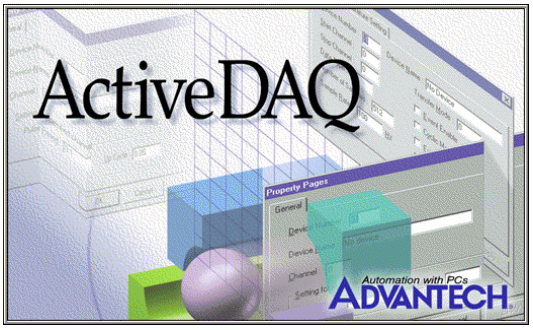
ActiveDAQ
User's Manual



1st Edition

Copyright Notice

Acknowledgements

Advantech Customer Services

Each and every Advantech product is built to the most exacting specifications to ensure reliable performance in the harsh and demanding conditions typical of industrial environments. Whether your new Advantech equipment is destined for the laboratory or the factory floor, you can be assured that your product will provide the reliability and ease of operation for which the name Advantech has come to be known.

Your satisfaction is our primary concern. Here is a guide to Advantech's customer services. To ensure you get the full benefit of our services, please follow the instructions below carefully.

Technical Support

We want you to get the maximum performance from your products. So if you run into technical difficulties, we are here to help. For the most frequently asked questions, you can easily find answers in your product documentation. These answers are normally a lot more detailed than the ones we can give over the phone.

So please consult this manual first. If you still cannot find the answer, gather all the information or questions that apply to your problem, and with the product close at hand, call your dealer. Our dealers are well trained and ready to give you the support you need to get the most from your Advantech products. In fact, most problems reported are minor and are able to be easily solved over the phone.

In addition, free technical support is available from Advantech engineers every business day. We are always ready to give advice on application requirements or specific information on the installation and operation of any of our products.

Technical Suppport Offices

USA

American Advantech Corporation
750 East Arques Avenue
Sunnyvale, CA 94086
Tel: (408)245-6678
Fax: (408)245-5678
E-mail: IAInfo@advantech.com

Asia

Advantech Co., LTD
4th Floor, 108-3 Min-Chuan Road
Shing-Tien City, Taipei County, Taiwan ROC
Tel: (+886-2) 2218-4567
Fax: (+886-2) 2218-1989
E-mail: IASupport@advantech.com.tw

Europe

Advantech Germany
Karlsruherstr. 11/1
D-70771 Leinf.-Echterdingen
Germany
Tel: +49 (0) 711 797 333 60
Fax: +49 (0) 711 797 333 85

Advantech Italy
Via Don Verderio
4/B-20060 Cassina de,
Pecchi (MI), Italy
Tel: +39-2-95343054
Fax: +39-2-95343067

Mainland China

Beijing office:
No. 7, 6th Street, Shang Di Zone
Haidian District, 100085
Beijing, China
Tel: +86-10-62984345~47, 62986314~17
Fax: +86-1-62984341~42

Shanghai office:
Room #701, 7th Floor, Hua-Fu Building A
585 Long Hua W. Road
200232 Shanghai, China
Tel: +86-21-64696831, 64697910
Fax: +86-21-64696834

Limited Warranty

Advantech Corporation does not warrant that the ActiveDAQ software package will function properly in every hardware/software environment. Advantech Corporation makes no representation or warranties of any kinds whatsoever with respect to the contents of this manual and specifically disclaims any implied warranties or fitness for any particular purpose. Advantech Corporation shall not be held liable for errors in this manual or for incidental or consequential damages in connection with the use of this manual or its contents. Advantech Corporation reserves the right to revise this manual at any time without prior notice.

About This Manual

This manual contains the information you need to get started with the ActiveDAQ software package. ActiveDAQ allows you to easily perform versatile I/O operations through properties, methods and events in programs developed with Microsoft Visual Basic, Microsoft Visual C++, Delphi and other ActiveX control container environments.

This manual contains step-by-step instructions for building applications with ActiveDAQ. You can modify these sample applications to suit your needs. This manual does not show you how to use every control or solve every possible programming problem. Specific questions should be directed to Advantech's application engineers.

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

This user manual is divided into the following sections:

- Chapter 1, ***Introduction to ActiveDAQ Controls***, introduces the ActiveDAQ ActiveX controls and how they can be used in your applications to get the most out of Advantech's Data Acquisition and Control cards. It also explains how ActiveDAQ works with the Device Installation Utility. Complete installation instructions for the ActiveDAQ controls and the Device Installation Utility are also included. In addition, Chapter 1 explains how to use the Device Installation Utility to enable your computer to use Advantech's DA & C hardware. This must be completed before you can write programs using ActiveDAQ to access your hardware.
- Chapter 2, ***Building ActiveDAQ Applications with Various Languages*** briefly explains how to use ActiveDAQ controls in three popular development environments.

- Chapter 3, ***Tutorial*** gives the new user a walk-through in creating a simple application in Visual Basic 5.0 that uses ActiveDAQ controls. Users of other development tools will also find this section informative in understanding how ActiveDAQ controls can simplify many programming tasks.
- Chapter 4, ***Using ActiveDAQ Controls*** gives step-by-step instructions for creating sample applications using the Analog Input Control, Analog Output Control, Digital Input Control, Digital Output Control, Counter Control, Pulse Output Control and the Alarm Control. A comprehensive introduction to the controls' properties, methods and events is included.
- Appendix A, ***Properties, Methonds and Events Reference*** is a listing of all the properties, methods and events that are supported by the ActiveDAQ controls.
- Appendix B, ***Error Messages*** is a listing of all the error and warning messages that you might see when using or programming the ActiveDAQ controls.
- Appendix C, ***Hardware Support Listing*** shows the ActiveDAQ methods that are supported by each of Advantech's products.

Contents

Figures

Tables

CHAPTER 1

Introduction to ActiveDAQ Controls

1.1 Introduction to ActiveDAQ

The following is an overview of ActiveDAQ, lists the ActiveDAQ system requirements, describes how to install and uninstall the software, and explains the basics of ActiveX controls.

1.2 What is ActiveDAQ?

ActiveDAQ is a collection of ActiveX controls for performing I/O operations within any compatible ActiveX control container, such as Visual Basic, Delphi, etc. You can easily perform the I/O operations through properties, events and methods. Specific information about the properties, methods, and events of the individual ActiveX controls can be found later in this manual.

With ActiveDAQ, you can perform versatile I/O operations to control your Advantech devices. The ActiveDAQ package contains the following components:

- **DAQDevice**: opens a dialog box for the user to select the device that they want to operate.
- **DAQAI**: performs single analog input and waveform analog input operations.
- **DAQAO**: performs analog output and waveform output operation.
- **DAQDI**: performs digital input operation.
- **DAQDO**: performs digital output operation.
- **DAQCounter**: performs event-counting and frequency measurement operation.
- **DAQPulse**: performs pulse output.
- **DAQAlarm**: performs alarm settings and alarm checking.

You can use these ActiveX controls in any application that supports them, including Microsoft Visual C++, Visual Basic, and Delphi.

1.3 Installing ActiveDAQ

The ActiveDAQ setup program installs ActiveDAQ through a process that lasts approximately five minutes. Installing the necessary software to use the ActiveDAQ ActiveX controls in your application involves two main steps:

1. Installing the ActiveDAQ controls and/or the DLL drivers
2. Using the Device Installation Utility to install and configure the drivers for the hardware that is attached to your computer

1.3.1 System Requirements

To use the ActiveDAQ ActiveX controls, you must have the following:

- Microsoft Windows 95/98 or Windows NT operating system
- Personal computer using at least a 33 MHz 80486 or higher microprocessor (66 MHz 80486 or higher microprocessor recommended)
- VGA resolution (or higher) video adapter
- ActiveX control container such as Visual Basic (32-bit version), Visual C++, or Delphi (32-bit version)
- Minimum of 16 MB of memory
- Minimum of 15 MB of free hard disk space
- Microsoft-compatible mouse

1.3.2 Installing the ActiveDAQ files and DLL Drivers

1. Insert the ActiveDAQ installation CD-ROM disc into your computer.
2. The installation program should start automatically. If auto-run is not enabled on your computer, use your Windows Explorer or the Windows Run command to execute setup.exe on the ActiveDAQ installation CD-ROM disc (assume “d” is the letter of your CD-ROM disc drive):

```
d:\setup.exe  (for both Windows 98/95 and NT)
```

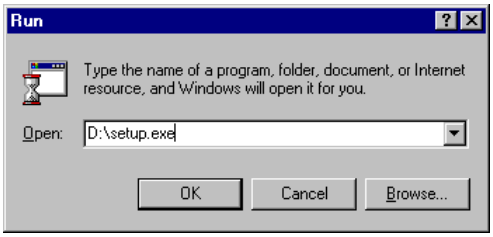


Figure 1-1: Run setup.exe from the installation CD-ROM disc

Follow the instructions in the installation wizard.

