `Reconfigure` method is used to update the output immediately, otherwise the `Configure` method reprograms the hardware for the next start.

4. Configure the Counter control to count events, which are the pulses generated by the pulse control. In the property page of the counter control under the General tab, set the Measurement Type to `Event count`. Under the Clock tab set the Timebase source to `Counter's source`. Under the Gate tab set the Gate Mode to `Ungated`.

5. There are two buttons on the form which control the operation of the pulse control. Generate event handlers for the `Click` event of each button and add the following code. In Visual C++ you will also need to create a member variable for the counter control. Add the following code to the two event handlers.

```
Private Sub StartCounter_Click()
    CWCounter1.Configure
    CWCounter1.Start
End Sub
Private Sub ReadCounter_Click()
    Dim CountValue As Long
    Dim Overflow As Boolean
    CWCounter1.ReadCounter CountValue, Overflow
    Text1.Text = CountValue
End Sub
```

   After the counter is started, you can use the Read button to interactively read the value of the counter and display it in the textbox.

6. If you have a timer control on your fom you can program it to automatically read the value of the counter once it has been started. If you are working in Visual C++ create a member variable for the timer control.

   To activate the timer add the following line to the end of the code in the `StartCounter_Click` subroutine.

```
Timer1.Enabled = True
```

   Create an event handler for the `Timer` event of the Timer control and add a call to the `ReadCounter_Click()` subroutine.

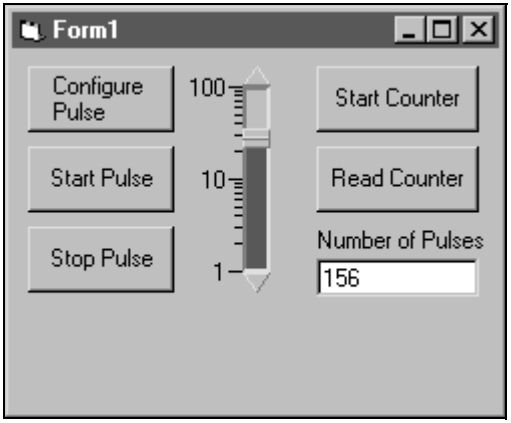
```
Private Sub Timer1_Timer()
    ReadCounter_Click
End Sub
```

7. Save your project and form as `Counters`.

Testing Your Program

1. Before you can run your program you need to connect the output of the counter running the pulse generation to the source input of the counter running events. This is an external connection you need to make. If you are only using one counter of the DAQ device you will need some other external hardware. If you are using the DAQ pulse control you should have a device to measure the output of the counter such as an oscilloscope or external counter. If you are using the DAQ counter control you will need to connect an external pulse source to the source input of the counter.
2. Run the program. Start the pulse generation by clicking on the *Configure Pulse* and *Start Pulse* buttons. Then start the counter. If you have the timer control enabled the textbox should display the value of the event count. Without the timer control, you can use the *Read Counter* button to update the textbox.
3. Change the value of the slide control, which changes the frequency of the pulse generation. Notice that the count value in the textbox changes faster or slower accordingly. You can also stop and restart the pulse generation which affects the count value.

Your completed running program should look similar to the one shown below.



4. Quit the progam (click on the ‘X’ button in the title bar of the window).

Using the Analysis Controls and Functions

Chapter 7

This chapter shows you how to use the ComponentWorks analysis controls and functions. You can use the analysis functions alone or with other controls to perform data analysis, manipulation and simulation. Functions are included for operations such as matrix and array calculations, frequency analysis, statistical analysis and signal generation. The analysis functions are stored as methods in a number of different analysis controls and are organized by functionality.

The individual controls and some of the functions are explained in this chapter. Additional information as well as a detailed description of each individual analysis function is found in the online help file. This chapter includes a tutorial that gives step-by-step instructions on using the analysis controls in a simple program. While the code listed in the tutorial uses Visual Basic syntax, the steps can be applied to any programming environment. Consult the appropriate chapters in this manual for information on using the ComponentWorks controls in other environments.

What are the Analysis Controls?

The analysis controls file `CWAnalysis.OCX` contains five separate ActiveX controls which include the different analysis functions. The functions are grouped into controls according to their uses:

- CWArray—array manipulation functions
- CWComplex—complex scalars and array manipulation functions
- CWMatrix—vector and matrix algebra functions
- CWStat—statistical functions
- CWDSP—digital signal processing and signal generation functions

Analysis Library Versions

ComponentWorks is distributed with one of three different versions of the analysis library. Each version contains a different set of functions in the library. The functions available to you depend on the package of ComponentWorks you purchased:

- Base Analysis Library, *ComponentWorks Base Package*—contains simple matrix algebra, array and complex number functions, and simple statistics functions.
- Digital Signal Processing (DSP) Analysis Library, *ComponentWorks Standard Development System*—contains the functions of the base analysis library plus DSP functions (time and frequency domain analysis, filters, windows), curve fitting functions, advanced array and complex number functions and measurement functions.
- Advanced Analysis Library (AAL), *ComponentWorks Full Development System*—contains the functions of the DSP analysis library plus advanced statistics and matrix algebra functions.

Although specific analysis functions or controls may not be part of your analysis library, each is shown in your development environment, such as the Visual Basic object browser or the Visual C++ component gallery. When you attempt to use a function that is not included in your analysis library, an error message appears to notify you that the function is not supported. See the *Testing and Debugging Your Application* section in Chapter 8, *Building Advanced Applications,* for more information.

The following table lists all the analysis functions, grouped by control. The last column specifies the ComponentWorks version which includes the specific function:

- B - Base Package (also in Standard and Full Development System)
- S - Standard Development System (also in Full Development System)
- F - Full Development System

Table 7-1. Analysis Control Function Tree

Control	Function	Function Name	CW Version
CWArray			
	1D 2D Operations		
	1D Maximum & Minimum	MaxMin1D	B
	1D Array Subset	Subset1D	B
	1D Array Reverse	Reverse1D	S
	1D Array Shift	Shift1D	S
	1D Array Sort	Sort1D	B
	1D Array Interleave	Interleave1D	B
	2D Array Transpose	Transpose2D	B
	MultiDimensional Element Operations		
	Array Addition	AddArray	B
	Array Subtraction	SubArray	B
	Array Multiplication	MulArray	B
	Array Division	DivArray	B
	Absolute Value	AbsArray	B
	Negative Value	NegArray	B
	Linear Evaluation	LinEvArray	B
	Polynomial Evaluation	PolEvArray	S
	Scaling	ScaleArray	S
	Quick Scaling	QScaleArray	S
	Array Clipping	ClipArray	S

Table 7-1. Analysis Control Function Tree (Continued)

Control	Function	Function Name	CW Version
	Array Clearing	ClearArray	B
	Array Setting	SetArray	B
	Array Copying	CopyArray	B
	Array Normalizing	NormalizeArray	S
	Variant Conversion	VarToDblArray	B
	MultiDimensional Array Operations		
	Array Size	ArraySize	B
	Sum of Elements	SumArray	S
	Product of Elements	ProArray	S
	Extract complete dimensions(s) from array	IndexArray	B
	Array Subset	SubsetArray	B
	Maximum and Minimum of Array	MaxMinArray	B
	Search Array	SearchArray	B
	Build/Concatenate Array	BuildArray	B
	Reshape Array	ReshapeArray	B
CWComplex			
	Complex Numbers		
	Complex Addition	CxAdd	B
	Complex Subtraction	CxSub	B
	Complex Multiplication	CxMul	B

Table 7-1. Analysis Control Function Tree (Continued)

Control	Function	Function Name	CW Version
	Complex Division	CxDiv	B
	Complex Reciprocal	CxRecip	B
	Complex Square Root	CxSqrt	S
	Complex Logarithm	CxLog	S
	Complex Natural Log	Cxlon	S
	Complex Power	CxPow	S
	Complex Exponential	CxExp	S
	Rectangular to Polar	ToPolar	B
	Polar to Rectangular	ToRect	B
	MultiDimensional Complex Operations		
	Complex Addition	CxAddArray	B
	Complex Subtraction	CxSubArray	B
	Complex Multiplication	CxMulArray	B
	Complex Division	CxDivArray	B
	Complex Linear Evaluation	CxLinEvArray	B
	Rectangular to Polar	ToPolarArray	B
	Polar to Rectangular	ToRectArray	B
CWMatrix			
	Vector & Matrix Algebra		
	Dot Product	DotProduct	B
	Matrix Multiplication	MatrixMul	B

Table 7-1. Analysis Control Function Tree (Continued)

Control	Function	Function Name	CW Version
	Matrix Inversion	InvMatrix	B
	Transpose	Transpose	B
	Determinant	Determinant	B
	Trace	Trace	F
	Solution of Linear Equations	LinEqs	F
	LU Decomposition	LU	F
	Forward Substitution	ForwSub	F
	Backward Substitution	BackSub	F
CWStat			
	Statistics		
	Mean	Mean	B
	Standard Deviation	StdDev	B
	Variance	Variance	F
	RootMean Squared Value	RMS	F
	Moments about the Mean	Moment	F
	Median	Median	F
	Mode	Mode	F
	Histogram	Histogram	B
	Probability Distributions		
	Normal Distribution Function	N_Dist	F
	TDistribution Function	T_Dist	F
	FDistricution Function	F_Dist	F

Table 7-1. Analysis Control Function Tree (Continued)

Control	Function	Function Name	CW Version
	χ^2 Distribution Function	XX_Dist	F
	Normal Distribution Inverse Function	InvN_Dist	F
	TDistribution Inverse Function	InvF_Dist	F
	FDistribution Inverse Function	InvF_Dist	F
	χ^2 Distribution Inverse Function	InvXX_Dist	F
	Analysis of Variance		
	One-way Analysis of Variance	ANOVA1Way	F
	Two-way Analysis of Variance	ANOVA2Way	F
	Three-way Analysis of Variance	ANOVA3Way	F
	Nonparametric Statistics		
	Contingency Table	Contingency_Table	F
	Curve Fitting		
	Linear Fit	LinFit	S
	Exponential Fit	ExpFit	S
	Polynomial Fit	PolyFit	S
	General Least Squares Linear Fit	GenLSFit	S
	Interpolation		
	Polynomial Interpolation	PolyInterp	F
	Rational Interpolation	RatInterp	F
	Spline Interpolation	SpInterp	F
	Spline Interpolant	Spline	F

Table 7-1. Analysis Control Function Tree (Continued)

Control	Function	Function Name	CW Version
CWDSP			
	Signal Generation		
	Impulse	Impulse	S
	Pulse	Pulse	S
	Ramp	Ramp	S
	Triangle	Triangle	S
	Sine Pattern	SinePattern	S
	Uniform Noise	Uniform	S
	White Noise	WhiteNoise	S
	Gaussian Noise	GaussianNoise	S
	Arbitary Wave	ArbitaryWave	S
	Chirp	Chirp	S
	Sawtooth Wave	SawtoothWave	S
	Sinc Waveform	Sinc	S
	Sine Waveform	SineWave	S
	Square Wave	SquareWave	S
	Triangle Wave	TriangleWave	S
	Frequency Domain Signal Processing		
	FFT	FFT	S
	Inverse FFT	InvFFT	S
	Real Valued FFT	ReFFT	S

Table 7-1. Analysis Control Function Tree (Continued)

Control	Function	Function Name	CW Version
	Real Value Inverse FFT	ReInvFFT	S
	Power Spectrum	Spectrum	S
	FHT	FHT	S
	Inverse FHT	InvFHT	S
	Cross Spectrum	CrossSpectrum	S
	Time Domain Signal Processing		
	Convolution	Convolve	S
	Correlation	Correlate	S
	Integration	Integrate	S
	Differentiate	Difference	S
	Pulse Parameters	PulseParam	S
	Decimate	Decimate	S
	Deconvolve	Deconvolve	S
	UnWrap Phase	UnWrapID	S
	IIR Digital Filters		
	Lowpass Butterworth	Bw_LPF	S
	Highpass Butterworth	Bw_HPF	S
	Bandpass Butterworth	Bw_BPF	S
	Bandstop Butterworth	Bw_BSF	S
	Lowpass Chebyshev	Ch_LPF	S
	Highpass Chebyshev	Ch_HPF	S
	Bandpass Chebyshev	Ch_BPF	S

Table 7-1. Analysis Control Function Tree (Continued)

Control	Function	Function Name	CW Version
	Bandstop Chebyshev	Ch_BSF	S
	Lowpass Inverse Chebyshev	InvCh_LPF	S
	Highpass Inverse Chebyshev	InvCh_HPF	S
	Bandpass Inverse Chebyshev	InvCh_BPF	S
	Bandstop Inverse Chebyshev	InvCh_BSF	S
	Lowpass Elliptic	Elp_LPF	S
	Highpass Elliptic	Elp_HPF	S
	Bandpass Elliptic	Elp_BPF	S
	Bandstop Elliptic	Elp_BSF	S
	IIR Filtering	IIRFiltering	S
	FIR Digital Filters		
	Lowpass Window	Wind_LPCoef	S
	Highpass Window	Wind_HPCoef	S
	Bandpass Window	Wind_BPCoef	S
	Bandstop Window	Wind_BSCoef	S
	Lowpass Kaiser Window	Ksr_LPCoef	S
	Highpass Kaiser Window	Ksr_HPCoef	S
	Bandpass Kaiser Window	Ksr_BPCoef	S
	Bandstop Kaiser Window	Ksr_BSCoef	S
	General EquiRipple FIR	Equi_Ripple	S
	Lowpass EquiRipple FIR	EquiRpl_LPCoef	S
	Highpass EquiRipple FIR	EquiRpl_HPCoef	S

Table 7-1. Analysis Control Function Tree (Continued)

Control	Function	Function Name	CW Version
	Bandpass EquiRipple FIR	EquiRpl_BPCoef	S
	Bandstop EquiRipple FIR	EquiRpl_BSCoef	S
	Windows		
	Triangle Window	TriWin	S
	Hanning Window	HanWin	S
	Hamming Window	HamWin	S
	Blackman Window	BkmanWin	S
	Kaiser Window	KsrWin	S
	BlackmanHarris Window	BlkHarrisWin	S
	Tapered Cosine Window	CosTaperedWin	S
	Exact Blackman Window	ExBkmanWin	S
	Exponential Window	ExpWin	S
	Flat Top Window	FlatTopWin	S
	Force Window	ForceWin	S
	General Cosine Window	GenCosWin	S
	Measurement		
	AC/DC Estimator	ACDCEstimator	S
	Amplitude/Phase Spectrum	AmpPhaseSpectrum	S
	Auto Power Spectrum	AutoPowerSpectrum	S
	Cross Power Spectrum	CrossPowerSpectrum	S
	Impulse Response	ImpulseResponse	S
	Network Functions	NetworkFunctions	S

Table 7-1. Analysis Control Function Tree (Continued)

Control	Function	Function Name	CW Version
	Power Frequency Estimate	PowerFrequency Estimate	S
	Scaled Window	ScaledWindow	S
	Spectrum Unit Conversion	SpectrumUnit Conversion	S
	Transfer Function	TransferFunction	S

Controls

Each analysis function is a method of its corresponding control. Parameters are passed to the functions like any other functions. In many cases, the calculated value or data is returned as a result from the function, instead of using an output variable. This allows you to assign the result of the function directly to another part of the program, such as the user interface, or the parameter of another function. For example:

```
Text1.Text = CWStat1.Mean(Data)
```

Because each function is a method of a control, you must place the correct control in your application to use the function. Additionally, each call to an analysis function includes the name of the control. For example, a call to the `AddArray` function, which is part of the `CWArray` control, would look like this:

```
SumArray = CWArray1.AddArray(Array1, Array2)
```

In functions where the information is not returned from the function, no return variable is assigned. For example:

```
CWDSP1.AutoPowerSpectrum Data, 0.001, Spectrum, deltaF
```

Many parameters passed to the analysis functions are of Variant data type. When these parameters are passed for output or return, they only need to be declared as such. For example:

```
Dim Data as Variant
Data = CWDSP1.SinePattern(1024, 5, 0, 4.5)
```

Analysis Function Descriptions

Because there are many functions in the analysis libraries, individual functions are not described in this manual. Each function, with its purpose and parameters, is described in detail in the online ComponentWorks reference manual. The reference manual also includes code examples for each function. You can access the help file directly from most programming environments. See the chapter in this manual specific to your programming environment for more information.

You can also open the help file by selecting the `ComponentWorks Reference` in the ComponentWorks program group under the **Start** button of the Windows task bar.

Error Messages

If any analysis function encounters an error, it sends an exception back to the application, which displays a dialog box with the error number and description. The analysis functions do not return the error code from the function. Consult the Appendix B, *Error Codes*, for more information about individual error messages and how to resolve them. Error handling and debugging is described in more detail in Chapter 8, *Building Advanced Applications*.

Tutorial: Using Simple Statistics Functions

This tutorial shows how to use some of the statistical functions of the CWStat control. The functions in this tutorial are part of the base analysis library, and are supported by all versions of ComponentWorks.

Place the analysis control containing the functions you are using in your application. You can then use the functions by adding them to your code manually, or using the tools provided by your development environment. The analysis controls do not have any properties that need to be edited. The only property you use with the control is the `Name` property, which you use when calling any function. For example, the default name of the `CWStat` controls is `CWStat1` and a call to the `Mean` function is `CWStat1.Mean`.

This tutorial uses the graph control from the user interface tools to display data arrays. If you do not have the user interface tools, you can still complete this tutorial. Disregard references to the graph control and use your own method for displaying data arrays.

This tutorial uses Visual Basic syntax, but the discussion is in general terms so that 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 on 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 `Stat`.
2. Load the ComponentWorks user interface control, CWGraph, and the CWStat analysis controls into your programming environment. Consult the chapter discussing your environment if you are not familiar with this operation.

3. From the toolbox or toolbar, place a CWStat control on the form. Keep its default name, `CWStat1`.

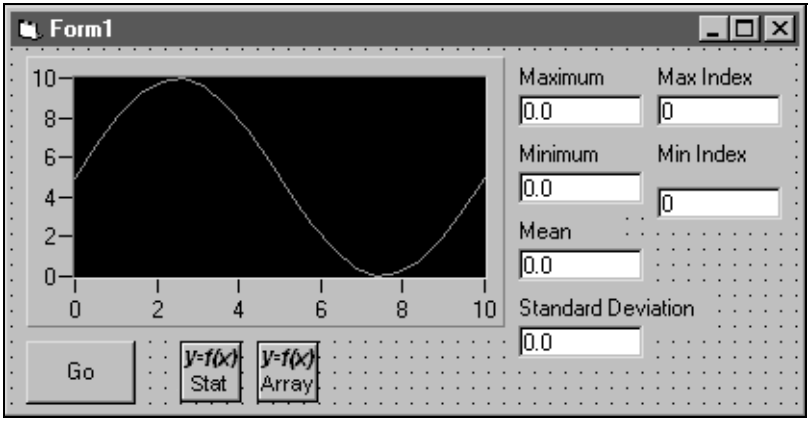
4. From the toolbox or toolbar, place a CWArray control on the form. Name the control `CWArray1`.

5. Place a CWGraph (graph) control on the form. Keep its default name, `CWGraph1`. You can open its property sheet to change any of its properties.

6. Place a button on the form. Change its name and caption property to `Go`.

7. Create six text boxes on the form and name them `Max`, `MaxIndex`, `Min`, `MinIndex`, `Mean`, and `StdDev`.
8. Add a label to each text box with the following descriptions: `Maximum`, `Max Index`, `Minimum`, `Min Index`, `Mean`, and `Standard Deviation`.

Your form should now look similar to the one shown below:



Developing the Program Code

When you press the **Go** button, the program generates an array of random numbers. It displays the data on the graph and also calculates and displays the following statistics of the data set: maximum, array index at maximum, minimum, array index at minimum, mean, and standard deviation.

1. Create a skeleton event handler for the `Click` event of the **Go** button.
   - In Visual Basic, double-click on the button on the form to create the `Go_Click` subroutine.
   - In Visual C++, use the **MFC ClassWizard** to create the event handler routine. Right-click on the button and select **ClassWizard**.
   - In Delphi, use the object inspector to create the event handler routine. Select the **Go** button, then press <F11> to open the object inspector. Select the **Events** tab. Double-click the empty field next to the Click event.
2. Add code inside the event handler routine to generate an array, fill it with random data and display it on the graph. If you are working in Visual C++, you first need to add a member variable for the graph control using the MFC Class Wizard.

```
Dim data(0 To 99)
For i = 0 To 99
    data(i) = Rnd
```

```
Next i
CWGraph1.PlotY data
```

3. Add the function to calculate the statistics of the data set. Use the `StdDev` function of the `CWStat` control to calculate the standard deviation and mean of the dataset, and the `MaxMin1D` function of the `CWArray` control for the remaining values. You also need to declare a number of variables to store the different calculated values. Add the following code to the program. The variable declarations should go at the top of the event handler routine. The analysis functions should go after the call to the `PlotY` method from the previous step.

```
Dim MeanVal as Variant, StdDevVal as Variant
Dim MaxVal as Variant, MaxIndexVal as Variant
  MinVal as Variant, MinIndexVal as Variant
CWStat1.StdDev data, MeanVal, StdDevVal
CWArray1.MaxMin1D data, MaxVal, MaxIndexVal, MinVal,
  MinIndexVal
```

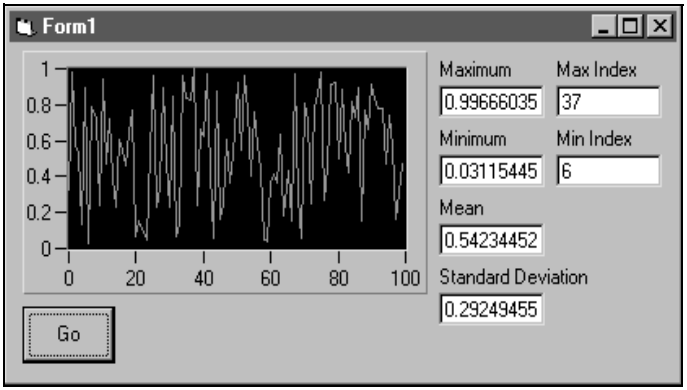
4. Add the necessary code after the analysis functions to display the calculated values in the textboxes on the user interface.

```
Mean.Text = MeanVal
StdDev.Text = StdDevVal
Max.Text = MaxVal
Min.Text = MinVal
MaxIndex.Text = MaxIndexVal
MinIndex.Text = MinIndexVal
```

5. Save the project and associated files as `Stat`.

Testing Your Program

Run the program. Click on the **Go** button to generate a data set and calculate the statistical values. The result should be similar to the following illustration:



When the statistical values are displayed in the text boxes, all digits of precision are displayed by default. You can edit the code which displays these values to limit the number of digits displayed. Consult your programming reference manual for information on how to limit the degree of precision.

You can also use the numeric edit box control to display the values and use its format string to limit the number of digits displayed.

Building Advanced Applications

Chapter 8

This chapter discusses how to build applications using more advanced features of ComponentWorks, including advanced data acquisition techniques, the DSP Analysis Library, and advanced user interface controls. It also discusses error tracking, error checking, and debugging techniques.

Using Advanced ComponentWorks Features

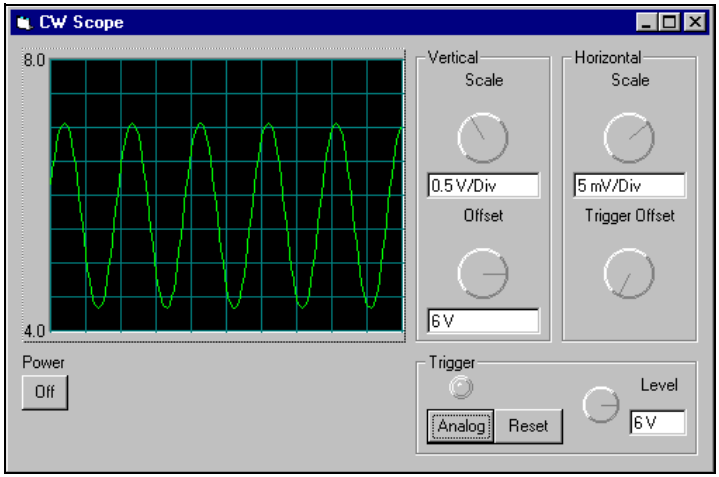
This chapter discusses some of the advanced features in the ComponentWorks controls. It also includes examples that show how the features are applied and used in real applications. You should be familiar by now with the basic operations of the different controls.

This chapter illustrates advanced data acquisition techniques, such as pretriggering and using start, stop, and pause conditions. You will also learn how to incorporate the DSP Analysis Library and use the spectrum functions. Finally, you will use advanced user interface control features, such as graph cursors and multiple axes and pointers.

This chapter discusses only the key features of sample applications that you can find in the `\ComponentWorks\Tutorial` directory. You can customize these examples to implement the advanced features in your own applications.

A Virtual Oscilloscope

The Virtual Oscilloscope application uses the ComponentWorks user interface and data acquisition analog input controls and a DAQ board to build a simple one-channel oscilloscope. Load the sample program from `\ComponentWorks\Tutorials` into your development environment to follow the discussion. The following screen shows the application at run-time:



Depending on the trigger mode button, the application acquires data in a single-shot, continuous, or analog trigger continuous operation. When the data is returned to the `AcquiredData` event, it is plotted on the graph. The vertical controls on the scope adjust the Y axis of the graph, while the horizontal and trigger settings affect the DAQ AI control. Any property settings of the DAQ AI control that the user does not control at run-time, such as the device and channel number, are set directly in the DAQ AI property pages. By default, channel 1 on device 1 is used for the acquisition. Wire your signal accordingly or change these values in the property pages of the AI control.

Data Acquisition Stop Condition Modes

The scope can run in one of three acquisition modes—single, continuous trigger, and analog continuous software trigger—which you select by using the **Single**/**Cont**/**Analog** button in the Trigger section of the form. The first two modes work well with any input signal, while the analog trigger mode works best with a periodic dynamic input signal.

The three trigger modes correspond to three stop condition modes of the data acquisition analog input control. For instance, the callback function for the Analog button programmatically sets a new value for the `CWAI1.StopCondition.Type` according to the state of the button, as shown in the following code for the analog trigger mode:

```
CWAI1.StopCondition.Type = cwaiSWAnalog
```

The DAQ control or header file defines the constant `cwaiSWAnalog` so you can easily set the different modes. You can retrieve any `StopCondition.Type` constant value by using the object browser in Visual Basic, consulting the online documentation, or looking in the header file for your environment. After changing the property value, the event handler of the trigger button also reconfigures and restarts the acquisition. For the analog trigger, it is important to set the trigger level to a value within the range of the input signal.

Data Acquisition Pretriggering

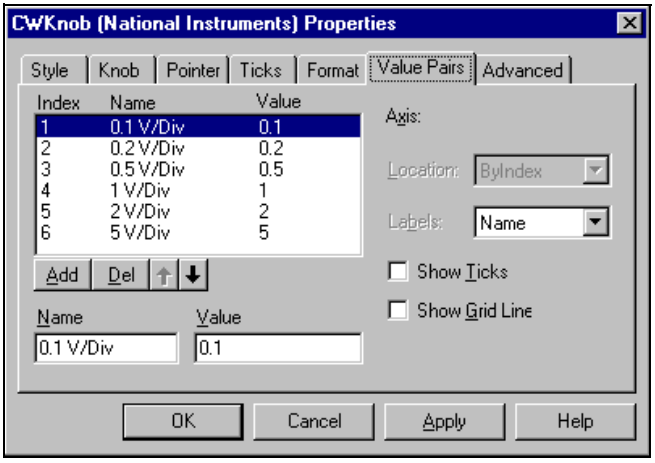
When the scope is in analog trigger mode, you can move the trigger point along the horizontal axis of the graph by using the **Trigger Offset** knob. In the code, the callback for the knob changes the `PretriggerScans` property of the AI control:

```
CWAI1.StopCondition.PreTriggerScans = TOffset.Value
```

Without pretriggering, the application acquires all scans after the stop condition trigger (analog trigger). When the number of PreTrigger scans is greater than zero, the application acquires the number of scans specified for the PreTrigger before the trigger occurs. After the trigger occurs, the array of data returned to the `CWAI1_AcquiredData` event handler contains data both from before and after the trigger. You can set the `PretriggerScans` property and all the stop trigger properties through the property pages of the DAQ AI control. Pretriggering works the same way with the analog software, the analog hardware, and digital hardware stop triggers.

User Interface Value Pairs

Value pairs are user interface control features for assigning names to specific values on a scale or axis, used in the graph, slide and knob controls. All the value pairs for a given axis are stored in the `ValuePairs` collection object of that axis. Each value pair object consists of a name and a value corresponding to a scale or axis of the particular control. You usually set value pairs through the property page of the control. You can also add, edit, or delete them programmatically. The following screen shows the value pairs property page for the Vertical Scale Knob control on the Virtual Oscilloscope form:



After you assign value pairs, such as the settings for the **Vertical Scale** knob, you can limit the allowed values for a control to the predefined value pairs. To do this, set the control to `Value Pairs Only` in the **Style** folder of the property sheet. In the Virtual Scope application, the vertical and horizontal scale knobs are Value Pairs Only controls. This way, the control is limited to preset settings and the program can retrieve the name, value, or index of the currently selected value pair. The application uses the value of the value pair to update the appropriate property on the graph or DAQ AI control, and uses the name of the value pair to update the appropriate text display. The

following code shows how to retrieve these three components of the currently selected value pair from a knob control.

Value:

```
CWKnob1.Value
```

Name:

```
CWKnob1.Axis.ValuePairs(CWknob1.ValuePairIndex).Name
```

Index:

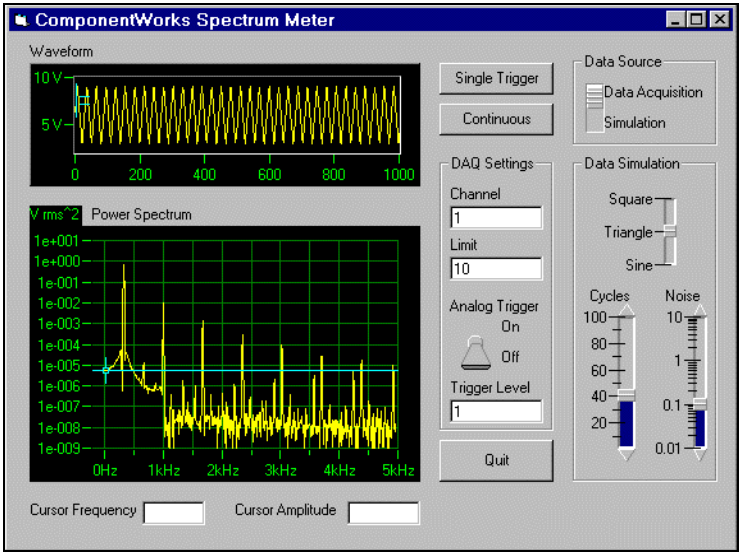
```
CWKnob1.ValuePairIndex
```

Value pairs are part of only one axis of a control, because you can specify multiple axes on a graph and associate value pairs with each individual axis.

Virtual Spectrum Meter

Find the Spectrum Meter application in `\ComponentWorks\Tutorial`. The Virtual Spectrum Meter application uses the DSP analysis functions to build a simple spectrum analyzer. The data can either be acquired with a DAQ board or simulated. If you do not have a DAQ board, select `Simulation` with the **Data Source** switch and characterize your signal in the Data Simulation section. The UI controls are used to

display the information as well as control the operations of the program. The following screen shows the application.



Once the Spectrum Meter is started, you can select either a Single Trigger or a Continuous Trigger acquisition. The single trigger takes a snapshot of the incoming signal so you can study the spectrum of the acquired signal. Use the continuous trigger to monitor changes in a signal as they are occurring. You can set some of the DAQ parameters, such as the channel, using the DAQ settings section of the user interface. The input limit specifies the maximum expected absolute voltage on the input signal, and determines the optimal gain to use on the acquisition process. You can also use an analog trigger with your data acquisition. Set other DAQ parameters, such as the device number, in the property pages of the AI control.

Use the cursors on the graphs to measure the amplitude of the incoming waveform and the frequency of any specific point in the spectrum. When you move one of the cursors, the corresponding display at the bottom of the user interface is automatically updated.

DSP Analysis Library

The digital signal processing (DSP) analysis functions are part of the ComponentWorks Standard and Full Development Systems. DSP functions include Fourier and Hartley transforms, spectrum analysis, convolution and correlation of data sets, and digital windowing and filtering of data. If you have installed the ComponentWorks Base Package, you will not be able to run these functions from your development environment. You can still examine the Spectrum Meter project and code and run the precompiled executable in the `\ComponentWorks\Tutorial` folder. Using the DSP functions is similar to using the Base Analysis Library functions covered in Chapter 7, *Using the Analysis Controls and Functions*. To use the DSP functions, you must first load the analysis control into your environment, and then place the CWDSP control on your form.

The Virtual Spectrum Meter application contains all the analysis functions in the `AnalyseAndGraph` subroutine. The application passes the data (acquired or simulated) to this routine for analysis and display.

The following is the Visual Basic code of the `AnalyseAndGraph` subroutine:

```
Private Sub AnalyseAndGraph(WaveformData() As Variant,
  SampleInterval As Double)

    Dim WindowedData As Variant
    Dim FreqInterval As Variant
    Dim SpectrumData As Variant
    Dim HalfSpectrumData As Variant

    WaveformGraph.PlotY WaveformData
    WindowedData = CWDSP1.HamWin(WaveformData)
    CWDSP1.AutoPowerSpectrum WindowedData,
      SampleInterval, SpectrumData, FreqInterval,
      SpectrumData, freqinterval
    HalfSpectrumData = CWArray1.Subset1D(SpectrumData,
      2, 510)
    SpectrumGraph.PlotY HalfSpectrumData, FreqInterval,
      FreqInterval * 2
End Sub
```

The analysis procedure consists of the Hamming window (`HamWin`) and the auto power spectrum function (`AutoPowerSpectrum`). The Hamming window reduces the effect of spectral noise leakage in the spectrum due to the finite length sample of a continuous signal. The auto power spectrum function is a single-sided, scaled spectrum, which the application can graph directly. The `AutoPowerSpectrum` function also calculates the frequency resolution of the spectrum (`FreqInterval`) using the `SampleInterval` value passed to `AnalyseAndGraph`. The frequency resolution is used in the `PlotY` method of the graph for scaling the X axis of the spectrum graph.

The DSP Analysis Library also includes a function named `SpectrumUnitConversion` for scaling the calculated spectrum between different formats, including linear, dB, and dBm, combined with Vrms, $Vrms^2$, Vpk, and Vpk^2, as well as amplitude and power spectral densities. Consult the online reference for more information on the different DSP functions.

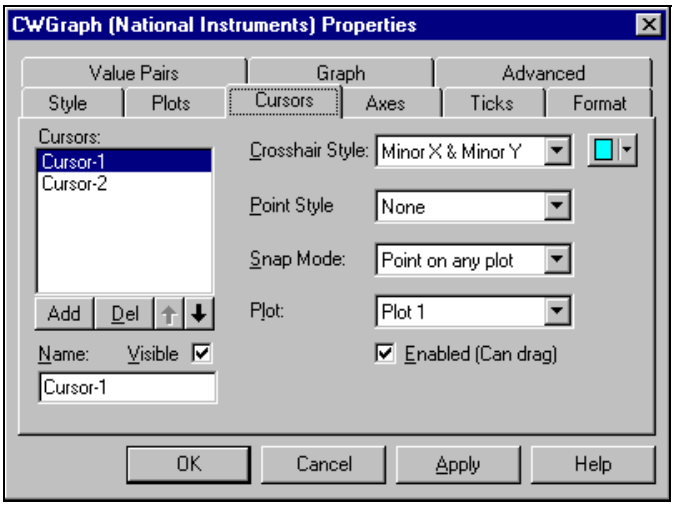
For low-level spectrum calculations, the DSP functions include FFT and inverse FFT algorithms. Time domain functions, such as Convolution, Correlation, Differentiate, and Integrate, are included along with a number of different windowing, IIR (infinite impulse response), and FIR (finite impulse response) digital filtering functions.

Cursors

The Virtual Spectrum Meter application form has cursors on the graph for marking sections of the plots. Use the two cursors on the waveform graph to mark the minimum and maximum values of the acquisition waveform and measure the waveform amplitude. The cursor on the spectrum graph marks a particular point in the spectrum and displays the associated frequency.

To use a cursors on a graph, create and configure one or more cursors in the property pages of the graph. You create additional cursors by pressing the **Add** button in the **Cursors** tab of the property pages, and then configure each cursor with the properties on the right. The Snap Mode of the cursor specifies whether the cursor jumps (snaps) to a point on the nearest plot or can be placed freely on the graph. When you select `Point on selected plot` for the Snap Mode, you can also specify a particular plot to which the cursor is connected.