3. The ActiveDAQ installation program loads:

Figure 1-2: Loading the ActiveDAQ Setup Program

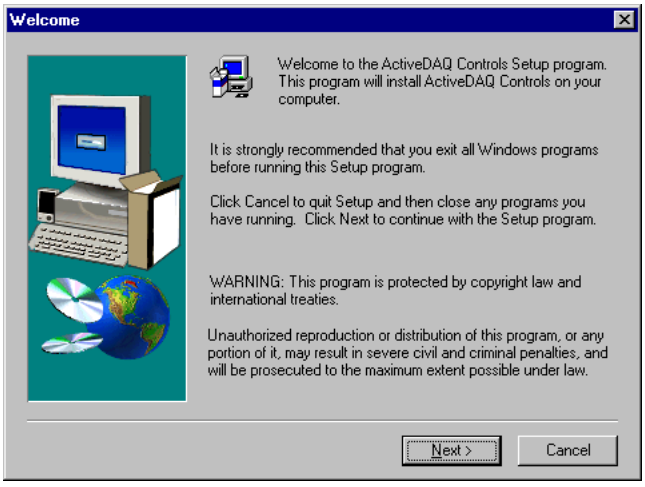


Figure 1-3: ActiveDAQ Setup Program Welcome Screen

4. An information screen is displayed. Click **Next** to advance to the next screen.

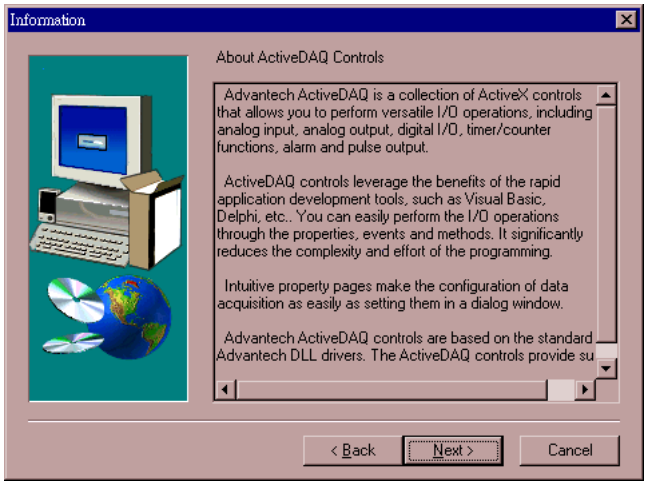


Figure 1-4: Installation Program Information Screen

5. You have to specify a file path on your local computer to install the ActiveDAQ and DLL driver files. The default path (for both Windows 95/98 and Windows NT systems) is:

   `C:\Program Files\Advantech`

You can choose a different directory path by clicking the **Browse** button.

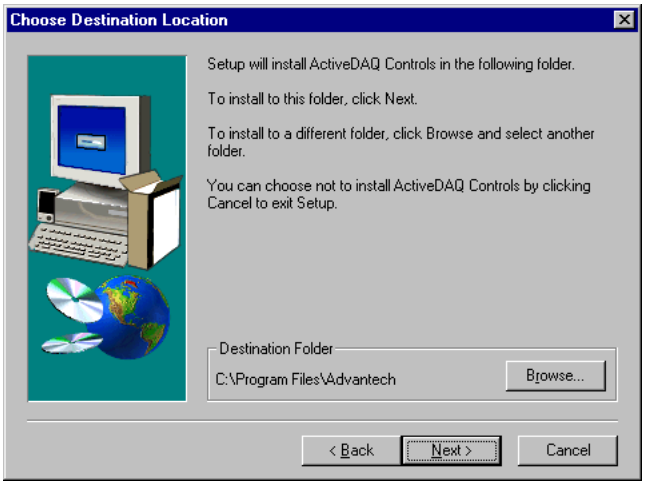


Figure 1-5: Choose Destination Directory for Program Files

6. Choose the kind of installation you want to perform. Your choices are:
   a. ActiveDAQ Only
   b. ActiveDAQ with Windows 95/98 Driver 1.11
   c. ActiveDAQ with Windows NT Driver 1.11

Choose **a** only if you have previously installed the version 1.11 DLL drivers that are appropriate for your operating system. Most users should choose options **b** or **c**. Click the **Next** button after making your selection.

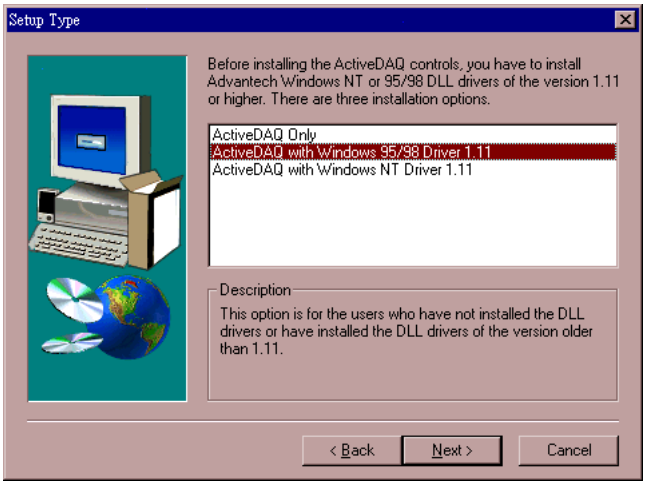


Figure 1-6: Choose the type of installation for your operating system

7. The ActiveDAQ installation program will copy a program shortcut to your Windows Programs menu. The name for the shortcut is "Advantech ActiveDAQ". You can change the name of the program shortcut if you want.

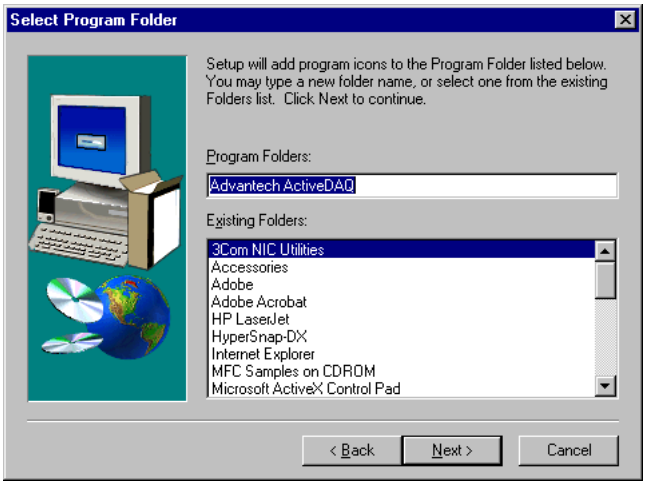


Figure 1-7: Making Shortcuts on Windows Start Menu

8. The ActiveDAQ setup program will now copy the required files to your computer's hard disk drive. A progress bar will display the files being copied to your computer. The DLL drivers and ActiveDAQ controls that enable communication with your Advantech board will be copied to either the C:\windows\system\ directory (for Windows 95/98) or C:\winnt\system32\ directory (for Windows NT 4.0).

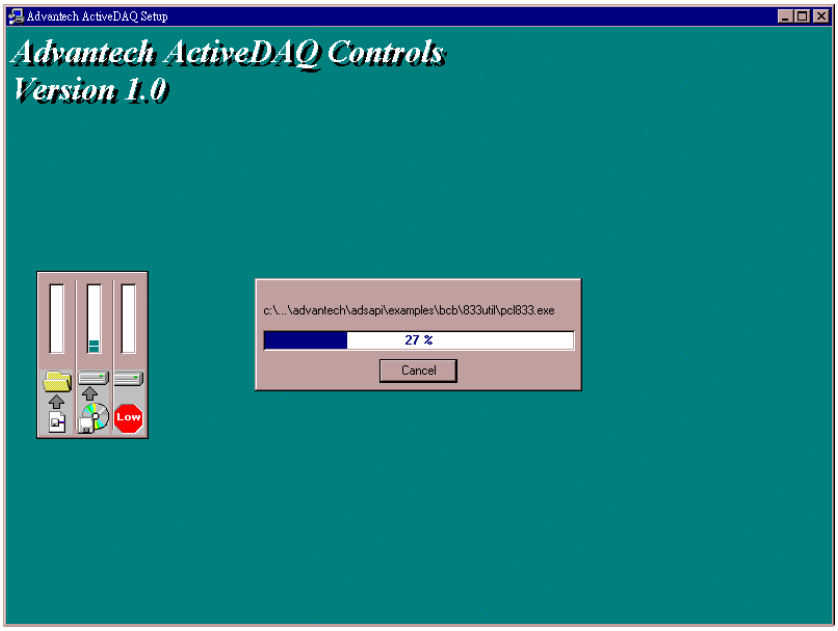


Figure 1-8: Copying Files to Hard Disk Drive

9. When the installation program finishes copying all the program files to your hard disk drive, the Setup Complete window will be displayed. Click the **Finish** button to close the installation program