The following screen shows the property pages for the cursors on the waveform graph of the Virtual Spectrum Meter application:



Individual cursors are represented in the object hierarchy by `Cursor` objects contained in the cursors collection of the graph control. You can manipulate individual cursors programmatically by following the normal conventions of working with collections and their objects. For example, in the program, the cursors are referenced with the name of the graph, cursor index, and cursor property.

```
Max = WaveformGraph.Cursors.Item(2).YPosition
```

See the chapter specific to your development environment and Chapter 5, *Using the Graphical User Interface Controls*, for more information on how to use the user interface controls.

Use event handler subroutines associated with the graph to process any interactions by the user with the cursors. There are four events on the graph relating to the cursors, of which the `CursorChange` and `CursorMouseUp` routines are the most commonly used. To generate one of these event handlers, select the graph and the corresponding cursor event according to your development environment. An event handler skeleton similar to the one below is generated:

```
Private Sub Graph_CursorChange(CursorIndex As Long, XPos
  As Variant, YPos As Variant, bTracking As Boolean)

End Sub
```

The event handler routine automatically provides the index of the cursor (used to determine which of multiple cursors generated the event), as well as the X and Y coordinates of the cursor on the graph. Place the commands to perform your tasks, such as displaying the position of the cursor, in the event handler routine. For the waveform graph cursors, the Virtual Spectrum Meter application reads the Y position of the two cursors to calculate the amplitude of the waveform data, as shown below:

```
Private Sub WaveformGraph_CursorChange(CursorIndex As
  Long, XPos As Variant, YPos As Variant, bTracking As
  Boolean)
    Dim Amplitude As Double
    Amplitude =
      Abs(WaveformGraph.Cursors.Item(1).YPosition -
      WaveformGraph.Cursors.Item(2).YPosition)
    CursorAmplitude = CStr(Round(Amplitude, 2)) + " V"
End Sub
```

Graph Track Mode

The `TrackMode` property of the graph determines how the graph reacts to interactions with the mouse and what events are generated by the graph at these times. By default, the `TrackMode` property is set to `cwGTrackDragCursor`, which allows the mouse to move the cursors on the graph. Moving a cursor generates the `CursorChange` event.

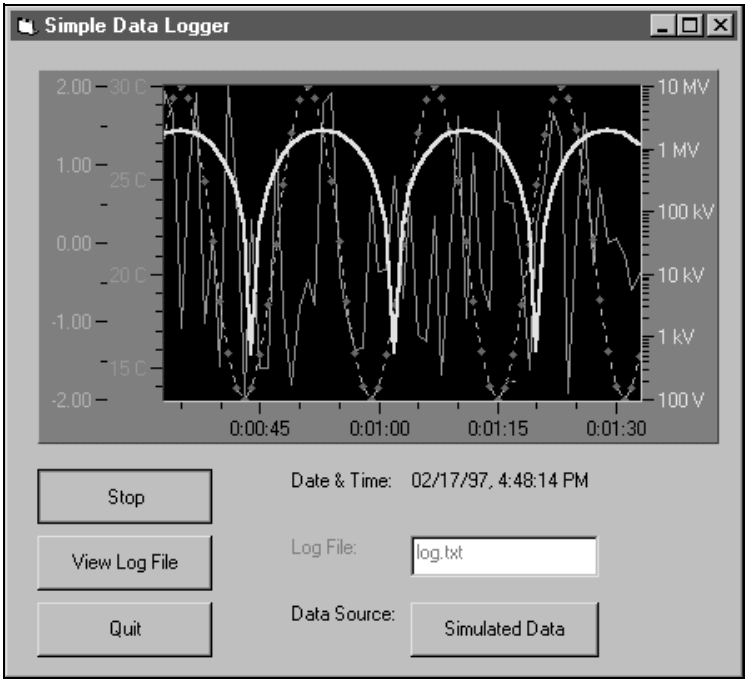
By setting the `TrackMode` property of the graph, either through the property sheets or programmatically, you can choose other selections for the Track Mode property. This setting generates events when the mouse interacts with the plot area as a whole, or individual plots on the graph. With these modes, you can use the mouse to select a specific plot or detect when the user moves the mouse over the plot area.

You can also use `TrackMode` to select panning and zooming. When `TrackMode` is set to **Pan**, any click-and-drag action on the graph shifts the contents of the graph following the mouse movement. When `TrackMode` is set to **ZoomRect**, any click-and-drag action on the graph draws a rectangular outline on the graph. When the mouse button is released, the graph zooms to the dimensions of the outline. For these two operations, you can select X axis, Y axis, and XY axis modes, limiting the motion to the specified axes. Consult the user interface

folder of the samples directory (`\ComponentWorks\Samples\…`) for an example of how to use these track modes.

A Virtual Data Logger

The Virtual Data Logger application records real time phenomena on multiple channels at slow rates over an extended period of time. It can either simulate the data being acquired or use a data acquisition card. The acquired data is logged to a serial ASCII file. Saving data in ASCII format allows you to read the file in many other applications, but it makes the file larger in size than if you saved the data in binary format. You can open the saved file from the virtual data logger to inspect the file. The following figure shows the application:



The data source is set to simulated data by default, but you can also acquire data from a DAQ board. If you use a DAQ board to acquire the data, set the properties for the DAQ AI control accordingly.

The **Start** button is an On/Off style for starting and stopping the acquisition/logging process. The **View Log File** button opens the recorded file using the Windows Notepad application. By default, the

application does not clear the log file when you start the logging process. Rather, it appends the data to any existing data in the file. You can change this feature in the program.

Multiple Graph Axes

You can configure multiple Y axes for a graph by using its property sheets. In the **Axes** folder of the property sheet, use the **Add** and **Del** buttons to add and delete axes. For each axis, select the range and style you want to use. After configuring the axes, switch to the **Ticks** folder in the property sheet, select each axis in the pull-down ring, and configure the labels and ticks for each axis.

In the same way you configure multiple axes or cursors, you can predefine the properties of multiple plots on the graph by using the **Plots** folder of the property sheet. By default, each plot is assigned to the same Y axis on the graph. Using the **Plots** folder, you can assign plots to different Y axes of a graph. When an application plots or charts data on the graph, the Y axis specified for a particular plot determines the scaling of that plot. If you generate more plots on a graph then there are defined plots, the Template plot style and the first Y axis are used for each undefined plot.

You can set any of the properties of the plots or axes programmatically. Both the plots and axes are individual objects (of type Plot and Axis) which are stored inside of corresponding collection objects (Plots and Axes, respectively) on the graph. To change a property of an individual axis or plot, you must first select the collection and then the individual object in the collection. You can then specify the property to read or write.

For example, the collection of plot objects is `Plots`, but you reference the individual plot objects by using the `Item` method on the collection. The methods of the Plots collection—such as `Add`, `Item`, and `Remove`—make changes to the collection as a whole, while the properties of the `Plot` object—such as `Name`, `AutoScale`, and `LineColor`—are the individual settings for one plot.

Objects in a collection are referenced by their one-based index using the `Item` method. For example, `CWGraph1.Axes.Item(1)` refers to the first axis on the graph (X axis). `CWGraph1.Plots.Item(2)` refers to the second plot on the graph (default name `Plot-2`). `CWGraph1.Axes.Item(2).Name` retrieves or sets the name of the second axis (first Y axis).

To assign simple properties—properties that contain boolean, double, integer, variant, or string values—to a plot or axis, simply use the property name like any other variable in your program. For example, to change the name of plot on a graph, use the following code:

```
CWGraph1.Plots(2).Name = “Temperature”
```

Assigning an object as a property to another object is similar. For example, because each plot on a graph is assigned a specific Y axis which is used for scaling of the data, it is possible to assign a new axis (an object) as a property to a plot. In most programming environments, you assign the object like you would any other variable. Doing such object assignments in Visual Basic requires the `Set` keyword, which differentiates the assignment of data from the assignment of objects. The following Visual Basic code, including the `Set` statement, assigns the third axis as an object to the `YAxis` property of the first plot:

```
Set CWGraph1.Plots.Item(1).YAxis =
  CWGraph1.Axes.Item(3)
```

Consult the Visual Basic documentation for more information on the `Set` statement.

Graph Axes Formats

The axes on the graph, as well as the slide and knob, support special formatting modes you can use to edit the labels on the tick marks. Besides doing simple formatting like using exponential or engineering notation, you can also add alphanumeric characters such as units (for instance, `Hz` or `V`), and convert to currency, percentage, or time formats. The time format includes options both for time and date. Two more advanced formatting modes are scaling and symbolic engineering.

Use the scaling format mode to perform simple mathematical functions (addition, subtraction, multiplication and division) automatically on your data before displaying it on your graph. For example, if you are using an IC temperature sensor whose output in volts corresponds to the temperature divided by 100, you can specify your axis to automatically scale the data by `*100`. The data displayed on this axis is automatically multiplied by 100 without any changes in the code of your program.

Symbolic engineering is normally used when displaying units with your tick labels. The use of the symbolic engineering format automatically adds prefixes such as k for kilo and m for milli before the units instead of using the exponent. This becomes especially useful when using logarithmic scaling.

You normally set the formatting of the axis in the property pages of the control, but it can also be set programmatically.

File Input/Output

The file I/O functions in the Virtual Data Logger application are an example how to perform simple file I/O in a program. Because file I/O is not part of the ComponentWorks functionality, the functions used will vary between different programming environments. Consult the data logger example in your programming environment for an example of how to perform file I/O. All the file I/O functions of the application are in the `LogData` subroutine, which you can use as a template for other input/output routines. The application uses a sequential ASCII file to store the data and appends new data to any existing data already in the specified file. Consult the reference manual and other example programs for your programming environment for more information on the file I/O functions. You can start with the names of the functions used in the data logger for searches in the reference manual.

Adding Testing and Debugging to Your Application

The tools available for debugging vary depending on your programming environment, but normally include features such as Breakpoints, Step-Run mode, and a Watch Window. Consult the reference manual for your programming environment for information on these and other debugging tools.

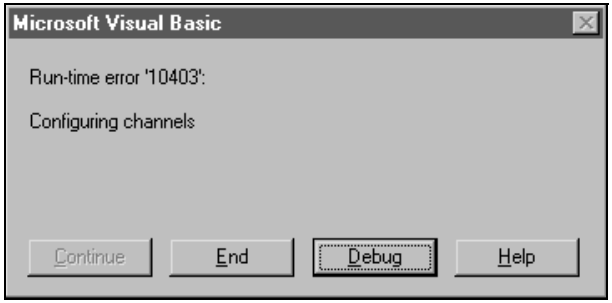
Error Checking

ComponentWorks controls can report error information to you and to the application in a number of different ways: by returning an error code from a function or method call, by generating an error or warning event, or by throwing an exception handled by your programming environment. The type of error reporting used depends on the type of application and the preference of the programmer.

By default, all the ComponentWorks controls generate exceptions when errors occur, rather than returning error codes from the methods. However, the DAQ controls have a property, `ExceptionOnError`, you can set to `False` to use return error codes instead of exceptions. Additionally, error events are fired by the DAQ controls if an error occurs during specific phases (contexts) of an acquisition process. The contexts for which error events are generated are set in the `ErrorEventMask` property of the DAQ controls.

Exceptions

Exceptions are error messages returned directly to your programming environment. Exceptions are normally processed by displaying a default error message. The error message usually allows you to end your application or to go into a debug mode where you can perform certain debugging functions. Part of the exception returned is an error number and error description, displayed as part of the error message. For example the DAQ AI control may return the following exception to Visual Basic:



Depending on your programming environment, you may be able to insert code that will catch exceptions being sent to your application and handle them in another manner. In Visual Basic you can do this by using the `On Error` statement. The two main implementations are `On Error Resume Next` and `On Error GoTo`. Following are two examples of error handling using these conventions.

- `On Error Resume Next` does not automatically generate an error message, but continues running the program at the next line. To handle the error, you should check and process the information in the `Err` object in your code:

```
Private Sub Acquire_Click()
On Error Resume Next
   CWAI1.Configure
   If Err.Number <> 0 Then MsgBox "Configure: " +
     CStr(Err.Number)
   CWAI1.Start
   If Err.Number <> 0 Then MsgBox "Start: " +
     CStr(Err.Number)
End Sub
```

- `On Error GoTo` also avoids using the default mechanism for handling the exception, running instead the section of your program you specify. You can define one error handler in your subroutine:

```
Private Sub Acquire_Click()
On Error GoTo ErrorHandler
   CWAI1.Configure
   CWAI1.Start
   Exit Sub
ErrorHandler:
   MsgBox "DAQ Error: " + CStr(Err.Number)
   Resume Next
End Sub
```

If you are not using Visual Basic, consult the documentation for your programming environment for information on how to handle exceptions.

Return Codes

If the `ExceptionOnError` property is set to `False`, the DAQ control methods return a status code to indicate whether an operation completed successfully. If the return value is something other than zero, it indicates a warning or error. A positive return value indicates a warning, signifying that a problem occurred in the operation, but that you should be able to continue with your application. A negative value indicates an error which means a critical problem has occurred in the operation, and that all other functions or methods dependent on the failed operation will also fail.

You can use the specific value of the return code for more detailed information about the error or warning. The ComponentWorks DAQ Controls can convert the error code into a more descriptive text message, as described in the *GetErrorText Function* section of this chapter.

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

```
lerr = CWAI1.Start

If lerr <> 0 Then MsgBox "Error at DAQ Start: " +
  CStr(lerr)
```

A common method in Visual Basic for displaying error information to the user is to use the `MsgBox` pop-up window. Normally, you can write one error handler for your application instead of duplicating it for every call to a function or method. For example, the following code shows how to create a `LogError` subroutine to use with the `Start` method, as well as for use with later functions or methods:

```
Private Sub LogError(code As Long)

   If code <> 0 Then
      MsgBox "DAQ Error: " + CStr(code)
   End If
End Sub
```

To use the `LogError` subroutine, place the call to `LogError` in front of every function or method call. The return code is automatically passed to `LogError` and processed.

```
LogError CWAI1.Start
```

If you are working in other environments, consult the documentation to determine how to display a simple dialog window or find another method of displaying error information.

Error and Warning Events

The DAQ controls in ComponentWorks also include their own error and warning events—`DAQError` and `DAQWarning`. You normally use return codes for error checking of the methods used on the DAQ controls. However, for asynchronous operations such as a continuous analog input, you cannot use return codes for error checking because no methods are called after the first `Start` method. In this case, the CWAI control generates its own error event if an error or warning occurs during an on-going acquisition. You can develop an event handler to process these error and warning events. The following code shows the skeleton event functions for the CWAI control:

```
CWAI1_DAQError(ByVal StatusCode As Long, ByVal ContextID
  As Long, ByVal ContextDescription As String)
```

```
CWAI1_DAQWarning(ByVal StatusCode As Long, ByVal
  ContextID As Long, ByVal ContextDescription As String)
```

You can create both of the above routines using your normal method for generating event handlers. The `StatusCode` variable automatically passed to the event handler contains the value of the error or warning condition. The `ContextID` contains a value describing the operation where the error or warning occurred, and the `ContextDescription` contains a more descriptive string describing the operation where the error or warning occurred.

The following code shows an example of how to use the `AI_DAQError` event in a Visual Basic application:

```
Private Sub CWAI1_DAQError(ByVal StatusCode As Long,
  ByVal ContextID As Long, ByVal ContextDescription As
  String)
    MsgBox ContextDescription + ":
    CStr(StatusCode)
End Sub
```

This code produces the following error message box:

By default, only asynchronous operations call error and warning events. You can set the `ErrorEventMask` property of the CWAI control to specify the operations for which the error and warning events are called. Consult the online reference for more information on how to set the `ErrorEventMask` property.

GetErrorText Function

If you use the return error code to perform your error checking for data acquisition controls, you may want to convert the error code values into more descriptive error texts. The ComponentWorks DAQ controls include a utility control that includes a method to convert error codes into descriptive error strings. To use this method, create a `DAQTools` control in your program and use the `GetErrorText` method as shown in the following example:

```
DAQError = CWAI1.Start

If DAQError <> 0 Then MsgBox
  CWDAQTools.GetErrorText(DAQError)
```

The following screen shows a message box generated by using the `GetErrorText` function in the previous example:

Debugging

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

Debug Print

One of the most common debugging methods is to print out or display key variables throughout the program execution. You can then monitor critical values and determine when your program varies from the expected progress. Some programming environments have dedicated debugging windows which are used to display such information without disturbing the rest of the user interface. For example, you can use the `Debug.Print` command in Visual Basic to print information directly to the debug window:

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

Breakpoint

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

Stopping at a breakpoint confirms that your application ran to the line of code containing the breakpoint. If you are unsure whether a specific section of code is being called, place a breakpoint in the routine to find out. Once you have stopped at a specific section of your code, you can use other tools, such as a watch window or debug window, to analyze or even edit variables. Consult your environment documentation to determine how to set breakpoints.

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

Watch Window

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

Single Step, Step Into and Step Over

Use *single stepping* to execute a program one line at a time. This way, you can check variables and the output from your program during execution. Single stepping is commonly used after a breakpoint to step though a questionable section of code slowly. There are several variations to the single step mode which may be supported in your environment.

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

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

Common Questions

Appendix A

This appendix contains a list of answers to frequently asked questions. It contains general ComponentWorks questions as well as specific data acquisition, graphical user interface, and analysis library questions.

ComponentWorks Common Questions

Installation and Getting Started

What does it mean when I place a ComponentWorks control on my form and get an error from Visual Basic saying I'm not licensed to use this control?

This error means that ComponentWorks was not installed properly. Make sure to install ComponentWorks on your computer using your installation disks. Copying the files or sharing them over a network does not work.