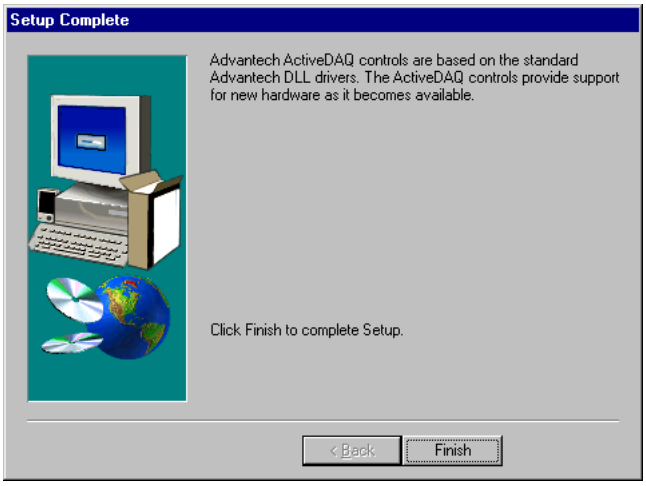


Figure 1-9: Installation Program Setup Complete Window

10. The installation of the ActiveDAQ controls is now finished and the program will close.

You will notice that the Advantech ActiveDAQ program shortcuts have been copied to your Windows Start menu so that you can easily launch the program and view the related documentation:

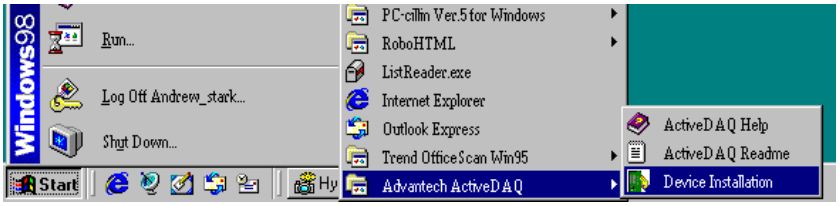


Figure 1-10: ActiveDAQ Shortcuts on the Windows Start Menu

Examples and documentation were copied to the file path that you specified in step 5. You can open your Windows Explorer to verify that these files were copied correctly. Specifically, the default file paths are:

```
c:\Program Files\Advantech\ActiveDAQ
```

which contains examples and additional documentation about the ActiveDAQ controls and

```
c:\Program Files\Advantech\Adsapi
```

that contains the Device Installation Utility for loading the DLL drivers to enable communication between the ActiveDAQ controls and your hardware.

If you want to verify that that the ActiveX controls were installed correctly and registered to your computer, you can open your development tool and check to see that the ActiveX controls are available. For example, in Microsoft Visual Basic, the ActiveDAQ controls that were installed will appear in the Components dialog box by choosing **Project | Components**…from Visual Basic's main menu:

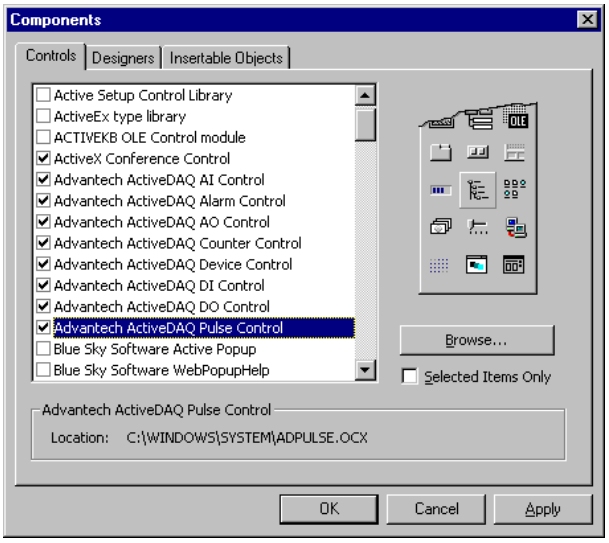


Figure 1-11: VB Components Dialog Box Showing ActiveDAQ

When added to your current VB project, the ActiveX controls will appear on the Visual Basic toolbar:

Figure 1-12: Visual Basic Toolbar Showing ActiveDAQ Control

1.3.3 Running the Device Installation Utility

You must run the Device Installation utility to install the DLL driver for your Advantech card before using the ActiveDAQ controls.

1. Start the Device Installation Utility by double-clicking on the Devinst.exe program icon in your Windows Explorer or choose the program shortcut on your Windows Start menu.
2. The Device Installation Utility loads. Choose **Setup | Device** on the Device Installation Utility's main menu.

Figure 1-13: Device Installation Utility Main Screen

3. The I/O Device Installation window loads. The pane at the top of the I/O Device Installation window displays the Advantech boards that are installed on the computer – this pane will be blank the first time you run the utility. Click the **Add** button to display the list of devices for which you can install the driver.

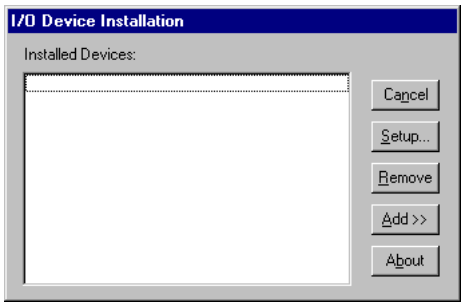


Figure 1-14: Device Installation Utility I/O Device Installation Window

4. The I/O Device Installation window expands to show a list of devices for which you can install the DLL driver. Choose the card name and model in the List of Devices list box. When the card that you want to install is highlighted, click the **Install** button.

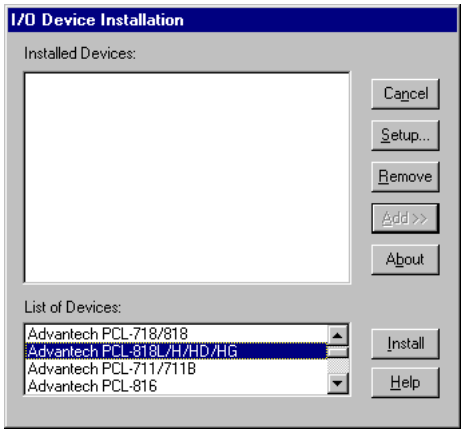


Figure 1-15: Choose Board Driver That You Want to Install

5. The Card/Device Setup dialog box appears. Enter your configuration settings that you want to use for the hardware. Consult your hardware documentation for more information about how to configure the hardware. The following figure shows the Setup dialog box for Advantech's PCL-818L data acquisition card.

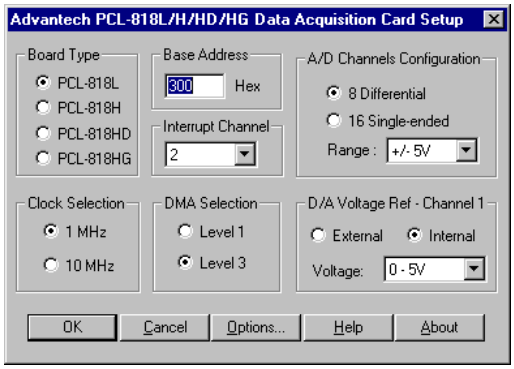


Figure 1-16: Card/Device Setup dialog box (PCL-818L shown)

6. The card's model and name will now appear in the upper pane of the I/O Device Installation window. If you want to change the configuration settings, highlight the entry and then click the **Setup** button. To remove it from your system, click the **Remove** button.

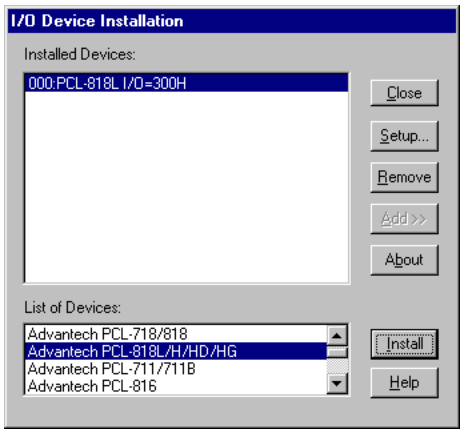


Figure 1-17: I/O Device Installation Window Showing Installed Devices

7. Installation and configuration of your Advantech board is now complete. Close the Device Installation Utility by selecting **File | Exit** from the Device Installation Utility's main menu.

The above example showed the installation of the DLL driver for only one Advantech board. You can install drivers for other boards by following the same procedure.

1.3.4 Uninstalling the ActiveDAQ Controls and DLL Drivers

ActiveDAQ includes an uninstallation utility to remove the files from your computer. To uninstall all the ActiveDAQ controls, DLL drivers and their related files, complete the following procedure:

1. Choose **Settings | Control Panel** from your Windows Start menu.
2. Double-click on the icon labeled "Add/Remove Programs"
3. Select the item labeled Advantech ActiveDAQ and click the **Add/ Remove** button.