What is the difference in Visual Basic 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 can also 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
```

I don't see any new controls in my Visual Basic toolbox. How do I load the ComponentWorks controls into Visual Basic?

To load the ComponentWorks controls in Visual Basic, right-click on the toolbox and select **Custom Controls...** from the popup menu. In the

dialog box, click on the **Browse** button to select a new control. The ComponentWorks controls are located in the `\Windows\System` directory and are named `CWDAQ.OCX`, `CWUI.OCX` and `CWANALYIS.OCX`, and so on. Select each of the controls and then click **OK** to return to Visual Basic. The new controls will be placed in the toolbox.

Load the ComponentWorks utility library by selecting **References** from the **Tools** menu and selecting the `CWUTILS.DLL` file (`\Windows\System` directory) through the **Browse** button.

How do I have the ComponentWorks controls and libraries loaded automatically when I start Visual Basic?

In Visual Basic 4 you can have the ComponentWorks controls and libraries loaded by adding them to the `AutoLoad` project. To do this, open the project `AUTO32LD.VBP`, found in your Visual Basic directory. Then load the controls and references and save the project.

What is the object browser in Visual Basic, where do I find it, and how do I use it?

The **Object Browser** is a tool in Visual Basic that displays the defined object types of the currently loaded controls and their properties and methods. It specifies the data type of the property for each property (which can be another object). For methods, it shows the prototype of each method, including its parameters.

Open the object browser in Visual Basic by pressing <F2> or selecting **View Object Browser...** from the **View** menu. The object browser is divided into three fields.

In the top field, select a library or project.

The left field displays all defined classes and controls for the selected project, such as `CWGraph` in the `CWUIControlsLib` project.

After you select a class or module, the right field displays all the properties, methods or functions of the selected class or module. When you select one of these elements in the right window, the bottom of the object browser displays the prototype of the selection, including any reference to another object type (class).

How do I load the ComponentWorks controls in Borland Delphi?

Follow these steps to load the ComponentWorks control in Delphi.

1. Save all your work in Delphi including files and projects.
2. Select **Install...** from the **Component** menu
3. In the **Install Components** window press the **OCX** button.
4. In the **Import OLE Control** window, select the registered control you want. The ComponentWorks controls all start with *National Instruments*. Click **OK** to close the window.

   If your control does not show in the **Import OLE Control** window, it is not registered with the operating system. Click on the **Register...** button, find the `.OCX` file that contains the control, and select it. This registers the control with the operating system. Most `.OCX` files are located in the `\System(32)` directory under Windows. The ComponentWorks `.OCX` files start with *CW*.
5. To load more controls, repeat steps 3 and 4.
6. When you are done selecting controls, click the **OK** button in the **Install Components** window. This loads the new controls and adds them to the OCX tab of the Component Palette. This step also close any open project and you will need to reopen it.

How do I distribute an application using ComponentWorks?

To distribute an application using ComponentWorks or any ActiveX controls, you need to distribute all the `.OCX` files, `.DLL` files and supporting OCXs and DLLs referenced in the application. In addition you 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. You can do this with an installer, which you build with the Setup Wizard/Tool provided by your programming environment, or manually using the `REGSVR32.EXE` utility.

To install and register an `.OCX` file, copy it to the `\System` (for Win95) or `\System32` (for WinNT) subdirectory of the Windows directory on the target computer. Then run:

```
regsvr32 c:\windows\system(32)\<ocxname>.ocx
```

To unregister a control use:

```
regsvr32 /u c:\windows\system(32)\<ocxname>.ocx
```

If you distribute the ComponentWorks OCXs, you also need to make sure that the following support DLLs are installed on the target computer. The DLLs are listed with the minimum required version number.

`stdole2.tlb`	`2.20.4054`
`mfc42.dll`	`4.2.6256`
`msvcirt.dll`	`4.20.6201`
`msvcrt.dll`	`4.20.6201`
`msvcrt40.dll`	`4.2000.6164`
`oleaut32.dll`	`2.20.4054`
`olepro32.dll`	`5.0.4055`

These files are located in the `\Redist` directory on the ComponentWorks installation CD. In addition, using Internet images in the UI controls requires `URLMON.DLL, WININET.DLL` and other files and registry settings. If Microsoft Internet Explorer 3.01 or later is installed on the system, the correct files and registry settings will be present. The CW installer does not install these files.

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

Analysis Controls

My analysis functions seem to have no effect. What does the return code -30008 mean?

To use the analysis functions in your code, you must place the analysis key icon on your form. Without this icon, the functions return the error `-30008, kErrFunctionNotLicensed`, and have no effect. If you have the analysis key icon on your form, you are calling a function not supported by your version of ComponentWorks. Refer to Chapter 7, *Using the Analysis Controls and Functions*, for more information.

Data Acquisition Controls

What methods and events do I need to use for the DAQ analog input control?

The DAQ AI control has four main methods (Configure, Start, Stop, Reset). The methods are listed and described in the *ComponentWorks Getting Result* manual and in the ComponentWorks online help.

After you set the properties of the AI control in the property pages, call the `Configure` method to pass the property values to the driver and hardware. Then call `Start` to start the actual acquisition. Use the `Stop` method to stop a continuous acquisition, and the `Reset` method to stop any acquisition and reset the driver and hardware. You need to call `Configure` again any time you change a property value programmatically, after you call the `Reset` method, or after the acquisition stops due to an error.

The main event used with the AI control is the `AcquiredData` event, which is called when a set number of points is acquired and which returns the data to the program. Two additional events, the `DAQError` and `DAQWarning` events, are generated in response to any errors or warnings that occur in the DAQ process. The `ErrorEventMask` property determines which operations (contexts) of the AI control these events are called for. Check the online reference for more detailed information on `ErrorEventMask`.

How do I assign new channels dynamically or retrieve channel information from the DAQ AI control in my program?

Use the `Channels` collection object of the AI control to retrieve information from individual channel objects. Use the syntax `CWAI.Channels.Item(n)` to access individual channel objects.

```
Dim ChannelInfo As String
ChannelInfo = CWAI1.Channels.Item(1).ChannelString
```

Use `CWAI.Channels.RemoveAll` and `CWAI.Channels.Add` with the appropriate information in the parameter list to delete all the channel objects in the channel list (`Channels` collection) and to add new channel objects to the collection.

```
CWAI.Channels.RemoveAll
CWAI.Channels.Add 1
```

Refer to the properties on `CWAI.Channels` in the online reference manual for more information.

Why does my EISA-A2000, AT-A2150, AT-DSP2200 data acquisition card not work with ComponentWorks?

These three boards are not compatible with the ComponentWorks data acquisition controls.

How do I pass an array to CWAI.AcquireData? I keep getting type mismatches on my declared arrays. What is a Variant data type?

The two data buffers (`Voltages` and `BinaryCodes`) of the `AcquireData` method are defined as variant data types. Do not declare them ahead of time as arrays. Select two variable names, declare them as `Variant`, and pass them to `AcquireData`. `CWAI.AcquireData` redeclares the variables to the correct array data type and array size.

Variant data types are used when the resultant data type may not be known ahead of time. This allows a function or method to redeclare the variant variable for the appropriate type.

```
Dim Voltages As Variant
Dim BinaryBuffer As Variant
Dim Timeout As Single
Timeout = 5
CWAI.AcquireData (Voltages, BinaryBuffer, Timeout)
```

When performing a single channel acquisition, both data buffers will be declared as one-dimensional arrays; a multi-channel acquisition declares two-dimensional arrays.

User Interface Controls

How do I add labels to the ComponentWorks graph axes?

ComponentWorks 1.1 and later includes a `Caption` property for the graph axes, which you can use to label each axis.

How do I show gridlines on my graph without displaying the scales similar to an oscilloscope screen?

In the **Ticks** section of the property pages of the graph enable the grid lines for the selected axis and disable all the ticks and labels for the same axis. Make sure to keep the `Visible` property of the axis enabled.

How do I set the default value for a control such as the knob or slider?

To set the default value for a knob, slider or switch, in Visual Basic or Delphi open the default property page (<F4> in Visual Basic, <F11> in Delphi) and set the `Value` property to the desired value.

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

How do I plot my data on the graph?

The easiest way to plot data is to use the `PlotY` method. This displays a simple plot of your values.

```
' dataArray can be an array or a variant containing
’ an array
CWGraph1.PlotY dataArray,0,1, True
```

Additional methods you can use to plot or chart data are `PlotXY`, `PlotXvsY`, `ChartY`, `ChartXY`, and `ChartXvsY`. The Plot methods plot a whole data set at once, deleting any previous information on the same plot. You can define multiple plots on a graph in the property pages and use the `Plot` methods to update individual plots.

The `Chart` methods append data to existing plots and are used to create scrolling charts. Consult the online reference documentation for more detailed descriptions of these methods.

How do I display a value on a slide or knob and how do I read values back from them? How do I read or set a button?

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 like 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 slide to 5
CWSlide1.Value = 5
‘read back the value from a knob
Dim ReadValue As Double
ReadValue = CWKnob1.Value
```

Buttons work in the same way, except that their values are booleans.

```
‘set a button
CWButton1.Value = True
‘read a button
If CWButton1.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 provides functions to read and write the value of that 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 slide 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_Slide.SetValue(5);
```

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

```
m_Meter.SetValue(m_Slide.GetValue());
```

The names of all the property functions for a given control can be seen in the **ClassView** of the **Project Workspace** in Visual C++. In the **Project Workspace**, select the **ClassView** and then select the desired control/object to view its property functions (as well as methods).

How can I change the style of my button, knob, or slide programmatically?

The button, knob and slide control each have a number of default styles that you can choose in the property pages of the control. In some applications you may want to switch the style of a control while the program is running.

It was not possible to change the style of a control programmatically using ComponentWorks 1.0(1). With 1.1 controls, however, you can use the `SetBuiltinStyle` method of these controls to change the style at run-time. The different styles are defined as constants in the UI controls.

```
CWSlide1.SetBuiltinStyle cwSlideStyleTank
CWKnob1.SetBuiltinStyle cwKnobStyleDial
CWButton1.SetBuiltinStyle cwButtonStyleRoundLED
```

How do I access or change a particular axis, plot, cursor, value pair on one of the UI controls?

Each of these objects, with the exception of the axis on the knob and slide, is normally contained within a collection object on the control. A collection object is a special object on a control which is used to store multiple objects of the same type. For example a graph may have many different axes. Rather than linking each `Axis` object directly to the graph control, one `Axes` collection is linked to the graph and it contains all of the axes of the graph.

Other objects which are contained in collections are `Plot`, `Cursor`, `ValuePair` and the `Channel` object on the data acquisition controls. The name of the collection object corresponding to one of these objects is the name of the contained object in plural form. For example, the collection of `Axis` objects is `Axes`; `Plot` is `Plots`; `Cursor` is `Cursors`; and so on.

The `ValuePair` object is contained in the `ValuePairs` collection, itself is part of the `Axis` object, which in turn is either part of the knob or slide control, or contained in the `Axes` collection of the graph.

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 graph use

```
CWGraph1.Plots.Item(1)
```

This code segment refers to the first plot on a graph as an object. You can then access the properties of the object by appending the name of the property. To read the X position of the second cursor on a graph use the following code.

```
x = CWGraph1.Cursors.Item(2).XPosition
```

The properties of the individual object are described in detail in the online reference manual. Search for the corresponding object name such as `CWAxis`, `CWPlot`, `CWCursor`, and so on, to find the description. Each of the collection objects also has a number of its own properties and methods.

You can use the `Count` property to determine the number of objects in a collection.

```
NumAxes = CWGraph1.Axes.Count
```

Use the `Add`, `Remove` and `RemoveAll` methods to programmatically change the number of objects in a collection. This way 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.

```
CWGraph1.Axes.Add
CWSlide1.Axis.ValuePairs.Remove 3
CWGraph1.Cursors.RemoveAll
```

Remember that value pair 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.

```
CWKnob1.Axis.ValuePairs.Item(1).Name = “Maximum”
CWGraph1.Axes.Item(3).ValuePairs.Item(2).Value = “7.5”
```

Error Codes

Appendix B

This appendix lists the error codes returned by the ComponentWorks DAQ controls and Analysis Library functions. It also lists some general ComponentWorks error codes.

Table B-1 lists the negative error codes returned by the DAQ controls. Each DAQ control returns an error code that indicates whether it executed successfully. When a control returns a code that is a negative number, it means that the control did not execute.

Table B-1. Data Acquisition Control Error Codes

Code	Name	Description
-10001	syntaxError	An error was detected in the input string; the arrangement or ordering of the characters in the string is not consistent with the expected ordering.
-10002	semanticsError	An error was detected in the input string; the syntax of the string is correct, but certain values specified in the string are inconsistent with other values specified in the string.
-10003	invalidValueError	The value of a numeric parameter is invalid.
-10004	valueConflictError	The value of a numeric parameter is inconsistent with another parameter, and the combination is therefore invalid.
-10005	badDeviceError	The device parameter is invalid.
-10006	badLineError	The line parameter is invalid.
-10007	badChanError	A channel is out of range for the board type or input configuration, the combination of channels is not allowed, or you must reverse the scan order so that channel 0 is last.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10008	badGroupError	The group is invalid.
-10009	badCounterError	The counter parameter is invalid.
-10010	badCountError	The count is too small or too large for the specified counter, or the given I/O transfer count is not appropriate for the current buffer or channel configuration.
-10011	badIntervalError	The analog input scan rate is too fast for the number of channels and the channel clock rate; or the given clock rate is not supported by the associated counter channel or I/O channel.
-10012	badRangeError	The analog input or analog output voltage range is invalid for the specified channel.
-10013	badErrorCodeError	The driver returned an unrecognized or unlisted error code.
-10014	groupTooLargeError	The group size is too large for the board.
-10015	badTimeLimitError	The time limit parameter is invalid.
-10016	badReadCountError	The read count parameter is invalid.
-10017	badReadModeError	The read mode parameter is invalid.
-10018	badReadOffsetError	The offset is unreachable.
-10019	badClkFrequencyError	The frequency parameter is invalid.
-10020	badTimebaseError	The timebase parameter is invalid.
-10021	badLimitsError	The limits are beyond the range of the board.
-10022	badWriteCountError	Your data array contains an incomplete update or you are trying to write past the end of the internal buffer or your output operation is continuous and the length of your array is not a multiple of one half of the internal buffer size.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10023	badWriteModeError	The write mode is out of range or is disallowed.
-10024	badWriteOffsetError	Adding the write offset to the write mark places the write mark outside the internal buffer.
-10025	limitsOutOfRangeError	The voltage limits are out of range for this board in the current configuration. Alternate limits were selected.
-10026	badInputBufferSpecification	The input buffer specification is invalid. This error results if, for example, you try to configure a multiple-buffer acquisition for a board that does not support multiple-buffer acquisition.
-10027	badDAQEventError	For DAQEvents 0 and 1 general value A must be greater than 0 and less than the internal buffer size. If DMA is used for DAQEvent 1, general value A must divide the internal buffer size evenly, with no remainder. If the TIO-10 is used for DAQEvent 4, general value A must be 1 or 2.
-10028	badFilterCutoffError	The cutoff frequency specified is not valid for this device.
-10029	obsoleteFunctionError	The function you are calling is no longer supported in this version of the driver.
-10030	badBaudRateError	The specified baud rate for communicating with the serial port is not valid on this platform.
-10031	badChassisIDError	The specified SCXI chassis does not correspond to a configured SCXI chassis.
-10032	badModuleSlotError	The SCXI module slot that was specified is invalid or corresponds to an empty slot.
-10033	invalidWinHandleError	The window handle passed to the function is invalid.
-10034	noSuchMessageError	No configured message matches the one you tried to delete.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10035	irrelevantAttributeError	The specified attribute is not relevant to this device and/or this scenario. Please consult the documentation for this function to determine the acceptable attributes for this device and/or scenario.
-10080	badGainError	The gain is invalid.
-10081	badPretrigCountError	The pretrigger sample count is invalid.
-10082	badPosttrigCountError	The posttrigger sample count is invalid.
-10083	badTrigModeError	The trigger mode is invalid.
-10084	badTrigCountError	The trigger count is invalid.
-10085	badTrigRangeError	The trigger range or trigger hysteresis window is invalid.
-10086	badExtRefError	The external reference value is invalid.
-10087	badTrigTypeError	The trigger type parameter is invalid.
-10088	badTrigLevelError	The trigger level parameter is invalid.
-10089	badTotalCountError	The total count specified is inconsistent with the buffer configuration and pretrigger scan count or with the board type.
-10090	badRPGError	The individual range, polarity, and gain settings are valid but the combination specified is not allowed for this board.
-10091	badIterationsError	You have attempted to use an invalid setting for the iterations parameter. The iterations value must be 0 or greater. Your device might be limited to only two values, 0 and 1.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10092	lowScanIntervalError	Some devices require a time gap between the last sample in a scan and the start of the next scan. The scan interval you have specified does not provide a large enough gap for the board. See your documentation for an explanation.
-10093	fifoModeError	FIFO mode waveform generation cannot be used because at least one condition is not satisfied.
-10100	badPortWidthError	The requested digital port width is not a multiple of the hardware port width or is not attainable by the DAQ hardware.
-10120	gpctrBadApplicationError	Invalid application used.
-10121	gpctrBadCtrNumberError	Invalid counterNumber used.
-10122	gpctrBadParamValueError	Invalid paramValue used.
-10123	gpctrBadParamIDError	Invalid paramID used.
-10124	gpctrBadEntityIDError	Invalid entityID used.
-10125	pctrBadActionError	Invalid action used.
-10200	EEPROMMreadError	Unable to read data from EEPROM.
-10201	EEPROMMwriteError	Unable to write data to EEPROM.
-10240	noDriverError	The driver interface could not locate or open the driver.
-10241	oldDriverError	The driver is out-of-date.
-10242	functionNotFoundError	The specified function is not located in the driver.
-10243	configFileError	The driver could not locate or open the configuration file, or the format of the configuration file is not compatible with the currently installed driver.
-10244	deviceInitError	The driver encountered a hardware-initialization error while attempting to configure the specified device.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10245	osInitError	The driver encountered an operating-system error while attempting to perform an operation, or the operating system does not support an operation performed by the driver.
-10246	communicationsError	The driver is unable to communicate with the specified external device.
-10247	cmosConfigError	The CMOS configuration-memory for the device is empty or invalid, or the configuration specified does not agree with the current configuration of the device, or the EISA system configuration is invalid.
-10248	dupAddressError	The base addresses for two or more devices are the same; consequently, the driver is unable to access the specified device.
-10249	intConfigError	The interrupt configuration is incorrect given the capabilities of the computer or device.
-10250	dupIntError	The interrupt levels for two or more devices are the same.
-10251	dmaConfigError	The DMA configuration is incorrect given the capabilities of the computer/DMA controller or device.
-10252	dupDMAError	The DMA channels for two or more devices are the same.
-10253	switchlessBoardError	NI-DAQ was unable to find one or more switchless boards you have configured using WDAQCONF.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10254	DAQCardConfigError	Cannot configure the DAQCard because: 1. The correct version of card and socket services software is not installed. 2. The card in the PCMCIA socket is not a DAQCard. 3. The base address and/or interrupt level requested are not available according to the card and socket services resource manager. Try different settings or use AutoAssign in the configuration utility.
-10255	remoteChassisDriverInit Error	There was an error in initializing the driver for Remote SCXI.
-10256	comPortOpenError	There was an error in opening the specified COM port.
-10257	baseAddressError	Bad base address specified in the configuration utility.
-10258	dmaChannel1Error	Bad DMA channel 1 specified in the configuration utility or by the operating system. 1. DMA channel 1 is the same as DMA channel 2 or 3. 2. The DMA channel assigned is not valid for this particular bus type (e.g. DMA channel 0, 1, 2, or 3 was assigned to an AT-MIO E Series device on an ISA bus computer).
-10259	dmaChannel2Error	Bad DMA channel 2 specified in the configuration utility or by the operating system. This error occurs when either: 1. DMA channel 2 is the same as DMA channel 1 or 3. 2. The DMA channel assigned is not valid for this particular bus type (e.g. DMA channel 0, 1, 2, or 3 was assigned to an AT-MIO E Series device on an ISA bus computer).

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10260	dmaChannel3Error	Bad DMA channel 3 specified in the configuration utility or by the operating system. This error occurs when either: 1. DMA channel 3 is the same as DMA channel 1 or 2. 2. The DMA channel assigned is not valid for this particular bus type (e.g. DMA channel 0, 1, 2, or 3 was assigned to an AT-MIO E Series device on an ISA bus computer).
-10261	userModeToKernelModeCall Error	The user mode code failed when calling the kernel mode code.
-10340	noConnectError	No RTSI signal/line is connected, or the specified signal and the specified line are not connected.
-10341	badConnectError	The RTSI signal/line cannot be connected as specified.
-10342	multConnectError	The specified RTSI signal is already being driven by a RTSI line, or the specified RTSI line is already being driven by a RTSI signal.
-10343	SCXIConfigError	The specified SCXI configuration parameters are invalid, or the function cannot be executed given the current SCXI configuration.
-10344	chassisSynchedError	The Remote SCXI unit is not synchronized with the host. Reset the chassis again to resynchronize it with the host.
-10345	chassisMemAllocError	The required amount of memory cannot be allocated on the Remote SCXI unit for the specified operation.
-10346	badPacketError	The packet received by the Remote SCXI unit is invalid. Check your serial port cable connections.
-10347	chassisCommunicationError	There was an error in sending a packet to the remote chassis. Check your serial port cable connections.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10348	waitingForReprogError	The Remote SCXI unit is in reprogramming mode and is waiting for reprogramming commands from the host (NI-DAQ Configuration Utility).
-10349	SCXIModuleTypeConflict Error	The module ID read from the SCXI module conflicts with the configured module type.
-10360	DSPInitError	The DSP driver was unable to load the kernel for its operating system.
-10370	badScanListError	The scan list is invalid. This error can result if, for example, you mix AMUX-64T channels and onboard channels, or if you scan multiplexed SCXI channels out of order.
-10400	userOwnedRsrcError	The specified resource is owned by the user and cannot be accessed or modified by the driver.
-10401	unknownDeviceError	The specified device is not a National Instruments product, or the driver does not support the device (for example, the driver was released before the device was supported).
-10402	deviceNotFoundError	No device is located in the specified slot or at the specified address.
-10403	deviceSupportError	The specified device does not support the requested action (the driver recognizes the device, but the action is inappropriate for the device).
-10404	noLineAvailError	No line is available.
-10405	noChanAvailError	No channel is available.
-10406	noGroupAvailError	No group is available.
-10407	lineBusyError	The specified line is in use.
-10408	chanBusyError	The specified channel is in use.
-10409	groupBusyError	The specified group is in use.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10410	relatedLCGBusyError	A related line, channel, or group is in use. If the driver configures the specified line, channel, or group, the configuration, data, or handshaking lines for the related line, channel, or group will be disturbed.
-10411	counterBusyError	The specified counter is in use.
-10412	noGroupAssignError	No group is assigned, or the specified line or channel cannot be assigned to a group.
-10413	groupAssignError	A group is already assigned, or the specified line or channel is already assigned to a group.
-10414	reservedPinError	Selected signal indicates a pin reserved by NIDAQ. You cannot configure this pin yourself.
-10415	externalMuxSupporError	This function does not support your DAQ device when an external multiplexer (such as an AMUX-64T or SCXI) is connected to it.
-10417	SCXIModuleNotSupported Error	At least one of the SCXI modules specified is not supported for the operation.
-10440	sysOwnedRsrcError	The specified resource is owned by the driver and cannot be accessed or modified by the user.
-10441	memConfigError	No memory is configured to support the current data transfer mode, or the configured memory does not support the current data transfer mode. (If block transfers are in use, the memory must be capable of performing block transfers.)
-10442	memDisabledError	The specified memory is disabled or is unavailable given the current addressing mode.
-10443	memAlignmentError	The transfer buffer is not aligned properly for the current data-transfer mode. For example, the buffer is at an odd address, is not aligned to a 32-bit boundary, is not aligned to a 512-bit boundary, and so on. Alternatively, the driver is unable to align the buffer because the buffer is too small.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10444	memFullError	No more system memory is available on the heap, or no more memory is available on the device, or insufficient disk space is available.
-10445	memLockError	The transfer buffer cannot be locked into physical memory. On PC AT machines, portions of the DMA data acquisition buffer may be in an invalid DMA region, for example, above 16 megabytes.
-10446	memPageError	The transfer buffer contains a page break; system resources may require reprogramming when the page break is encountered.
-10447	memPageLockError	The operating environment is unable to grant a page lock.
-10448	stackMemError	The driver is unable to continue parsing a string input due to stack limitations.
-10449	cacheMemError	A cache-related error occurred, or caching is not supported in the current mode.
-10450	physicalMemError	A hardware error occurred in physical memory, or no memory is located at the specified address.
-10451	virtualMemError	The driver is unable to make the transfer buffer contiguous in virtual memory and therefore cannot lock the buffer into physical memory; thus, you cannot use the buffer for DMA transfers.
-10452	noIntAvailError	No interrupt level is available for use.
-10453	intInUseError	The specified interrupt level is already in use by another device.
-10454	noDMACError	No DMA controller is available in the system.
-10455	noDMAAvailError	No DMA channel is available for use.
-10456	DMAInUseError	The specified DMA channel is already in use by another device.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10457	badDMAGroupError	DMA cannot be configured for the specified group because it is too small, too large, or misaligned. Consult the user manual for the device in question to determine group ramifications with respect to DMA.
-10458	diskFullError	The storage disk you specified is full.
-10459	DLLInterfaceError	The DLL could not be called due to an interface error.
-10480	muxMemFullError	The scan list is too large to fit into the mux-gain memory of the board.
-10481	bufferNotInterleavedError	You must provide a single buffer of interleaved data, and the channels must be in ascending order. You cannot use DMA to transfer data from two buffers; however, you may be able to use interrupts.
-10541	TRIG1ResourceConflict	CTRB1 will drive COUTB1, however CTRB1 will also drive TRIG1. This may cause unpredictable results when scanning the chassis.
-10560	invalidDSPHandleError	The DSP handle input is not valid.
-10561	DSPDataPathBusyError	Either DAQ or WFM can use a PC memory buffer, but not both at the same time.
-10600	noSetupError	No setup operation has been performed for the specified resources. Or, some resources require a specific ordering of calls for proper setup.
-10601	multSetupError	The specified resources have already been configured by a setup operation.
-10602	noWriteError	No output data has been written into the transfer buffer.
-10603	groupWriteError	The output data associated with a group must be for a single channel or must be for consecutive channels.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10604	activeWriteError	Once data generation has started, only the transfer buffers originally written to can be updated. If DMA is active and a single transfer buffer contains interleaved channel-data, new data must be provided for all output channels currently using the DMA channel.
-10605	endWriteError	No data was written to the transfer buffer because the final data block has already been loaded.
-10606	notArmedError	The specified resource is not armed.
-10607	armedError	The specified resource is already armed.
-10608	noTransferInProgError	No transfer is in progress for the specified resource.
-10609	transferInProgError	A transfer is already in progress for the specified resource.
-10610	transferPauseError	A single output channel in a group cannot be paused if the output data for the group is interleaved.
-10611	badDirOnSomeLinesError	Some of the lines in the specified channel are not configured for the transfer direction specified. For a write transfer, some lines were configured for input. For a read transfer, some lines were configured for output.
-10612	badLineDirError	The specified line does not support the specified transfer direction.
-10613	badChanDirError	The specified channel does not support the specified transfer direction.
-10614	badGroupDirError	The specified group does not support the specified transfer direction.
-10615	masterClkError	The clock configuration for the clock master is invalid.
-10616	slaveClkError	The clock configuration for the clock slave is invalid.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10617	noClkSrcError	No source signal has been assigned to the clock resource.
-10618	badClkSrcError	The specified source signal cannot be assigned to the clock resource.
-10619	multClkSrcError	A source signal has already been assigned to the clock resource.
-10620	noTrigError	No trigger signal has been assigned to the trigger resource.
-10621	badTrigError	The specified trigger signal cannot be assigned to the trigger resource.
-10622	preTrigError	The pretrigger mode is not supported or is not available in the current configuration, or no pretrigger source has been assigned.
-10623	postTrigError	No posttrigger source has been assigned.
-10624	delayTrigError	The delayed trigger mode is not supported or is not available in the current configuration, or no delay source has been assigned.
-10625	masterTrigError	The trigger configuration for the trigger master is invalid.
-10626	slaveTrigError	The trigger configuration for the trigger slave is invalid.
-10627	noTrigDrvError	No signal has been assigned to the trigger resource.
-10628	multTrigDrvError	A signal has already been assigned to the trigger resource.
-10629	invalidOpModeError	The specified operating mode is invalid, or the resources have not been configured for the specified operating mode.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10630	invalidReadError	The parameters specified to read data were invalid in the context of the acquisition. For example, an attempt was made to read 0 bytes from the transfer buffer, or an attempt was made to read past the end of the transfer buffer.
-10631	noInfiniteModeError	Continuous input or output transfers are not allowed in the current operating mode, or continuous operation is not allowed for this type of device.
-10632	someInputsIgnoredError	Certain inputs were ignored because they are not relevant in the current operating mode.
-10633	invalidRegenModeError	This board does not support the specified analog output regeneration mode.
-10634	noContTransferInProgress Error	No continuous (double buffered) transfer is in progress for the specified resource.
-10635	invalidSCXIOpModeError	Either the SCXI operating mode specified in a configuration call is invalid, or a module is in the wrong operating mode to execute the function call.
-10636	noContWithSynchError	You cannot start a continuous (double-buffered) operation with a synchronous function call.
-10637	bufferAlreadyConfigError	Attempted to configure a buffer after the buffer had already been configured. You can configure a buffer only once.
-10680	badChanGainError	All channels of this board must have the same gain.
-10681	badChanRangeError	All channels of this board must have the same range.
-10682	badChanPolarityError	All channels of this board must have the same polarity.
-10683	badChanCouplingError	All channels of this board must have the same coupling.
-10684	badChanInputModeError	All channels of this board must have the same input range.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10685	clkExceedsBrdsMaxConv Rate	The clock rate selected exceeds the recommended maximum rate for this board.
-10686	scanListInvalidError	A configuration change has invalidated the acquisition buffer, or an acquisition buffer has not been configured.
-10687	bufferInvalidError	A configuration change has invalidated the allocated buffer.
-10688	noTrigEnabledError	The total number of scans and pretrigger scans implies that a trigger start is intended, but no trigger is enabled.
-10689	digitalTrigBError	Digital trigger B is illegal for the number of total scans and pretrigger scans specified.
-10690	digitalTrigAandBError	This board does not allow digital triggers A and B to be enabled at the same time.
-10691	extConvRestrictionError	This board does not allow an external sample clock with an external scan clock, start trigger, or stop trigger.
-10692	hanClockDisabledError	The acquisition cannot be started because the channel clock is disabled.
-10693	extScanClockError	You cannot use an external scan clock when doing a single scan of a single channel.
-10694	unsafeSamplingFreqError	The sample frequency exceeds the safe maximum rate for the hardware, gains, and filters used.
-10695	DMANotAllowedError	You have set up an operation that requires the use of interrupts. DMA is not allowed. For example, some DAQ events, such as messaging and LabVIEW occurrences, require interrupts.
-10696	multiRateModeError	Multirate scanning can not be used with AMUX-64, SCXI, or pre-triggered acquisitions.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10697	rateNotSupportedError	NIDAQ was unable to convert your timebase/interval pair to match the actual hardware capabilities of the specified board.
-10698	timebaseConflictError	You cannot use this combination of scan and sample clock timebases for the specified board.
-10699	polarityConflictError	You cannot use this combination of scan and sample clock source polarities for this operation, for the specified board.
-10700	signalConflictError	You cannot use this combination of scan and convert clock signal sources for this operation, for the specified board.
-10701	noLaterUpdateError	The call had no effect because the specified channel had not been set for later internal update.
-10702	prePostTriggerError	Pretriggering and posttriggering cannot be used simultaneously on the Lab and 1200 series devices.
-10710	noHandshakeModeError	The specified port has not been configured for handshaking.
-10720	noEventCtrError	The specified counter is not configured for event-counting operation.
-10740	SCXITrackHoldError	A signal has already been assigned to the SCXI track-and-hold trigger line, or a control call was inappropriate because the specified module is not configured for one-channel operation.
-10780	sc2040InputModeError	When you have an SC2040 attached to your device, all analog input channels must be configured for differential input mode.
-10781	outputTypeMustBeVoltage Error	The polarity of the output channel cannot be bipolar when outputting currents.
-10782	sc2040HoldModeError	The specified operation cannot be performed with the SC-2040 configured in hold mode.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10783	calConstPolarityConflict Error	Calibration constants in the load area have a different polarity from the current configuration. Therefore, you should load constants from factory.
-10800	timeOutError	The operation could not complete within the time limit.
-10801	calibrationError	An error occurred during the calibration process.
-10802	dataNotAvailError	The requested amount of data has not yet been acquired, or the acquisition has completed and no more data is available to read.
-10803	transferStoppedError	The transfer has been stopped to prevent regeneration of output data.
-10804	earlyStopError	The transfer stopped prior to reaching the end of the transfer buffer.
-10805	overRunError	The clock source for the input transfer is faster than the maximum input-clock rate; the integrity of the data has been compromised. Alternatively, the clock source for the output transfer is faster than the maximum output-clock rate; a data point was generated more than once since the update occurred before new data was available.
-10806	noTrigFoundError	No trigger value was found in the input transfer buffer.
-10807	earlyTrigError	The trigger occurred before sufficient pretrigger data was acquired.
-10808	LPTCommunicationError	An error occurred in the parallel port communication with the SCXI-1200.
-10809	gateSignalError	Attempted to start a pulse width measurement with the pulse in the active state.
-10810	internalDriverError	An unexpected error occurred inside the driver when performing an operation.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10811	internalKernelError	An unexpected error occurred inside the kernel of the device while performing this operation.
-10840	softwareError	The contents or the location of the driver file was changed between accesses to the driver.
-10841	firmwareError	The firmware does not support the specified operation, or the firmware operation could not complete due to a data-integrity problem.
-10842	hardwareError	The hardware is not responding to the specified operation, or the response from the hardware is not consistent with the functionality of the hardware.
-10843	underFlowError	The update rate exceeds your system's capacity to supply data to the output channel.
-10844	underWriteError	At the time of the update for the device-resident memory, insufficient data was present in the output transfer buffer to complete the update.
-10845	overFlowError	At the time of the update clock for the input channel, the device-resident memory was unable to accept additional data—one or more data points may have been lost.
-10846	overWriteError	New data was written into the input transfer buffer before the old data was retrieved.
-10847	dmaChainingError	New buffer information was not available at the time of the DMA chaining interrupt; DMA transfers will terminate at the end of the currently active transfer buffer.
-10848	noDMACountAvailError	The driver could not obtain a valid reading from the transfer-count register in the DMA controller.
-10849	OpenFileError	The configuration file or DSP kernel file could not be opened.
-10850	closeFileError	Unable to close a file.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10851	fileSeekError	Unable to seek within a file.
-10852	readFileError	Unable to read from a file.
-10853	writeFileError	Unable to write to a file.
-10854	miscFileError	An error occurred accessing a file.
-10855	osUnsupportedError	NI-DAQ does not support the current operation on this particular version of the operating system.
-10856	osError	An unexpected error occurred from the operating system while performing an operation.
-10880	updateRateChangeError	A change to the update rate is not possible at this time because: 1. When waveform generation is in progress, you cannot change the interval timebase. 2. When you make several changes in a row, you must give each change enough time to take effect before requesting further changes.
-10881	partialTransferComplete Error	You cannot do another transfer after a successful partial transfer.
-10882	daqPollDataLossError	The data collected on the Remote SCXI unit was overwritten before it could be transferred to the buffer in the host. Try using a slower data acquisition rate if possible.
-10883	wfmPollDataLossError	New data could not be transferred to the waveform buffer of the Remote SCXI unit to keep up with the waveform update rate. Try using a slower waveform update rate if possible.
-10884	pretrigReorderError	Could not rearrange data after a pretrigger acquisition completed.

Table B-1. Data Acquisition Control Error Codes (Continued)

Code	Name	Description
-10920	gpctrDataLossError	One or more data points may have been lost during buffered GPCTR operations due to the speed limitations of your system.
-10940	chassisResponseTimeout Error	No response was received from the Remote SCXI unit within the specified time limit.
-10941	reprogrammingFailedError	Reprogramming the Remote SCXI unit was unsuccessful. Please try again.
-10942	invalidResetSignatureError	An invalid reset signature was sent from the host to the Remote SCXI unit.
-10943	chassisLockupError	The interrupt service routine on the remote SCXI unit is taking a longer than necessary amount of time to complete, which can possibly affect the integrity of the remote SCXI operations. You do not need to reset your remote SCXI unit at this point; however, please clear and restart your data acquisition.

If an error condition occurs during a call to any of the functions in the ComponentWorks Analysis Controls, the exception contains the error code. This code is a value that specifies the type of error that occurred. You can find the currently defined error codes and their associated meanings in Table B-2.

Table B-2. Analysis Error Codes

Code	Name	Description
0	NoErr	No error, the call is successful.
-20001	OutOfMemErr	There is not enough space left to perform the specified routine.
-20002	EqSamplesErr	The input sequences must be the same size.
-20003	SamplesGTZeroErr	The number of samples must be greater than zero.
-20004	SamplesGEZeroErr	The number of samples must be greater than or equal to zero.
-20006	SamplesGETwoErr	The number of samples must be greater than or equal to two.
-20007	SamplesGEThreeErr	The number of samples must be greater than or equal to three.
-20008	ArraySizeErr	The input arrays do not contain the correct number of data values for this function.
-20009	PowerOfTwoErr	The size of the input array must be a valid power of two: $size = 2^m$, $0 < m < 23$.
-20010	MaxXformSizeErr	The maximum allowable transform size has been exceeded.
-20011	DutyCycleErr	The duty cycle must be equal to or fall between 0 and 100: $0 \leq duty\ cycle \leq 100$.
-20012	CyclesErr	The number of cycles must be greater than zero and less than or equal to the number of samples.
-20013	WidthLTSamplesErr	The width must meet: $0 < width < samples$.