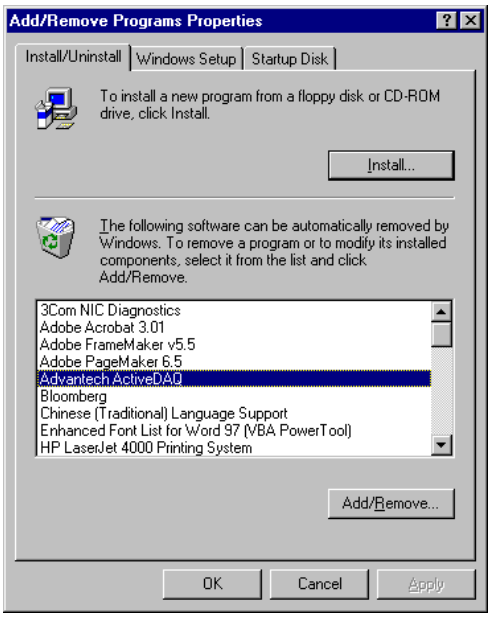


Figure 1-18: Add/Remove Programs in the Windows Control Panel

4. Confirm that you want to delete the files by clicking **Yes**.

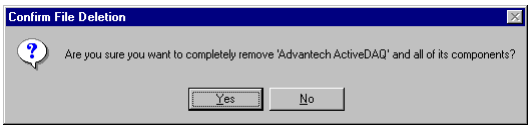


Figure 1-19: Click Yes to Uninstall ActiveDAQ

5. The uninstallation utility will start running. Click the **OK** button when uninstallation is completed.

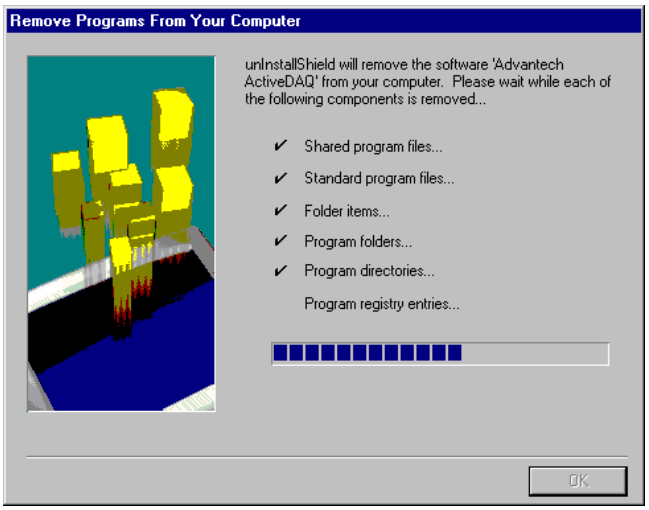


Figure 1-20: Uninstallation utility removing ActiveDAQ programs

CHAPTER 2

Building ActiveDAQ Applications With Various Languages

2.1 Using Various Development Environments

This chapter describes how can you use the ActiveDAQ controls with the following development tools:

- Microsoft Visual C++ for Windows 95/NT version 5.0
- Microsoft Visual Basic for Windows 95/NT version 5.0
- Inprise Delphi for Windows 95/NT version 4.0

If you are not using one of these development tools, consult your development tool reference manual for details on creating applications with ActiveX controls.

This chapter assumes that you are familiar with the basic concepts of using Visual Basic, Delphi and Visual C++, including selecting the type of application, designing the form, placing the control on the form, configuring the properties of the control, creating the code (event handler routines) for this control.

2.2 Developing Visual Basic Applications

To use ActiveDAQ controls, complete the following procedure:

2.2.1 Loading ActiveDAQ Controls into the VB Toolbox

Before using ActiveDAQ controls to build an application, you must add them to the Visual Basic toolbox. You can follow the procedure to add them:

1. Select **Components** from the **Project** menu
2. Scroll down to the ActiveDAQ controls, which you can find in the Controls list.
3. Select the controls you want to use in your project. If ActiveDAQ controls are not shown in the list, press the **Browse** button to select the desired controls from the \WINDOWS\SYSTEM(32) directory.

The Components list dialog box is as below:

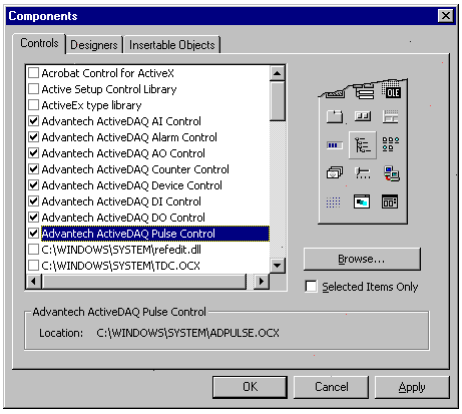


Figure 2-1: Visual Basic Components list dialog box showing ActiveDAQ controls

2.2.2 Configuring Controls with Property Sheets

After you add the ActiveDAQ controls to the Visual Basic toolbox, select the corresponding icon in the toolbox and place it on a Visual Basic form. You can then edit or configure its properties in the Visual Basic default property sheet. To access the sheet, select a control and select **Properties Window** from the **View** menu. For example, the property sheet of ActiveDAQ's Analog Input Control is shown in the figure below.

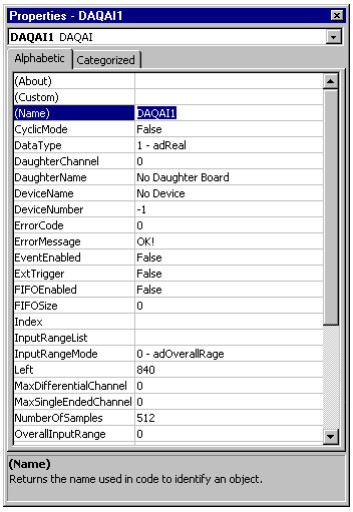


Figure 2-2: ActiveDAQ Analog Input Control property sheet in Visual Basic

In addition, you also can read and set the properties of ActiveDAQ controls at runtime programmatically. For example, you can set the gain code of a device by setting the property of the ActiveDAQ's analog input control.

```
Adain1.OverallInputRange = 1
```

2.2.3 Using ActiveDAQ Controls' Methods

To call a method of an ActiveDAQ control, add the name of the method after the name of the control. For example, you can call the RawInput method of ActiveDAQ's analog input control

```
Reading = Adain1.RawInput(channel)
```

2.2.4 ActiveDAQ Controls' Event Routines

The ActiveDAQ controls generate events in response to some occurrence in the controls. To develop the event's routine code, double-click on the control to open the code editor. Then select the desired event and write code for it. The following example is the event routine generated for ActiveDAQ Analog Input Control.

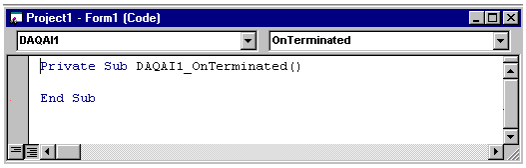


Figure 2-3: Event routine generated by ActiveDAQ Analog Input Control

2.2.5 Using the Object Browser

The VB Object Browser assists you to create Visual Basic code. It can display a simple description for the available properties, methods and events of ActiveDAQ controls. To open the Object Browser, select **Object Browser** from the **View** menu. The result is in the figure below.

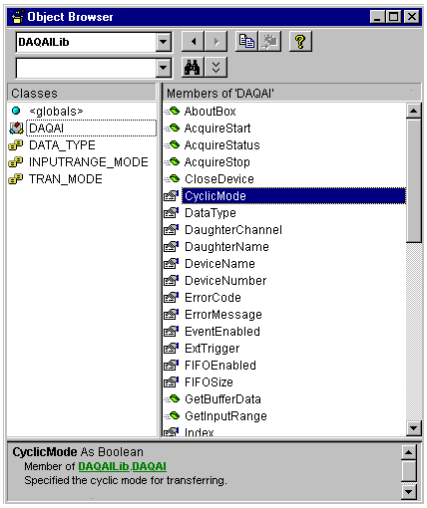


Figure 2-4: ActiveDAQ in the Visual Basic Object Browser

2.3 Developing Delphi Applications

To use ActiveDAQ controls, complete the following procedure:

2.3.1 Loading ActiveDAQ into the Component Palette

Before using ActiveDAQ controls to build an application, you must add them to the Component Palette. Follow the procedure below to add them:

1. Select Import ActiveX Control… from the Component menu
2. Scroll down to the ActiveDAQ controls, which you can find in the Import ActiveX Controls list.
3. Select the controls you want to use in your project, and click Install…. If ActiveDAQ controls are not in the list, press the **Add** button to select the desired controls from the \WINDOWS\SYSTEM(32) directory.

The **Import ActiveX Control** list dialog box is as below:

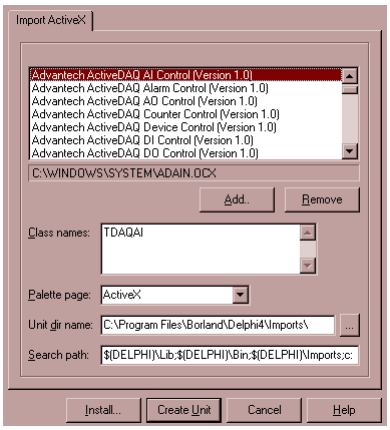


Figure 2-5: Delphi's Import ActiveX Control dialog box showing ActiveDAQ controls

4. Click the **Install** button to add the controls into your program.

2.3.2 Using the Object Inspector

After you add the ActiveDAQ controls to the Delphi Component Palette, select the corresponding icon in the Component Palette and place it on a Delphi form. You can then edit or configure its properties in the Delphi Object Inspector. To access it, select a control and select **Properties Window** from the **View** menu. For example, the property sheet of ActiveDAQ's Analog Input Control is in the figure below.

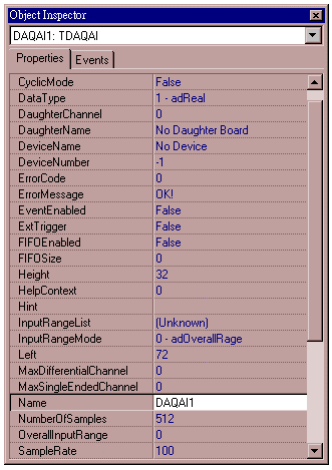


Figure 2-6: ActiveDAQ Analog Input Control property sheet in Delphi

In addition, you also can read and set the properties of ActiveDAQ controls at runtime programmatically. For example, you can set the gain code of a device by setting the property of the ActiveDAQ's analog input control.

```
Adain1.OverallInputRange := 1;
```

2.3.3 Using ActiveDAQ Controls' Methods

To call a method of an ActiveDAQ control, add the name of the method after the name of the control. For example, you can call the RawInput method of ActiveDAQ's analog input control:

```
Reading := Adain1.RawInput(channel);
```

2.3.4 Developing ActiveDAQ Controls' Event Routines

The ActiveDAQ controls generate the events in response to some occurrence in the controls. To develop the event routine code, select **Object Inspector** from the **View** menu, then select the **Event** tab. It is shown below

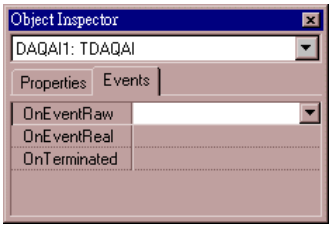


Figure 2-7: The Object Inspector's View Tab

In the Event tab, select the desired event and double click on the empty field next to the event name. Delphi generates the event handler routine in the code window that allows you write code for it. The following example is the event routine generated for ActiveDAQ's Analog Input Control.

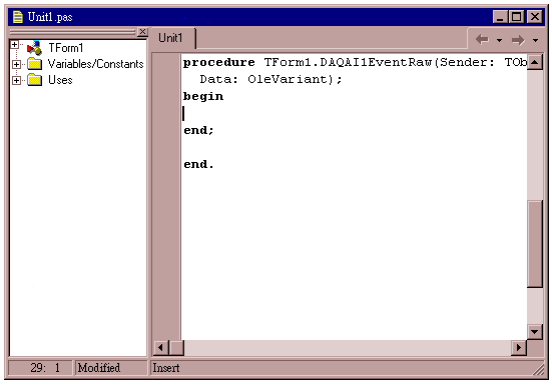


Figure 2-8: Analog Input Control Event Routine

2.4 Developing Visual C++ Applications

To create a Visual C++ application working with ActiveDAQ controls, you can use the Visual C++ MFC AppWizard to create a skeleton project and program. After you create a project with ActiveX control support, you can follow the procedure to develop an application using ActiveDAQ controls. The following procedure uses a dialog-based application to illustrate the procedure.

2.4.1 Loading ActiveDAQ into the Controls Toolbar

Before using ActiveDAQ controls to build an application, you must add them to the Controls Toolbar. You can follow the procedure to add them:

1. Open the workspace window by selecting **Workspace** from the **View** menu.
2. Select the Resource View tab from the button of the Workspace window.
3. Double-click on one dialog entry in the resource tree.
4. Select **Add to Project… -> Components and Controls** from the **Project** menu, and double click on Registered ActiveX Controls.

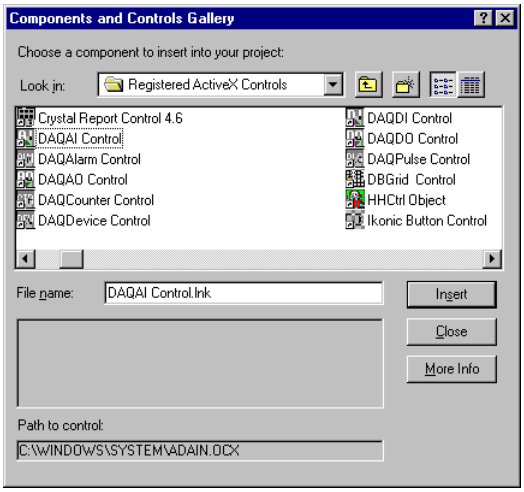


Figure 2-9: Adding ActiveDAQ Controls to a VC++ Project

5. Select one registered ActiveX controls to insert into the Controls Toolbar and your project.

2.4.2 Configuring ActiveDAQ Controls' Properties

After you add the ActiveDAQ controls to the Visual C++ Controls Toolbar, select the corresponding icon from the toolbox and place it on the dialog form as below.

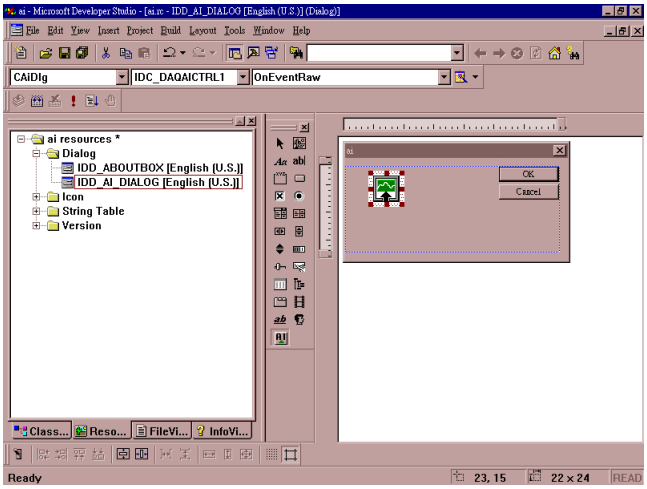


Figure 2-10: Placing an ActiveDAQ on a dialog form

You can then edit or configure its properties in the property sheet. To access it, right click on it and select **Properties**. For example, the property sheet of ActiveDAQ's Analog Input Control is shown in the figure below.

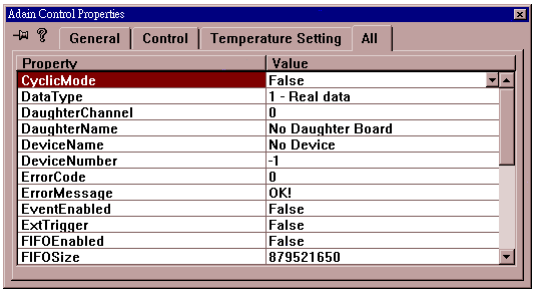


Figure 2-11: Analog Input Control Property Sheet

2.4.3 Programming with ActiveDAQ Controls

Before you can use the properties or methods of a control, assign a member variable name to the control. You can use the **ClassWizard** to add a member variable and map it to the control.

Unlike Visual Basic or Delphi, you don't read and set the properties directly in Visual C++. You must instead use the functions of the wrapper class, created by Visual C++, to map to the control, to read and set the properties. These functions are named starting with Get or Set followed by the name of the properties. For example, to set the *OverallInputRange* property of the ActiveDAQ's analog input control, use *SetOverallInputRange* function of the wrapper class. You can then view the available property functions for a control by clicking on the **ClassView** tab from the button of Workspace window.

You can also use the functions of the wrapper class to access the methods of a control. To call a method, append the method name of the control to the member variable name mapped to the control and pass the appropriate parameters.

2.4.4 Developing ActiveDAQ Controls Event Routines

The ActiveDAQ controls generate the events in response to some occurrence in the controls. To develop the event routine code, right-click on a control and select ClassWizard. Then select the **Message Maps** tab. In the **Message Maps** tab, select the desired control in the ObjectIDs field and select the desired event from the **Messages** field. At last press the **Add Function…** button to generate the event handler routine and press the Edit Code button to edit the event handler routine.