Table B-2. Analysis Error Codes (Continued)

Code	Name	Description
-20014	DelayWidthErr	The following condition must be met: $0 \le (delay + width) < samples$.
-20015	DtGEZeroErr	dt must be ≥ 0.
-20016	DtGTZeroErr	dt must be greater than zero.
-20017	IndexLTSamplesErr	The index must meet: $0 \le index \le samples$.
-20018	IndexLengthErr	The following condition must be met: $0 \le (index + length) < samples$.
-20019	UpperGELowerErr	The upper value must be greater than or equal to the lower value.
-20020	NyquistErr	The cut-off frequency, *fc*, must meet: $0 \le fc \le fs/2$.
-20021	OrderGTZeroErr	The order must be greater than zero.
-20022	DecFactErr	The decimating factor must meet: $0 < decimating\ factor \le samples$.
-20023	BandSpecErr	The following conditions must be met: $0 \le f_low \le f_high \le fs/2$.
-20024	RippleGTZeroErr	The ripple amplitude must be greater than zero.
-20025	AttenGTZeroErr	The attenuation must be greater than zero.
-20026	WidthGTZeroErr	The width must be > 0.
-20027	FinalGTZeroErr	The final value must be > 0.
-20028	AttenGTRippleErr	The attenuation must be greater than the ripple amplitude.
-20029	StepSizeErr	The step-size, *u*, must meet: $0 \le u \le 0.1$.
-20030	LeakErr	The leakage coefficient, leak, and step-size parameter, *u*, must meet: $0 \le leak \le u$.

Table B-2. Analysis Error Codes (Continued)

Code	Name	Description
-20031	EqRplDesignErr	The filter cannot be designed with the specified input parameters.
-20032	RankErr	The rank of the filter must meet: $1 \leq (2*rank + 1) \leq size$.
-20033	EvenSizeErr	The number of coefficients must be odd for this filter.
-20034	OddSizeErr	The number of coefficients must be even for this filter.
-20035	StdDevErr	The standard deviation must be greater than zero for normalization.
-20036	MixedSignErr	The second (Y) array input must be nonzero and either all positive or all negative.
-20037	SizeGTOrderErr	The number of data points in the Y Values array must be greater than the order.
-20038	IntervalsErr	The number of intervals must be > 0.
-20039	MatrixMulErr	The number of columns in the first matrix is not equal to the number of rows in the second matrix or vector.
-20040	SquareMatrixErr	The input matrix must be a square matrix.
-20041	SingularMatrixErr	The system of equations cannot be solved because the input matrix is singular.
-20042	LevelsErr	The numbers of levels is outside the range.
-20043	FactorErr	The level of factor is outside the range for some data.
-20044	ObservationsErr	Zero Observations were made at some level of a factor.
-20045	DataErr	The total number of data points must be equal to product *levels/each factor * observations/cell*.
-20046	OverflowErr	There is an overflow in the calculated F-value for the ANOVA Fit function.

Table B-2. Analysis Error Codes (Continued)

Code	Name	Description
-20047	BalanceErr	The data is unbalanced. All cells must contain the same number of observations.
-20048	ModelErr	The Random Effect model was requested when the Fixed effect model was required.
-20049	DistinctErr	The x-values must be distinct.
-20050	PoleErr	The interpolating function has a pole at the requested value.
-20051	ColumnErr	The first column in the X matrix must be all ones.
-20052	FreedomErr	The degrees of freedom must be one or more.
-20053	ProbabilityErr	The probability must meet the condition: $0 < p < 1$.
-20054	InvProbErr	The probability must meet the condition $0 \leq p < 1$.
-20055	CategoryErr	The number of categories or samples must be greater than one.
-20056	TableErr	The contingency table has a negative number.
-20057	BetaFuncErr	The parameter to the beta function should be $0 < p < 1$.
-20058	DimensionErr	Invalid number of dimensions or dependent variables.
-20059	NegNumErr	Negative number error.
-20060	DivByZeroErr	Divide by zero err.
-20061	InvSelectionErr	Invalid selection: useful in programs that perform a task based on a user selection.
-20062	MaxIterErr	Maximum iteration was exceeded.
-20063	PolyErr	The coefficients of the polynomial are invalid.
-20064	InitStateErr	The internal memory state of this function was not initialized correctly.

Table B-2. Analysis Error Codes (Continued)

Code	Name	Description
-20065	ZeroVectorErr	The elements in the vector cannot all be zero.
-20066	IIRFilterInfoErr	The information in IIR filter structure is not correct.
-20080	AdaptTIErr	Time increment must be greater than the window length divided by 16.
-20081	GaborDNErr	dN must be >0.
-20082	GaborTIErr	Time increment must be greater than dN.
-20083	JTFAWindowLErr	Window length must be > 4 and a power of 2.
-20084	JTFATIErr	Time increment must not be greater than the window length divided by 4.
-20085	JTFAHilbertErr	The size of the input array and its Hilbert transform must be equal.
-20086	STFTWindowLErr	Window length must be > 2 and a power of 2.
-20101	BaseGETopErr	The top value must be greater than the base value.
-20102	ShiftRangeErr	The shifts must meet: $\|shifts\| < samples$.
-20103	OrderGEZeroErr	The order must be positive.
-20999	FatalDSPErr	Serious algorithm failure. Call National Instruments support.
-20300	EngineeringMathErr	The starting error codes of engineering math.

Table B-3 lists the negative error codes that may be returned by any ComponentWorks operation.

Table B-3. General ComponentWorks Error Codes

Code	Name	Description
-30000	kErrUnexpected	An unexpected error has occurred.
-30001	kErrTooManyGroups	Too many controls are configured for this DAQ device. Reset one of the other controls configured for this device before configuring any more controls.
-30002	kErrInvalidParameterValue	You have passed an invalid value for one of the parameters to the function, method or property.
-30003	kErrInvalidParameterType	You have passed an invalid type into a parameter of a VARIANT type.
-30004	kErrDivideByZero	A divide by zero error has occurred.
-30005	kErrImaginaryNumber	The result of a calculation is an imaginary number.
-30006	kErrOverflow	An overflow error has occurred.
-30007	kErrOutOfMemory	Out of memory.
-30008	KErrFunctionNotLicensed	You have called a function or method that requires a license that you do not have.
-30009	kErrCantReconfigure	The properties of the currently configured task are too different from the one you are trying to reconfigure to. For example, the device may be different.
-30010	kErrIllegalMethodFor CurrentState	The method called cannot be used on the currently configured task.
-30013	kErrNeedNewerNIDAQ	Component, or the current settings require a more recent version of NI-DAQ than is current installed.
-30014	kErrCannotReturnBothData Types	Scaled and unscaled data cannot be returned simultaneously.
-30015	kErrInvalidProgressInterval	`CWAI.ProgressInterval must divide evenly into CWAI.NScans.`

Table B-3. General ComponentWorks Error Codes

-30016	kErrInvalidNScansPer Buffer	`CWAI.NScans must divide evenly into CWAI.NScansPerBuffer.`
-30017	kErrDIOInvalidProgress Interval	Progress Interval must divide evenly into NPatterns.
-30100	KErrElementTypeErr	The type of the element is wrong.
-30101	KErrCouldNotLock	Cannot lock the array.
-30102	KErrCouldNotUnlock	Cannot unlock the array.
-30103	KErrBadArray	The array is not a safe array.
-30104	kErrParameterTypes	The input parameters must all contain the same data type.

Customer Communication

Appendix C

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 U.S. 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.

	Telephone	Fax
Australia	03 9879 5166	02 9874 4455
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 99
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 5734815	03 5734816
Italy	02 413091	06 57284309
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 States	512 794 0100	512 794 8411
U.K.	01635 523545	01635 523154

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 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

DAQ hardware _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

ComponentWorks and NI-DAQ version _____

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*

Edition Date: June 1997

Part Number: 321170B-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 _____

_____

Phone (___)_____ Fax (___) _____

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

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Glossary

Prefix	Meaning	Value
m-	milli-	10^{-3}
µ-	micro-	10^{-6}
n-	nano-	10^{-9}

Numbers/Symbols

12-bit — Resolution of a data acquisition device. A 12-bit device converts an analog voltage into a 12-bit binary integer. The binary value is scaled to a voltage representation in software.

16-bit — Resolution of a data acquisition device. A 16-bit device converts an analog voltage into a 16-bit binary integer. The binary value is scaled to a voltage representation in software.

1D — One-dimensional.

2D — Two-dimensional.

9513 Counter — *See* AM9513 Counter.

A

A — Amperes.

A/D — Analog-to-digital.

AC — Alternating current.

AC signal — A signal with significant frequency components.

ActiveX	A synonym of OLE, referring to a set of Microsoft technologies for reusable software components.
ActiveX Control	*See* OLE control.
ADC	Analog-to-digital converter. An electronic device, often an integrated circuit, that converts an analog voltage to a digital number.
ADC resolution	The resolution of the ADC, which is measured in bits. An ADC with 16 bits has a higher resolution than a 12-bit ADC.
AI	Analog input.
AIGND	The analog input ground pin on a DAQ device.
AIPoint	Analog Input Single Point control.
AM 9513 Counter	Counter/Timer chip used on legacy MIO and other data acquisition devices, replaced by the DAQ-STC chip on newer devices, including the E-series DAQ boards. The AM9513 Counter does not support some of the advanced features of the DAQ-STC.
amplification	A type of signal conditioning that improves accuracy in the resulting digitized signal and reduces noise.
AMUX devices	*See* analog multiplexer.
analog multiplexer	Devices that increase the number of measurement channels while still using a single instrumentation amplifier. Also called AMUX devices.
analog trigger	A trigger that occurs at a user-selected point on an incoming analog signal. Triggering can be set to occur at a specified level on either an increasing or a decreasing signal (positive or negative slope). Analog triggering can be implemented either in software or in hardware. When implemented in software, all data is collected, transferred into system memory, and analyzed for the trigger condition. When analog triggering is implemented in hardware, no data is transferred to system memory until the trigger condition has occurred.
ANOVA	Analysis of variance.
AO	Analog output.

AOPoint	Analog output single point control.
array	Ordered, indexed set of data elements of the same type.
asynchronous	Property of a function or operation that begins an operation and returns control to the program prior to the completion or termination of the operation.
ATC	Analog Trigger Circuit; an analog trigger contained on some E series DAQ devices, required for any analog triggering (including ETS operations).

B

BCD	Binary-coded decimal.
bipolar	A signal range that includes both positive and negative values (for example, -5 to 5 V).
buffer	Temporary storage for acquired or generated data.

C

callback (function)	A user-defined function that is 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.
channel	Pin or wire lead to which you apply or from which you read the analog or digital signal. Analog signals can be single-ended or differential. For digital signals, you group channels to form ports. Ports usually consist of either four or eight digital channels.
channel clock	The clock controlling the time interval between individual channel sampling within a scan. Boards with simultaneous sampling do not have this clock.
channel list	A collection of channel objects that specify the channels used by a control.
Chart History	A CWGraph property that determines how many points the graph stores when charting before deleting old data.

Chart Style	A CWGraph property that specifies how a chart method updates the display as new data is plotted.
clock	Hardware component that controls timing for reading from or writing to groups; an input pin on a counter/timer.
cm	centimeters.
code width	The smallest detectable change in an input voltage of a DAQ device.
Collection	A collection is a 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	A way to organize the data in a 2D array by columns.
common-mode voltage	Any voltage present at the instrumentation amplifier inputs with respect to amplifier ground.
condition object	An object used to specify the Start, Pause, or Stop condition on a data acquisition process.
Control Refresh	A CWGraph property that determines when changes to the graph are displayed. The property name is `ImmediateUpdates`.
conversion device	Device that transforms a signal from one form to another. For example, analog-to-digital converters (ADCs) for analog input, digital-to-analog converters (DACs) for analog output, digital input or output ports, and counter/timers are conversion devices.
counter/timer group	A collection of counter/timer channels. You can use this type of group for simultaneous operation of multiple counter/timers.
coupling	The manner in which a signal is connected from one location to another.

D

D/A	Digital-to-analog.
DAC	Digital-to-analog converter. An electronic device, often an integrated circuit, that converts a digital number into a corresponding analog voltage or current.
DAQ	Data acquisition.
DAQ-STC	DAQ System Timing Controller. An ASIC developed by National Instruments for enhanced timing control on data acquisition devices. DAQ-STC is used on E-Series and other National Instruments DAQ devices, and is required for certain counter/timer operations, including buffered counter measurements and frequency shift keying.
data acquisition	Process of acquiring data, typically from A/D or digital input plug-in boards.
DC	Direct current.
DC signal	A signal made up solely of a non-dynamic component, steady or very slowly changing voltage.
Delphi	Borland Delphi programming environment.
device	A plug-in data acquisition board that can contain multiple channels and conversion devices.
device number	The slot number or board ID number assigned to the board when you configured it.
DFT	Discrete Fourier Transform.
DI	Digital Input.
DIFF	Differential. A differential input is an analog input consisting of two terminals, both of which are isolated from computer ground and whose difference you measure.
differential measurement	A way you can configure your device to read signals, in which you do not need to connect either input to a fixed reference, such as the earth or a building ground.

digital trigger	A TTL level signal having two discrete levels—a high and a low level used to trigger (start or stop) another process.
DIO	Digital Input/Output.
dithering	The addition of Gaussian noise to an analog input signal.
DLL	Dynamic link library.
DMA	Direct memory access. A method by which data you can transfer data to computer memory from a device or memory on the bus (or from computer memory to a device) while the processor does something else. DMA is the fastest method of transferring data to or from computer memory.
DO	Digital Output.
down counter	Performing frequency division on an internal signal.
driver	Software that controls a specific hardware device, such as a data acquisition board.
DSP	Digital signal processing.

E

E-Series Device	Series of enhanced data acquisition devices that include technologies such as DAQ-STC ASIC, Plug and Play compatibility, and the NI-PGIA. Some functionality in the ComponentWorks DAQ controls can only be used by E-Series devices.
EEPROM	Electrically erased programmable read-only memory. Read-only memory that you can erase with an electrical signal and reprogram.
EISA	Extended Industry Standard Architecture.
ETS	*See* Equivalent Time Sampling.
Equivalent Time Sampling	An analog input data acquisition method in which point are acquired with increasing delay from a fixed point on a repetitive waveform.

event — An object generates an event in 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* event.

exception — An error message generated by a control, sent directly to the application or programming environment containing the control.

external trigger — A voltage pulse from an external source that triggers an event such as A/D conversion.

F

FFT — Fast Fourier Transform.

FHT — Fast Hartley Transform.

FIFO — A first-in-first-out memory buffer. In a FIFO, the first data stored is the first data sent to the acceptor.

filtering — A type of signal conditioning that allows you to filter unwanted signals from the signal you are trying to measure.

FIR — Finite impulse response.

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.

floating signal sources — Signal sources with voltage signals that are not connected to an absolute reference or system ground. Also called nonreferenced signal sources. Some common example of floating signal sources are batteries, transformers, or thermocouples.

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

Format — A flexible specification that defines how a number is displayed on an axis 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.
FSR	Frequency Shift Keying. An advanced pulse generation mode in which the output frequency is switched by another digital signal.
function tree	The hierarchical structure in which the functions in a library or an instrument driver are grouped. The function tree simplifies access to a library or instrument driver by presenting functions organized according to the operation they perform, as opposed to a single linear listing of all available functions.

G

gain	The factor by which a signal is amplified, sometimes expressed in decibels.
GATE input pin	A counter input pin that controls when counting in your application occurs.
grounded measurement system	*See* referenced single-ended measurement system.
grounded signal sources	Signal sources with voltage signals that are referenced to a system ground, such as the earth or a building ground. Also called referenced signal sources.
GUI	Graphical user interface.

H

handshaked digital I/O	A type of digital acquisition/generation where a device or module accepts or transfers data after a digital pulse has been received. Also called latched digital I/O.
hardware triggering	A form of triggering where you set the start time of an acquisition and gather data at a known position in time relative to a trigger signal.
hex	hexadecimal.
Hz	Hertz. The number of scans read or updates written per second.

I

I/O	Input/output. The transfer of data to or from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces.
I/O Connector	The connector on a data acquisition device, used to connect the device to external signals or devices.
ICtr(82C53)	Simple counter/timer chip used on 1200-, 700-, 500-, Lab-, and LPM- series data acquisition devices. ICtr(82C53) supports limited counter pulse capabilities and can be controlled using the ICtr functions in the ComponentWorks DAQTools control.
IDFT	inverse Discrete Fourier Transform.
IEEE	Institute of Electrical and Electronic Engineers.
IFFT	inverse Fast Fourier Transform.
IFHT	inverse Fast Hartley Transform.
IIR	infinite impulse response.
immediate digital I/O	A type of digital acquisition/generation where LabVIEW updates the digital lines or port states immediately or returns the digital value of an input line. Also called nonlatched digital I/O.
in.	inches.
input limits	The upper and lower voltage inputs for a channel. You must use a pair of numbers to express the input limits.
instrument driver	A library of functions to control and use one specific physical instrument. Also a set of functions that adds specific functionality to an application.
interrupt	A signal indicating that the central processing unit should suspend its current task to service a designated activity.
interval clock	Clock used in a DAQ device in an analog input operation to control the delay between samples acquired from consecutive channels in a scan.

interval delay — The delay between samples acquired from consecutive channels in a scan during an analog input operation.

interval scanning — Scanning method where there is a longer interval between scans than there is between individual channels comprising a scan.

ISA — Industry Standard Architecture.

isolation — A type of signal conditioning in which you isolate the transducer signals from the computer for safety purposes. This protects you and your computer from large voltage spikes and makes sure the measurements from the DAQ device are not affected by differences in ground potentials.

K

ksamples — 1,000 samples.

Kwords — 1,024 words of memory.

L

latched digital I/O — A type of digital acquisition/generation where a device or module accepts or transfers data after a digital pulse has been received. Also called handshaked digital I/O.

LED — Light-emitting diode.

limit settings — The maximum and minimum voltages of the analog signals you are measuring or generating.

linearization — A type of signal conditioning that linearizes the voltage levels from transducers, so the voltages can be scaled to measure physical phenomena.

LSB — Least significant bit.