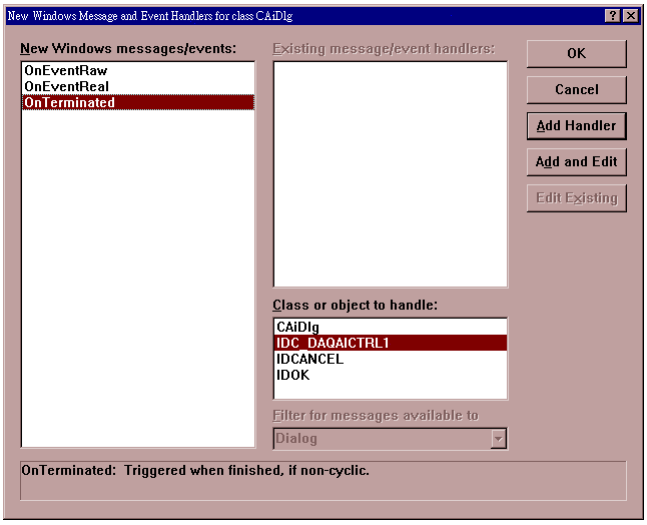


Figure 2-12: ActiveDAQ Control Event Handler Routine

2.5 Compatibility With Visual Basic, Delphi and Visual C++

The ActiveDAQ controls version 1.0 is compatible with the following languages:

- Microsoft Visual Basic for Windows 95/98/NT versions 5.0 and 6.0
- Inprise Delphi for Windows 95/98/NT version 4.0 with Service Pack 3
- Microsoft Visual C++ for Windows 95/98/NT version 5.0 and 6.0

If you are not using one of these development tools, consult your development tool reference manual for details on creating applications with ActiveX controls

CHAPTER 3

Tutorial

3.1 ActiveDAQ Introductory Tutorial

This chapter provides an example to demonstrate how to build an application using ActiveDAQ controls from scratch. The example makes use of the Analog Input Control to scan the values from a specified channel. Visual Basic, Delphi and Visual C++ are used to build the application and demonstrate the step-by-step procedure. For information about using other controls or other developer tools, please refer to Chapter 2 *Building ActiveDAQ Controls' Applications with Various Languages* and Appendix A *Properties, Methods and Events Reference*.

The sample reads an analog input channel from a virtual device and displays the result on the screen. The Advantech DLL driver supports the virtual device named demo board, whose first channel generates a simulated sine wave. By following this example, you will get an overall view about how to program using ActiveDAQ controls.

This chapter assumes that you are familiar with the basic concepts of using Visual Basic, Delphi and Visual C++.

3.2 ActiveDAQ Tutorial for Visual Basic Applications

3.2.1 Step 1: Add Demo Board With DEVINST.EXE

1. Go into the **Start** menu and click on the Device Installation icon in the Advantech ActiveDAQ folder.

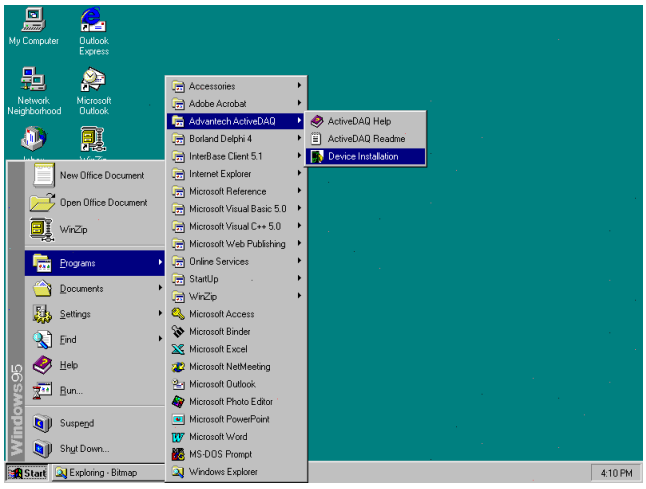


Figure 3-1: Start DEVINST.EXE

2. Then it will launch the Device Installation Program as below.

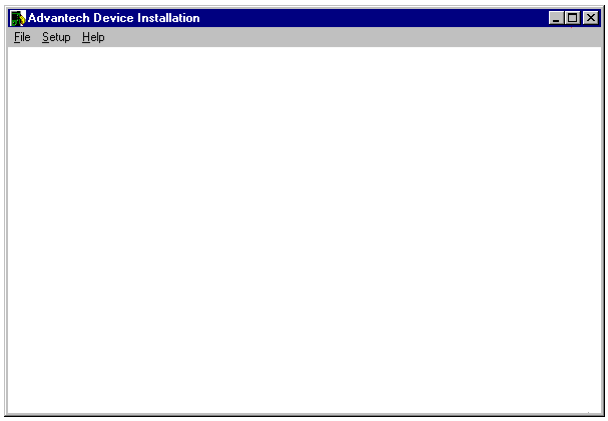


Figure 3-2: Device Installation Utility interface

3. Click on the **Device..** from the **Setup** menu. A dialog box is displayed:

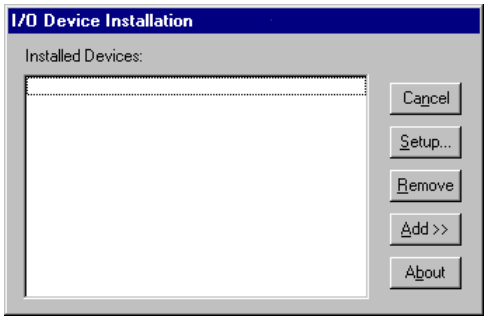


Figure 3-3: Device Installation Utility Setup window

4. Press the **Add>>** button and select Advantech DEMO Board item in the **List of Devices** field.

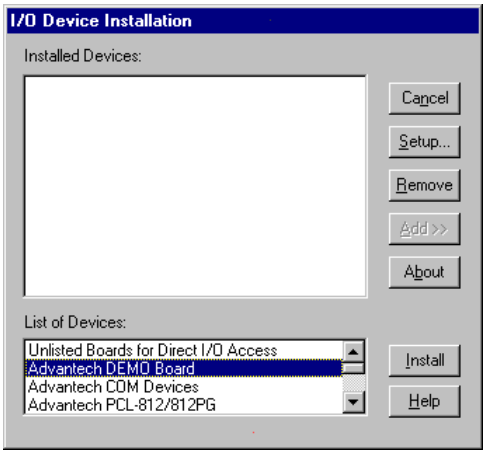


Figure 3-4: Select the DEMO board from the Setup window

5. Press the **Install** button, and a configuration dialog box is displayed as below:

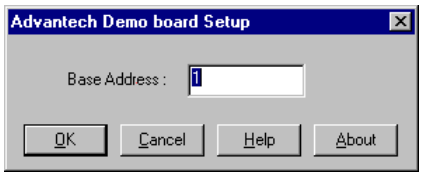


Figure 3-5: Demo board setup configuration window

6. Use the default value and press the **OK** button. You will see a new entry in the **Installed Devices** list.

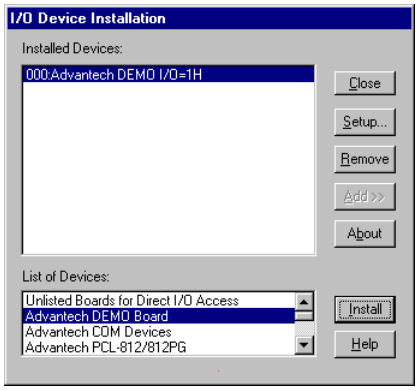


Figure 3-6: Device Installation Utility Installed Devices window

7. Press the **Close** button and exit the Device Installation Utility.

3.2.2 Step 2: Load ActiveDAQ Controls into VB Toolbox

1. Go into the **Start** menu and click on the Visual Basic 5.0 icon in the Microsoft Visual Basic 5.0 folder .

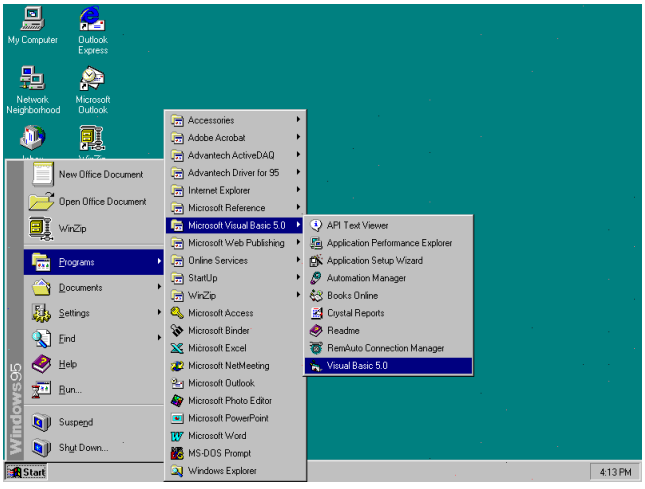


Figure 3-7: Start Visual Basic

2. Then it will launch Visual Basic 5.0 program as below:

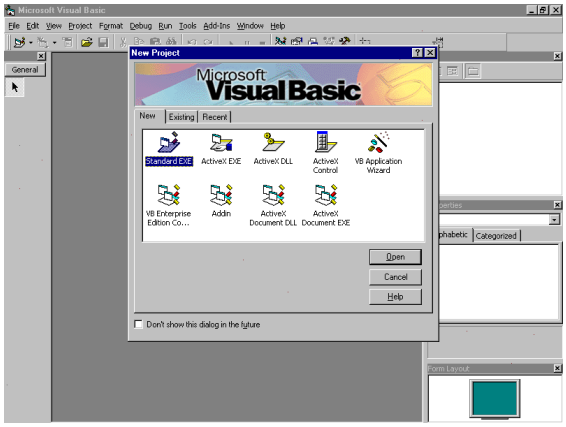


Figure 3-8: The Visual Basic Integrated Development Environment

3. Select Standard EXE icon and press the **Open** button. A new project is created. Click on **Components**… from the **Project** menu. The Components dialog box is loaded as shown below:

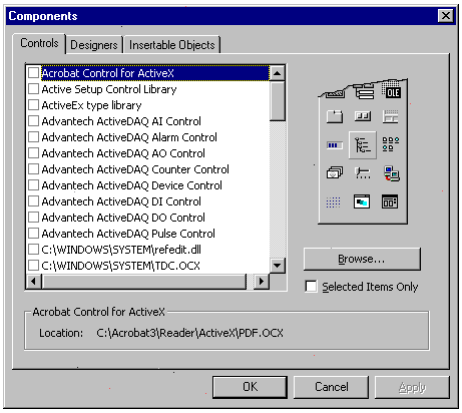


Figure 3-9: The Visual Basic Components dialog box

4. Select the Advantech ActiveDAQ Analog Input Control and Advantech ActiveDAQ Device Control from the list in the **Controls** tab. Click the **Apply** button. Icons that represent these controls will appear in the Visual Basic toolbox as below. Then press the **OK** button to exit the dialog box.

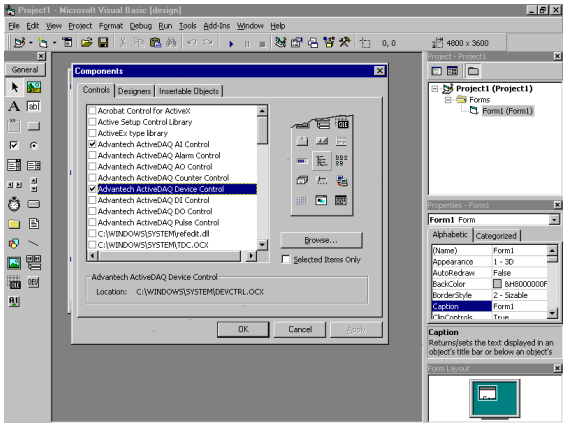


Figure 3-10: Visual Basic toolbox showing Analog Input and Device Controls

3.2.3 Step 3: Design the Form.

1. Place a **DAQDevice** and a **DAQAI** control from the toolbox on the form. Use the default names.
2. Place two **TextBox** controls from the toolbox on the form. Then switch to the Property Window, and enter txtAIValue and txtDeviceName as their **Name** properties.
3. Place two **Label** controls from the toolbox on the form. Enter “Device” and “Analog input” as their **Caption** fields.
4. Place two **CommandButton** controls from the toolbox on the form. Enter cmdSelectDevice, cmdRead as their **Name** properties. Enter Select Device and Read as the **Caption** properties.

Your form should look similar to the one shown below:

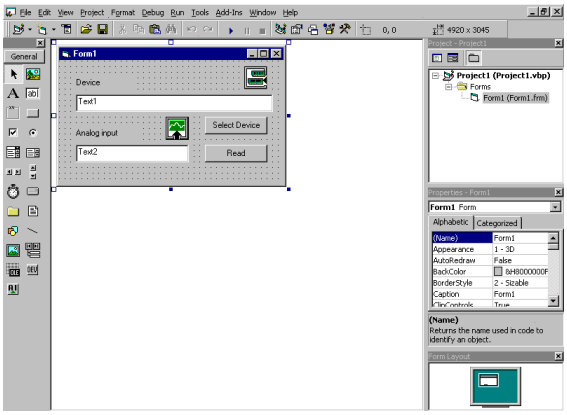


Figure 3-11: Designing the form

3.2.4 Step 4: Configure AI Control in the Property Sheet

1. Click on the **Properties Window** from the **View** menu.
2. Select DAQAI1 from the top of the Property Window
3. Select adReal in the DataType field of the DAQAI1 Property Window.

3.2.5 Step 5: Writing Code for the ActiveDAQ Controls

1. Double-click on the Select Device button on the form, and write the following code:

```
Private Sub cmdSelectDevice_Click()
    DAQDevice1.SelectDevice
    txtDeviceName.Text = DAQDevice1.DeviceName
End Sub
```

2. Double-click on the Read button on the form, and enter the following code:

```
Private Sub cmdAcquire_Click()
    DAQAI1.DeviceNumber = DAQDevice1.DeviceNumber
    DAQAI1.OpenDevice
    txtAIValue = DAQAI1.RealInput(0)
    DAQAI1.CloseDevice
End Sub
```

3.2.6 Step 6: Test Your Program

1. Press F5 to run the program The startup screen is shown below:

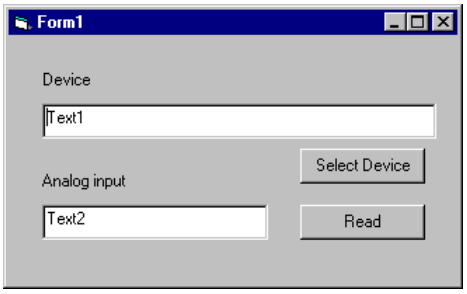


Figure 3-12: Press F5 key to run your program

2. Press the Select Device button on the form. A dialog box is displayed as follows:

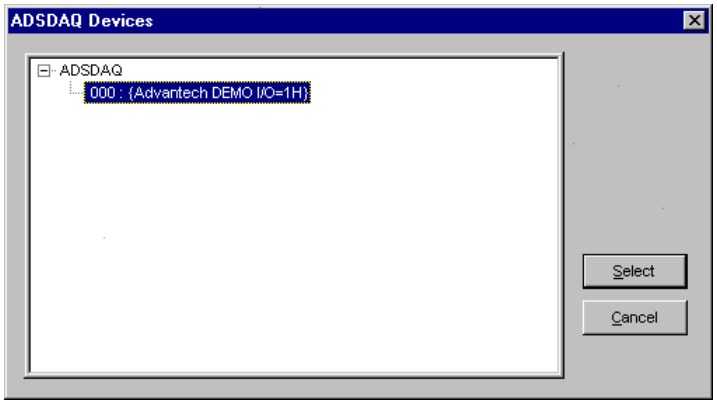


Figure 3-13: Press Select Device button on the form

3. Select 000:{Advantech DEMO I/O=1H} item and press the **Select** button. The selected device is shown in the Device field. Then press the Read button, the data will appear as follows.