M

MB — megabytes of memory.

memory buffer — *See* buffer.

method	A function that performs a specific action on or with an object. The operation of the method often depends on the values of the object's properties.
mse	Mean squared error.
multibuffered I/O	Input operation for which you allocate more than one memory buffer so you can read and process data from one buffer while the acquisition fills another.
multiplexed mode	An SCXI operating mode in which analog input channels are multiplexed into one module output so that your cabled DAQ device has access to the module's multiplexed output as well as the outputs on all other multiplexed modules in the chassis through the SCXI bus. Also called serial mode.
multiplexer	A set of semiconductor or electromechanical switches with a common output that can select one of a number of input signals and that you commonly use to increase the number of signals measured by one ADC.

N

NI-DAQ	The driver level software that controls National Instruments data acquisition cards and devices.
non-referenced signal sources	Signal sources with voltage signals that are not connected to an absolute reference or system ground. Also called floating signal sources. Some common example of non-referenced signal sources are batteries, transformers, or thermocouples.
non-referenced single-ended	All measurements are made with respect to a common reference, but the voltage at this reference can vary with respect to the measurement system ground.
nonlatched digital I/O	A type of digital acquisition/generation where the DIO control updates the digital lines or port states immediately or returns the digital value of an input line. Also called immediate digital I/O.
NRSE	Nonreferenced single-ended.

O

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

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

OCX — OLE Control eXtension. Another name for OLE or ActiveX controls, reflected by the .OCX file extension of ActiveX control files.

OLE — Object linking and embedding.

OLE control — A standard software tool that adds additional functionality to any compatible OLE container. The DAQ, GUI, and analysis tools in ComponentWorks are all OLE controls. An OLE control defines properties, methods, objects, and events.

onboard channels — Channels provided by the plug-in data acquisition board.

OUT output pin — A counter output pin where the counter can generate various TTL pulse waveforms.

output limits — The upper and lower voltage or current outputs for an analog output channel. The output limits determine the polarity and voltage reference settings for a board.

P

parallel mode — A type of SCXI operating mode in which the module sends each of its input channels directly to a separate input channel of the device to the module.

pattern — One update (input or output) on a digital port. The number of updates on a buffered digital operation is measured in number of patterns.

pattern generation	A type of handshaked (latched) digital I/O in which internal counters generate the handshaked signal, which in turn initiates a digital transfer. Because counters output digital pulses at a constant rate, this means you can generate and retrieve patterns at a constant rate because the handshaked signal is produced at a constant rate.
pause condition	A condition on a data acquisition process that determines when the acquisition is paused. The condition can be a digital hardware signal or the state of an analog hardware signal relative to set limits.
PFI	Programmable Function Input. Input and output lines on the I/O controller of E-series data acquisition devices.
PGIA	Programmable gain instrumentation amplifier.
Plot	A CWGraph group of methods that displays a new set of data while deleting any previous data on the graph. A plot also refers to one of the traces (data lines) on a graph representing the data in one row or column of an array. Each plot on the graph has its own properties, such as color, style, and so on.
Plug and Play devices	Devices that do not require dip switches or jumpers to configure resources on the devices. Also called switchless devices.
PnP	*See* Plug and Play devices.
Pointer	An indicator on a CWSlide or CWKnob object. You can use a collection of pointers to display different values on the same object. In the collection, each pointer is referenced by an index in the collection and each individual pointer has its own properties such as color, style, mode, and so on.
postriggering	The technique you use on a data acquisition board to acquire a programmed number of samples after trigger conditions are met.
pretriggering	The technique you use on a data acquisition board to keep a continuous buffer filled with data, so that when the trigger conditions are met, the sample includes the data leading up to the trigger condition.
property	An attribute that controls the appearance or behavior of an object. The property can be a specific value or another object with its own properties and methods. For example, a value property is the color (property) of a plot (object), while an object property is a specific

Y axis (property) on a graph (object). The Y axis itself is another object with properties, such as minimum and maximum values.

pulse — Physical signal generated by the Pulse control. Pulse uses TTL levels on most data acquisition devices.

pulse delay — The amount of time from the start of a pulse generation until the active phase of the pulse, measured in seconds. In a positive polarity pulse, the output from the counter will be low for the duration of the pulse delay before going high.

pulse period — The period of a continuous or finite pulse train. The period is defined as the amount of time between two consecutive rising or falling edges of the signal. The period is the inverse of the frequency of the pulse train.

pulse trains — Multiple pulses.

pulse width — The width of a pulse generated by a counter on a data acquisition device, measured in seconds.

pulsed output — A form of counter signal generation by which a pulse is output when a counter reaches a certain value.

R

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

referenced single-ended (RSE) — All measurements are made with respect to a common reference or a ground. Also called a grounded measurement system.

RMS — Root mean square.

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

RSE — Referenced single-ended.

RTD — Resistance temperature detector. A temperature-sensing device whose resistance increases with increases in temperature.

RTSI — Real-Time System Integration bus. The National Instruments timing bus that interconnects data acquisition boards directly, by means of connectors on top of the boards, for precise synchronization of functions.

S

sample — A single (one and only one) analog or digital input or output data point.

sample counter — The clock that counts the output of the channel clock, in other words, the number of samples taken. On boards with simultaneous sampling, this counter counts the output of the scan clock and hence the number of scans.

scan — One or more analog or digital input samples. Typically, the number of input samples in a scan is equal to the number of channels n the input group. For example, one pulse from the scan clock produces one scan which acquires one new sample from every analog input channel in the group.

scan clock — The clock controlling the time interval between scans. On boards with interval scanning support (for example, the AT-MIO-16F-5), this clock gates the channel clock on and off. On boards with simultaneous sampling, this clock clocks the track-and-hold circuitry.

scan rate — The number of scans per second. For example, at a scan rate of 10Hz, each channel is sampled 10 times per second.

scan width — The number of channels in the channel list or number of ports in the port list you use to configure an analog or digital input group.

SCXI — Signal Conditioning eXtensions for Instrumentation. The National Instruments product line for conditioning low-level signals within an external chassis near sensors, so only high-level signals in a noisy environment are sent to data acquisition boards.

sec — Seconds.

settling time — The amount of time required for a voltage to reach its final value within specified limits.

signal conditioning — The manipulation of analog signals to prepare them for digitizing.

signal divider	Performing frequency division on an external signal.
single-ended inputs	Analog inputs that you measure with respect to a common ground.
Snap Mode	Mode that controls the available coordinates for cursors to line up on a plot.
software analog triggering	A method of triggering in which you to simulate an analog trigger using software.
SOURCE input pin	An counter input pin where the counter counts the signal transitions.
start condition	A condition on a data acquisition process that determines when the actual acquisition starts. The condition can be a software trigger, an analog hardware trigger, or a digital hardware trigger.
STC	System timing controller. *See* DAQ-STC.
stop condition	A condition on a data acquisition process that determines when the acquisition stops. The condition can be *none* (the acquisition stops when all points have been acquired), *continuous* (the acquisition runs continuously), *software analog trigger*, *hardware analog trigger*, or *hardware digital trigger*.
strain gauge	A thin conductor, which is attached to a material, that detects stress or vibrations in that material.
Style	The display style of a GUI object. An object can have different display styles while maintaining the same functionality. For example, the button can be an LED, toggle switch, vertical or horizontal slide, push button, command button, and more.
switchless device	Devices that do not require dip switches or jumpers to configure resources on the devices. Also called Plug and Play devices.
synchronous	Property or operation that begins an operation and returns control to the program only when the operation is complete.
syntax	The set of rules to which statements must conform in a particular programming language.

T

TC	Terminal count. The highest value of a counter.
thermocouple	A template sensing device which measures temperature by the changing electrical potential between two different metals.
toggled output	A form of counter signal generation by which the output changes the state of the output signal from high to low, or low to high when the counter reaches a certain value.
trace	A data line on a graph. Traces are generated using the Plot or Chart methods of the graph. Also called a plot.
track-and-hold	A circuit that tracks an analog voltage and holds the value on command.
Track Mode	Property of a graph that specifies how the mouse interacts with the graph. Use the Track Mode to turn on zooming and panning for the graph.
transducer excitation	A type of signal conditioning that uses external voltages and currents to excite the circuitry of a signal conditioning system into measuring physical phenomena.
trigger	A condition for starting or stopping clocks.

U

UI	User Interface.
unipolar	A signal range that is always positive, for example, 0 to 10 V.
update	One or more analog or digital output samples. Typically, the number of output samples in an update is equal to the number of channels in the output group. For example, one pulse from the update clock produces one update which sends one new sample to every analog output channel in the group.
update clock	Clock that sets the update rate for the AO, DI, and DO controls.
update rate	The number of output updates per second.

update width	The number of channels in the channel list or number of ports in the port list you use to configure an analog or digital output group.

V

V	Volts.
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	A control whose only valid values are its value pairs.
VB	Microsoft Visual Basic.
VC++	Microsoft Visual C++.
VDC	Volts, direct current.
Vref	Voltage reference.

W

waveform	Multiple voltage readings taken at a specific sampling rate.

Index

Numbers

1D and 2D operations (table), 7-3

A

AcquireData method, 6-11
ActiveX controls. *See also* events; methods; properties.
    setting properties, 1-8 to 1-13
    types of, 1-5 to 1-6
Advanced Analysis Library, 7-2
advanced application development, 8-1 to 8-21
    distributing applications using ComponentWorks, A-3 to A-4
    testing and debugging, 8-14 to 8-21
        breakpoints, 8-20
        Debug.Print command, 8-20
        error and warning events, 8-18 to 8-19
        error checking, 8-14 to 8-15
        exceptions, 8-15 to 8-16
        GetErrorText function, 8-19
        return codes, 8-17 to 8-18
        single stepping, 8-21
        Step Into mode, 8-21
        Step Over mode, 8-21
        watch window, 8-21
    Virtual Data Logger application, 8-11 to 8-14
    Virtual Oscilloscope application, 8-2 to 8-5
    Virtual Spectrum Meter application, 8-5 to 8-10
AI control, 6-9 to 6-14
    AI object, 6-10
    asynchronous acquisition, 6-10 to 6-11
    ChannelClock object, 6-12 to 6-13
    error handling, 6-12
    methods and events, 6-10 to 6-12
    PauseCondition object, 6-13 to 6-14
    ScanClock object, 6-12 to 6-13
    StartCondition object, 6-13 to 6-14
    StopCondition object, 6-13 to 6-14
    synchronous acquisition, 6-11 to 6-12
    tutorial, 6-14 to 6-19
        developing code, 6-17 to 6-18
        form design, 6-15 to 6-16
        setting DAQ properties, 6-16 to 6-17
        testing the program, 6-19
AI object, 6-10
AIPoint control, 6-6 to 6-8
    AIPoint object, 6-6 to 6-7
    object hierarchy (figure), 6-6
    tutorial, 6-14 to 6-19
        developing code, 6-17 to 6-18
        form design, 6-15 to 6-16
        setting DAQ properties, 6-16 to 6-17
        testing the program, 6-19
AIPoint object, 6-6 to 6-7
analog input, single point. *See* AIPoint control.
analog output, single point. *See* AOPoint control.
Analysis Error Codes, Table B-2, B-22 to B-26
Analysis Library controls
    descriptions in online reference manual, 7-13
    error messages, 7-13
    function tree (table)
        availability in different library versions, 7-2
        CWArray control, 7-3 to 7-4
        CWComplex control, 7-4 to 7-5
        CWDSP control, 7-8 to 7-12
        CWMatrix control, 7-5 to 7-6
        CWStat control, 7-6 to 7-7

functions, methods, and controls, 7-12
overview, 1-2
purpose and use, 1-13
questions and answers, A-4 to A-5
statistics function tutorial, 7-13 to 7-17
developing code, 7-15 to 7-16
form design, 7-14 to 7-15
testing the program, 7-17
versions of the library, 7-2
analysis of variance functions (table), 7-7
AO control, 6-21 to 6-25
AO object, 6-22
IntervalClock object, 6-24 to 6-25
methods and events, 6-23 to 6-24
object hierarchy (figure), 6-21
StartCondition object, 6-25
UpdateClock object, 6-24 to 6-25
AO object, 6-22
AOPoint control, 6-19 to 6-21
AOPoint object, 6-20
methods, 6-20 to 6-21
tutorial, 6-26 to 6-29
developing code, 6-27 to 6-29
form design, 6-26 to 6-27
testing the program, 6-29
AOPoint object, 6-20
application development. *See also* advanced application development.
Delphi applications, 4-1 to 4-8
Visual Basic applications, 2-1 to 2-11
Visual C++ applications, 3-1 to 3-11
array operation functions (table)
1D and 2D operations, 7-3
multidimensional array operations, 7-4
multidimensional element operations, 7-3 to 7-4
asynchronous acquisition methods and events, 6-10 to 6-11
axes, Virtual Data Logger application
formats, 8-13 to 8-14
multiple, 8-12 to 8-13
Axes collection, 5-22
Axis object, 5-6, 5-23

B

Base Analysis Library, 7-2
Borland Delphi. *See* Delphi applications.
breakpoints, 8-20
buffered waveform digital input. *See* DI control.
bulletin board support, C-1
button control. *See also* graph and button control tutorial.
events, 5-15
purpose and use, 5-14

C

calibration functions, 6-49
channel object, 6-8
channel strings
devices requiring reverse list of channels (note), 6-4
purpose and use, 6-3 to 6-4
SCXI channel strings, 6-4 to 6-5
ChannelClock object
AI control, 6-12 to 6-13
AIPoint control, 6-8
Channels collection, 6-7 to 6-8
charting data, 5-15. *See also* graph control.
ChartXvsY method, 5-18
ChartXY method, 5-18
ChartY method, 5-18, 5-29
clock objects
ChannelClock, 6-12 to 6-13
IntervalClock, 6-24 to 6-25
ScanClock, 6-12 to 6-13
UpdateClock, 6-24 to 6-25
collection objects, 1-7 to 1-8
collections
Axes, 5-22
Channels, 6-7 to 6-8
Cursors, 5-21
definition, 1-7
Lines, 6-33
Plots, 5-19
Pointers, 5-5

Ports, 6-32 to 6-33
ValuePairs collection, 5-7
common questions. *See* questions about ComponentWorks.
complex number functions (table), 7-4 to 7-5
complex operation functions, multidimensional (table), 7-5
ComponentWorks
components of, 1-1 to 1-2
general error codes, Table B-3, B-27 to B-28
overview, 1-1 to 1-2
questions about, A-1 to A-10
system requirements, 1-2 to 1-3
Configure functions, 6-48
Configure method
AO control, 6-23
asynchronous acquisition, AI control, 6-10 to 6-11
UpdateClock object, 6-42
control methods. *See* methods.
controls. *See* ActiveX controls; Analysis Library controls; specific controls.
Conversion functions, 6-48
counter/timer hardware, 6-50 to 6-65
Counter and Pulse controls tutorial, 6-60 to 6-65
designing the form, 6-60 to 6-62
developing the code, 6-62 to 6-64
testing your program, 6-65
counter/timer input and output, 6-50
counter control-counting and measurement preparations, 6-50 to 6-51
counter object, 6-51 to 6-53
measurement types, 6-51 to 6-53
methods and events, 6-53 to 6-54
AcquiredData event, 6-53
buffered measurements, 6-54 to 6-55
Pulse Control- digital pulse and pulsetrain generation, 6-55 to 6-60
Pulse obect, 6-56 to 6-57
pulse type operations, 6-56 to 6-57
methods, 6-58
FSK and ETS pulse generation, 6-59 to 6-60
FSK pulse generation mode
ETS pulse generation mode 6-59 to 6-60
Cursor object, 5-21 to 5-22
cursors, Virtual Spectrum Meter application, 8-8 to 8-10
Cursors collection, 5-21
curve fitting functions (table), 7-7
custom property page
CWGraph control example (figure), 1-9
definition, 1-9
customer communication, *xiv*, C-1 to C-2
CWArray control function tree, 7-3 to 7-4
1D and 2D operations, 7-3
multidimensional array operations, 7-4
multidimensional element operations, 7-3 to 7-4
CWButton control (table), 5-2
CWComplex control function tree, 7-4 to 7-5
complex numbers, 7-4 to 7-5
multidimensional complex operations, 7-5
CWDSP control function tree, 7-8 to 7-12
FIR digital filters, 7-10 to 7-11
frequency domain signal processing, 7-8 to 7-9
IIR digital filters, 7-9 to 7-10
measurement, 7-11 to 7-12
signal generation, 7-8
time domain signal processing, 7-9
windows, 7-11
CWGraph control
associated styles (table), 5-2
custom property page (figure), 1-9
CWKnob control (table), 5-2
CWMatrix control function tree, 7-5 to 7-6
CWNumEdit control (table), 5-2
CWSlide control (table), 5-2
CWStat control function tree, 7-6 to 7-7
analysis of variance, 7-7

curve fitting, 7-7
interpolation, 7-7
nonparametric statistics, 7-7
probability distributions, 7-6 to 7-7
statistics, 7-6

D

DAQ controls. *See* data acquisition controls.
DAQTools, 6-48 to 6-49
function groups, 6-48 to 6-49
using DAQTools functions, 6-49
data acquisition configuration, 6-2
data acquisition controls, 6-1 to 6-49
AI control, 6-9 to 6-14
AI object, 6-10
asynchronous acquisition, 6-10 to 6-11
ChannelClock object, 6-12 to 6-13
error handling, 6-12
methods and events, 6-10 to 6-12
PauseCondition object, 6-13 to 6-14
ScanClock object, 6-12 to 6-13
StartCondition object, 6-13 to 6-14
StopCondition object, 6-13 to 6-14
synchronous acquisition, 6-11 to 6-12
AIPoint and AI DAQ control tutorial, 6-14 to 6-19
developing code, 6-17 to 6-18
form design, 6-15 to 6-16
setting DAQ properties, 6-16 to 6-17
testing the program, 6-19
AIPoint control, 6-6 to 6-8
AIPoint object, 6-6 to 6-7
channel object, 6-8
ChannelClock object, 6-8
Channels collection, 6-7 to 6-8
AO control, 6-21 to 6-25
AO object, 6-22
IntervalClock object, 6-24 to 6-25
methods and events, 6-23 to 6-24
object hierarchy (figure), 6-21
StartCondition object, 6-25
UpdateClock object, 6-24 to 6-25
AOPoint control, 6-19 to 6-21
AOPoint object, 6-20
methods, 6-20 to 6-21
AOPoint control tutorial, 6-26 to 6-29
developing code, 6-27 to 6-29
form design, 6-26 to 6-27
testing the program, 6-29
counter/timer hardware, 6-50 to 6-65
counter/timer input and output, 6-50
DAQTools, 6-48 to 6-49
function groups, 6-48 to 6-49
using DAQTools functions, 6-49
data acquisition configuration, 6-2
DI control, 6-36
DI object, 6-36 to 6-37
methods and events, 6-38 to 6-39
UpdateClock object, 6-37 to 6-38
digital controls and hardware, 6-30 to 6-43
DIO control, 6-30 to 6-31
common properties and methods, 6-34 to 6-35
DIO object, 6-31 to 6-32
Ports collection and Port object, 6-32 to 6-33
DIO control tutorial, 6-44 to 6-48
developing code, 6-45 to 6-47
form design, 6-44 to 6-45
testing the program, 6-47 to 6-48
DO control, 6-40
DO object, 6-40 to 6-41
methods and events, 6-42 to 6-43
UpdateClock object, 6-41 to 6-42
error codes, Table B-1, B-1 to B-21
Lines collection and Line object, 6-33
common properties and methods, 6-34 to 6-35
list of DAQ controls, 6-1
object hierarchy and common properties, 6-2 to 6-6
channel strings, 6-3 to 6-4

Device, DeviceName, and DeviceType, 6-3
ExceptionOnError and ErrorEventMask, 6-5 to 6-6
SCXI channel strings, 6-4 to 6-5
overview, 1-1, 6-1 to 6-2
questions and answers, A-5 to A-7
data acquisition utility functions (DAQTools), 6-48 to 6-49
Data Logger application. *See* Virtual Data Logger application.
debugging and testing applications, 8-14 to 8-21
breakpoints, 8-20
Debug.Print command, 8-20
error and warning events, 8-18 to 8-19
error checking, 8-14 to 8-15
exceptions, 8-15 to 8-16
GetErrorText function, 8-19
return codes, 8-17 to 8-18
single stepping, 8-21
Step Into mode, 8-21
Step Over mode, 8-21
watch window, 8-21
default property page
definition, 1-8
Visual Basic example (figure), 1-9
Delphi applications, 4-1 to 4-8
building user interface, 4-3
editing properties programmatically, 4-5 to 4-6
events, 4-8
loading ComponentWorks controls, 4-1 to 4-3, A-3
methods, 4-7
online help for learning controls, 4-8
placing controls, 4-3 to 4-4
programming with ComponentWorks controls, 4-6 to 4-7
property sheets, 4-4 to 4-5
Device property, 6-3
DeviceName property, 6-3
DeviceType property, 6-3
DI control, 6-36
DI object, 6-36 to 6-37
digital controls and hardware, 6-30 to 6-43
common properties and methods, 6-34 to 6-35
DI control--buffered waveform digital input, 6-36
DI object, 6-36 to 6-37
DIO control--single point digital input and output, 6-30 to 6-31
DIO object, 6-31 to 6-32
DO control--buffered waveform digital output, 6-40
DO object, 6-40 to 6-41
Lines collection and Line object, 6-33
methods and events
DI control, 6-38 to 6-39
DO control, 6-42 to 6-43
Ports collection and Port object, 6-32 to 6-33
UpdateClock object
DI control, 6-37 to 6-38
DO control, 6-41 to 6-42
Digital Signal Processing Analysis Library, 7-2
digital signal processing and signal generation functions
FIR digital filters (table), 7-10 to 7-11
frequency domain signal processing (table), 7-8 to 7-9
IIR digital filters (table), 7-9 to 7-10
measurement functions (table), 7-11 to 7-12
signal generation functions (table), 7-8
time domain signal processing functions (table), 7-9
Virtual Spectrum Meter application, 8-7 to 8-8
windows functions (table), 7-11
DIO control
single point digital input and output, 6-30 to 6-31

tutorial, 6-44 to 6-48
developing code, 6-45 to 6-47
form design, 6-44 to 6-45
testing the program, 6-47 to 6-48
DIO object
common properties and methods, 6-34 to 6-35
purpose and use, 6-31 to 6-32
distributing applications using ComponentWorks, A-3 to A-4
DO control, 6-40
DO object, 6-40 to 6-41
documentation
conventions used in manual, *xiii*
organization of manual, *xi-xii*
DSP functions. *See* digital signal processing and signal generation functions.

E

electronic support services, C-1 to C-2
e-mail support, C-2
equivalent time sampling, See ETS
error and warning events, 8-18 to 8-19
error checking/handling
AI control, 6-12
DAQ controls, 6-5 to 6-6
GetErrorText function, 6-48, 8-19
testing applications, 8-14 to 8-15
error codes, B-1 to B-28
Analysis Error Codes, Table B-2, B-22 to B-26
Data Acquisition Control Error Codes, Table B-1, B-1 to B-21
General ComponentWorks Error Codes, Table B-3, B-27 to B-28
error messages
Analysis Library controls, 7-13
exceptions, 8-15 to 8-16
ErrorEventMask property, 6-5 to 6-6
ETS and FSK pulse generation, 6-59 to 6-60
ETS pulse generation mode 6-59 to 6-60
FSK pulse generation mode, 6-59
event handler routines
developing, 1-12
Visual Basic applications, 2-6 to 2-7
events
AO control, 6-23 to 6-24
asynchronous acquisition, 6-10 to 6-11
button control, 5-15
definition, 1-5
DI control, 6-38 to 6-39
DO control, 6-42 to 6-43
error and warning events, 8-18 to 8-19
graph control, 5-23
knob and slide controls, 5-8
numeric edit box control, 5-9 to 5-10
synchronous acquisition, 6-11 to 6-12
using in applications
Delphi applications, 4-8
Visual C++ applications, 3-10 to 3-11
ExceptionOnError property, 6-5 to 6-6
exceptions
examples, 8-16
testing applications, 8-15 to 8-16

F

fax and telephone support, C-2
Fax-on-Demand support, C-2
file input/output functions, Virtual Data Logger application, 8-14
FIR digital filters (table), 7-10 to 7-11
forms
Delphi applications, 4-1
Visual Basic applications, 2-1
FOUT functions, 6-49
frequency domain signal processing (table), 7-8 to 7-9
frequency shift keying, *See* FSK
FSK and ETS pulse generation, 6-59 to 6-60
FSK pulse generation mode, 6-59
ETS pulse generation mode 6-59 to 6-60
FTP support, C-1

G

GetErrorText function, 6-48, 8-19
graph and button control tutorial, 5-25 to 5-29
    developing program code, 5-27 to 5-28
    form design, 5-25 to 5-26
    testing the program, 5-29
graph axes, Virtual Data Logger application
    formats, 8-13 to 8-14
    multiple, 8-12 to 8-13
graph control, 5-15 to 5-24
    Axes collection, 5-22
    Axis object, 5-23
    chart methods, 5-18
    Cursor object, 5-21 to 5-22
    Cursors collection, 5-21
    events, 5-23
    Graph object, 5-17
    hierarchy of (figure), 5-16
    overview, 1-5 to 1-6
    panning and zooming, 5-24
    plot methods, 5-17 to 5-18
    Plot object, 5-19 to 5-20
    Plots collection, 5-19
    PlotTemplate object, 5-20
    purpose and use, 5-15 to 5-16
    tutorial, 5-25 to 5-29
Graph object, 5-17
graph track mode, Virtual Spectrum Meter application, 8-10
Graphical User Interface controls, 5-1 to 5-29
    button control, 5-14 to 5-15
        events, 5-15
    controls and associated styles (table), 5-2
    graph and button control tutorial, 5-25 to 5-29
        developing program code, 5-27 to 5-28
        form design, 5-25 to 5-26
        testing the program, 5-29
    graph control, 5-15 to 5-24
        Axes collection, 5-22
        Axis object, 5-23
        chart methods, 5-18
        Cursor object, 5-21 to 5-22
        Cursors collection, 5-21
        events, 5-23
        Graph object, 5-17
        hierarchy of (figure), 5-16
        panning and zooming, 5-24
        plot methods, 5-17 to 5-18
        Plot object, 5-19 to 5-20
        Plots collection, 5-19
        PlotTemplate object, 5-20
        tutorial, 5-25 to 5-29
    knob, slide, and numeric edit box control tutorial, 5-10 to 5-13
        developing program code, 5-11 to 5-12
        form design, 5-10 to 5-11
        testing the program, 5-13
    knob and slide controls, 5-4 to 5-8
        Axis object, 5-6
        events, 5-8
        hierarchy of (figure), 5-4
        knob and slide object, 5-5
        Pointer object, 5-5
        Pointers collection, 5-5
        Statistics object, 5-8
        Ticks and Labels objects, 5-6
        tutorial, 5-10 to 5-13
        ValuePair object, 5-7
        ValuePairs collection, 5-7
    numeric edit box control, 5-9 to 5-10
        events, 5-9 to 5-10
    object hierarchy and common objects, 5-3
    questions and answers, A-7 to A-10
GUI controls. *See* Graphical User Interface controls.

H

Help button, 1-13
help files, online. *See* online help files.

I

ICtr functions, 6-48 to 6-49
IIR digital filters (table), 7-9 to 7-10
installation, 1-2 to 1-4
    from CD-ROM, 1-3
    from floppy disks, 1-3
    installed files, 1-4
    instrument driver DLLs, 1-4
    procedure, 1-2 to 1-4
    questions about ComponentWorks, A-1 to A-4
    system requirements, 1-2 to 1-3
instrument driver DLLs
    installing, 1-4
    using in Visual Basic applications, 2-7 to 2-8
instrument drivers
    definition, 2-7
    overview, 1-2
interchannel delay, 6-12
interpolation functions (table), 7-7
IntervalClock object, 6-24 to 6-25
Item method, 1-11

K

knob and slide controls, 5-4 to 5-8
    Axis object, 5-6
    events, 5-8
    hierarchy of (figure), 5-4
    knob and slide object, 5-5
    Pointer object, 5-5
    Pointers collection, 5-5
    Statistics object, 5-8
    Ticks and Labels objects, 5-6
    tutorial, 5-10 to 5-13
        developing program code, 5-11 to 5-12
        form design, 5-10 to 5-11
        testing the program, 5-13
    ValuePair object, 5-7

L

Labels object, 5-6
Line object, 6-33
Lines collection, 6-33

M

matrix algebra functions (table), 7-5 to 7-6
measurement functions (table), 7-11 to 7-12
methods. *See also* specific methods.
    AI object, 6-10 to 6-12
        asynchronous acquisition, 6-10 to 6-11
        synchronous acquisition, 6-11 to 6-12
    AO control, 6-23 to 6-24
    AOPoint object, 6-20 to 6-21
    asynchronous acquisition, 6-10 to 6-11
    definition, 1-5
    DI control, 6-38 to 6-39
    DIO control, 6-34 to 6-35
    DO control, 6-42 to 6-43
    functions as methods of corresponding control, 7-12
    parameters, 2-5 to 2-6
    synchronous acquisition, 6-11 to 6-12
    using in applications
        Delphi applications, 4-7
        Visual Basic applications, 2-5 to 2-6
        Visual C++ applications, 3-9
    working with control methods, 1-11 to 1-12
multidimensional array operations (table), 7-4
multidimensional complex operations (table), 7-5
multidimensional element operations (table), 7-3 to 7-4
multiple graph axes, Virtual Data Logger application, 8-12 to 8-13

N

NI-DAQ driver configuration utility, 6-2
nonparametric statistics functions (table), 7-7
numeric edit box control, 5-9 to 5-10
    events, 5-9 to 5-10
    purpose and use, 5-9
    tutorial, 5-10 to 5-11
        developing program code, 5-11 to 5-12
        form design, 5-10 to 5-11
        testing the program, 5-13

O

object browser, in Visual Basic, 2-8 to 2-10
object hierarchy
    AIPoint control (figure), 6-6
    AO control (figure), 6-21
    DAQ controls, 6-2 to 6-6
    graph control (figure), 5-16
    knob and slide controls (figure), 5-4
    purpose and use, 1-6 to 1-7
    similarity in different controls, 5-3
    Slide object example, 1-7
one dimensional operations (table), 7-3
online help files
    descriptions of analysis functions, 7-13
    learning about ComponentWorks controls, 1-13
        Delphi applications, 4-8
        Visual Basic applications, 2-11
        Visual C++ applications, 3-11
Oscilloscope application. *See* Virtual Oscilloscope application.

P

panning and zooming graphs, 5-24
parameters for methods, 2-5 to 2-6
PauseCondition object, 6-13 to 6-14
Plot object, 5-19 to 5-20
Plots collection, 5-19
PlotTemplate object, 5-20
plotting data, 5-15. *See also* graph control.
PlotXvsY method, 5-17 to 5-18
PlotXY method, 5-17
PlotY method
    changing properties programmatically (example), 1-11 to 1-12
    format for accepting data, 5-17
    graph and button control tutorial, 5-27
Pointer object, 5-5
Pointers collection, 5-5
Port object, 6-32 to 6-33
Ports collection, 6-32 to 6-33
pretriggering modes, Virtual Oscilloscope application, 8-3
probability distribution functions (table), 7-6 to 7-7
properties
    definition, 1-5
    DIO, Port, and Line objects, 6-34 to 6-35
    editing programmatically
        Delphi applications, 4-5 to 4-6
        Visual Basic applications, 2-4 to 2-5
    setting, 1-8 to 1-13
        Analysis Library and instrument driver DLLs, 1-13
        changing properties programmatically, 1-10
        developing event handler routines, 1-12
        help file, 1-13
        Item method, 1-11
        property sheets, 1-8 to 1-9
        working with control methods, 1-11 to 1-12
    Visual C++ applications, 3-7 to 3-9
property sheets
    custom property page, 1-9
    default property page, 1-8 to 1-9
    Delphi applications, 4-4 to 4-5
    setting default values, 1-8
    Visual Basic applications, 2-3 to 2-4
Pulse Control- digital pulse and pulsetrain generation, 6-55 to 6-60

FSK and ETS pulse generation, 6-59 to 6-60
FSK pulse generation mode
ETS pulse generation mode 6-59 to 6-60
methods, 6-58
Pulse object, 6-56 to 6-57
pulse type operations, 6-56 to 6-57
Pulse and Counter controls tutorial, 6-60 to 6-65
designing the form, 6-60 to 6-62
developing the code, 6-62 to 6-64
testing your program, 6-65

Q

questions about ComponentWorks, A-1 to A-10
analysis controls, A-4 to A-5
DAQ controls, A-5 to A-7
installation and getting started, A-1 to A-4
user interface controls, A-7 to A-10

R

Reset functions, 6-48
Reset method
AIPoint object, 6-6, 6-7
AO control, 6-23
AOPoint object, 6-20, 6-21
asynchronous acquisition, 6-10 to 6-11
UpdateClock object, 6-42
return codes, testing applications, 8-17 to 8-18

S

scan rate, 6-12
ScanClock object, 6-12 to 6-13
SCXI channel strings, 6-4 to 6-5
Set functions, 6-48
setting properties. *See* properties, setting.
signal processing functions. *See* digital signal processing and signal generation functions.
single point analog input. *See* AIPoint control.
single point analog output. *See* AOPoint control.
single stepping, 8-21
SingleRead method
AIPoint object, 6-6 to 6-7
DIO, Port, and Line objects, 6-34
SingleWrite method
AOPoint object, 6-20
DIO, Port, and Line objects, 6-34
slide controls. *See* knob and slide controls.
software objects, 1-6
Spectrum Meter application. *See* Virtual Spectrum Meter application.
Start method
AO control, 6-23
asynchronous acquisition, AI control, 6-10
UpdateClock object, 6-42
StartCondition object, 6-13 to 6-14
statistics functions
analysis of variance functions (table), 7-7
curve fitting functions (table), 7-7
interpolation functions (table), 7-7
nonparametric statistics functions (table), 7-7
probability distribution functions (table), 7-6 to 7-7
simple statistics (table), 7-6
tutorial, 7-13 to 7-17
developing code, 7-15 to 7-16
form design, 7-14 to 7-15
testing the program, 7-17
Statistics object, 5-8
Step Into mode, 8-21
Step Over mode, 8-21
stop condition modes, Virtual Oscilloscope application, 8-3
Stop method, asynchronous acquisition, 6-10

StopCondition object, 6-13 to 6-14
synchronous acquisition methods and events, 6-11 to 6-12
system requirements, 1-2 to 1-3

T

technical support, C-1 to C-2
telephone and fax support, C-2
testing applications. *See* debugging and testing applications.
Ticks object, 5-6
time domain signal processing functions (table), 7-9
TrackMode property
    Graph object, 5-17
    panning and zooming, 5-24
    Virtual Spectrum Meter application, 8-10
two dimensional operations (table), 7-3

U

UpdateClock object
    DI control, 6-37 to 6-38
    DO control, 6-41 to 6-42
user interface, building. *See also* Graphical User Interface controls.
    Delphi applications, 4-3
    Visual Basic applications, 2-2
    Visual C++ applications, 3-4 to 3-5
user interface controls. *See also* Graphical User Interface controls.
    overview, 1-1
    questions and answers, A-7 to A-10

V

value pairs, Virtual Oscilloscope application, 8-4 to 8-5
ValuePair object, 5-7
ValuePairs collection, 5-7
vector and matrix algebra functions (table), 7-5 to 7-6
Virtual Data Logger application, 8-11 to 8-14
    file input/output, 8-14
    graph axes formats, 8-13 to 8-14
    multiple graph axes, 8-12 to 8-13
Virtual Oscilloscope application
    data acquisition pretriggering, 8-3
    data acquisition stop condition modes, 8-3
    user interface value pairs, 8-4 to 8-5
Virtual Spectrum Meter application, 8-5 to 8-10
    cursors, 8-8 to 8-10
    DSP Analysis Library functions, 8-7 to 8-8
    graph track mode, 8-10
Visual Basic applications, 2-1 to 2-11
    building user interface, 2-2
    default property page (figure), 1-9
    developing event handler routines, 2-6 to 2-7
    editing properties programmatically, 2-4 to 2-5
    instrument driver DLLs, 2-7 to 2-8
    loading ComponentWorks controls into toolbox, 2-1 to 2-2
    object browser for building code, 2-8 to 2-10
    online help for learning controls, 2-11
    pasting code into programs, 2-11
    property sheets, 2-3 to 2-4
    questions about ComponentWorks, A-1 to A-3
    working with control methods, 2-5 to 2-6
Visual C++ applications, 3-1 to 3-11
    adding ComponentWorks controls to toolbar, 3-3 to 3-4
    building user interface, 3-4 to 3-5
    creating applications, 3-1 to 3-3
    events, 3-10 to 3-11
    methods, 3-9
    online help for learning controls, 3-11
    programming with ComponentWorks controls, 3-5 to 3-6
    properties, 3-7 to 3-9

W

warning events, 8-18 to 8-19
watch window, 8-21
waveform analog input. *See* AI control.
waveform analog output. *See* AO control.
waveform digital input, buffered. *See* DI control.
windows functions (table), 7-11
Write method
    AO control, 6-23
    UpdateClock object, 6-42, 6-43

Z

zooming graphs, 5-24