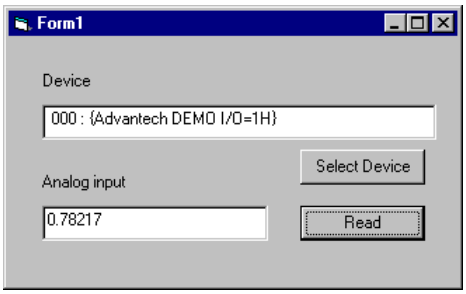


Figure 3-14: Running the example

3.3 ActiveDAQ Tutorial for Delphi Applications

3.3.1 Step 1: Add Demo Board With DEVINST.EXE

1. The same as step 1 for making a Visual Basic application. Please see step 1 on page 30.

3.3.2 Step 2: Load ActiveDAQ Controls into Delphi Component Palette

1. Go into the **Start** menu and click on the Delphi 4 icon in the Borland Delphi 4 folder.

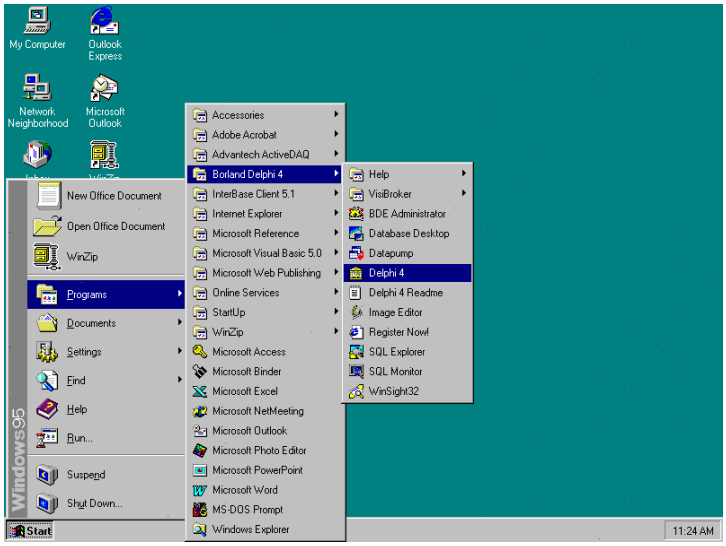


Figure 3-15: Starting Inprise Delphi 4

2. Delphi 4 will launch as shown below:

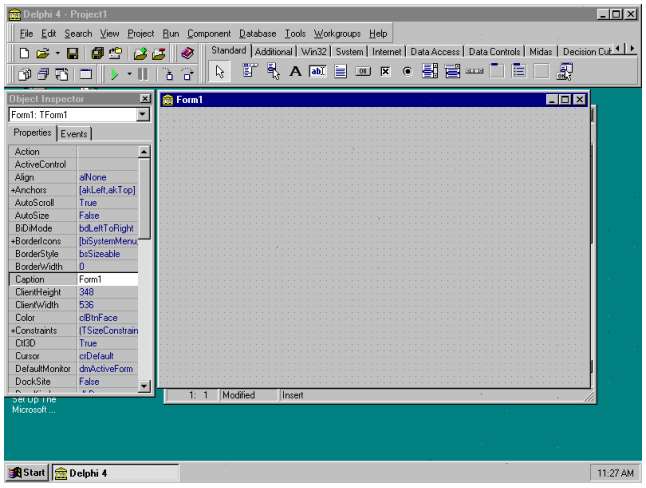


Figure 3-16: Delphi 4 main program

3. Select **Import ActiveX Control…** from the **Component** menu. The Import ActiveX Control list dialog box loads:

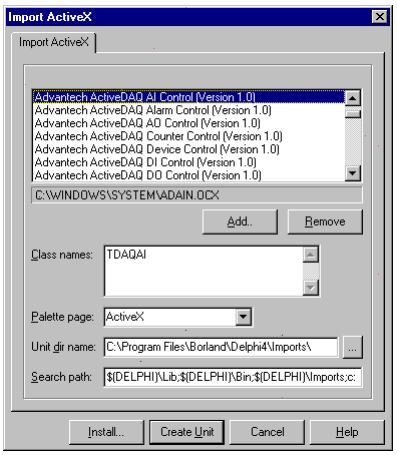


Figure 3-17: Delphi's Import ActiveX Control dialog box

4. Scroll down to the Advantech ActiveDAQ AI Control and click the **Install…** button. A dialog box is displayed as follows.

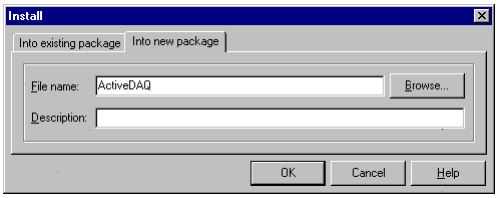


Figure 3-18: Installing the ActiveDAQ AI control

5. Enter ActiveDAQ into the File name field under the *Into new package* tab, and press **OK**. Then scroll down to the Advantech ActiveDAQ Device Control in the Import ActiveX dialog box.

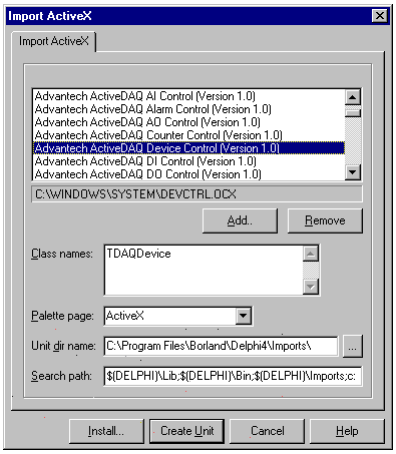


Figure 3-19: Importing an ActiveDAQ control into Delphi

6. Press the **Install** button. A dialog box is displayed as below. Click the **OK** button.

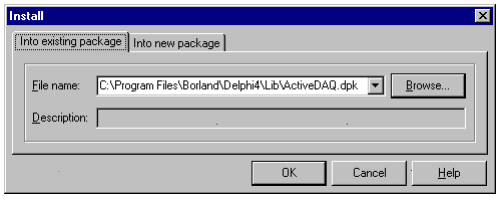


Figure 3-20: Installing the ActiveDAQ control into Delphi

7. The ActiveDAQ AI and ActiveDAQ Device controls are loaded into the Component Palette. You can check it by clicking on **Install Package…** from the **Component** menu. A dialog box is shown as below.

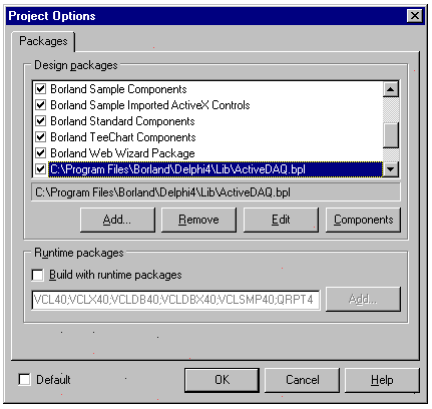


Figure 3-21: Component Palette showing loaded ActiveDAQ control

8. Scroll down to the ActiveDAQ.bpl item in the list and press the **Components** button. A dialog box is shown as below. There are two controls in the Installed components list: the AI and Device controls.

Figure 3-22: Components dialog box showing two ActiveDAQ controls

3.3.3 Step 3: Design the form

1. Switch to the form and select the ActiveX tab from the Component Palette.

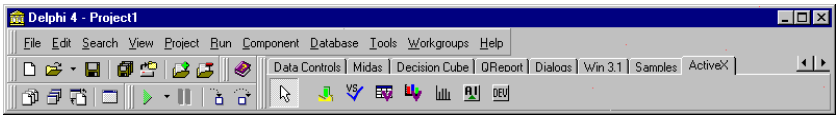


Figure 3-23: The ActiveX tab in the Component Palette

2. Place a DAQDevice control and a DAQAI control from the Component Palette on the form. Use the default names, DAQAI1 and DAQDevice1.
3. Switch to the Standard tab from the Component Palette and place two Edit controls on the form. Then enter txtAIValue and txtDeviceName as their Name properties in the Object Inspector.
4. Place two Label controls under the Standard tab on the form. Enter Device and Analog Input as their Caption fields in the Object Inspector.
5. Place two Button controls on the form. Enter cmdSelectDevice, cmdRead as their Name properties, and Select Device and Read as the Caption properties in the Object Inspector.

Your form should look similar to the one shown below:

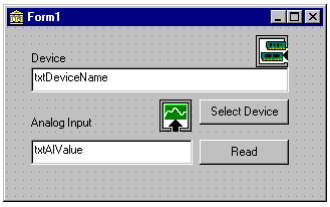


Figure 3-24: Form design for the Delphi DAQDevice/DAQAI example

3.3.4 Step 4: Configure the AI Control in the Object Inspector

1. Click on the DAQAI1 control on the form. Its properties are shown in the Object Inspector window. Select 1 – adReal in the DataType field.

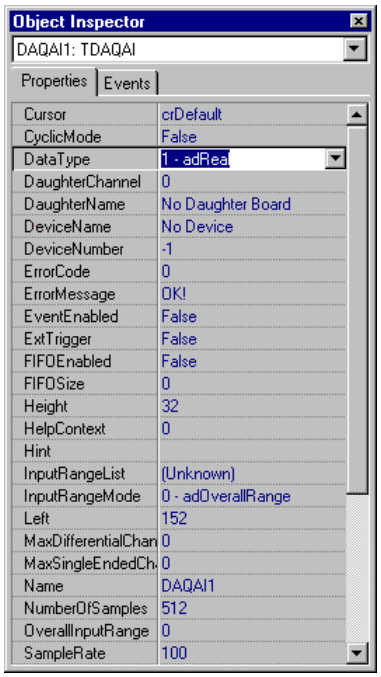


Figure 3-25: Object Inspector showing DAQAI1 properties

3.3.5 Step 5: Writing Code for the ActiveDAQ Controls

1. Go back to the form and double-click on the Select Device button. Delphi generates the routine skeleton in the code window and you write the following code:

```
procedure TForm1.cmdSelectDeviceClick(Sender: TObject;
begin
   DAQDevice1.SelectDevice;
   txtDeviceName.Text := DAQDevice1.DeviceName;
end;
```

2. Double-click on the Read button on the form, and enter the following code:

```
procedure TForm1.cmdReadClick(Sender: TObject);
begin
   DAQAI1.DeviceNumber := DAQDevice1.DeviceNumber;
   DAQAI1.OpenDevice;
   TxtAIValue.Text := FloatToStr(DAQAI1.RealInput(0));
   DAQAI1.CloseDevice;
end;
```

3.3.6 Step 6: Test Your Program

1. Press F9 to run the program. The startup screen is shown as below:

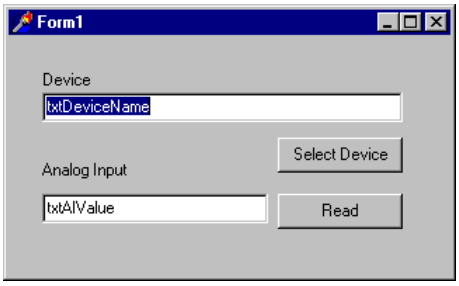


Figure 3-26: Example program startup screen

2. Press the **Select Device** button on the form. A dialog box is displayed as shown in the following figure:

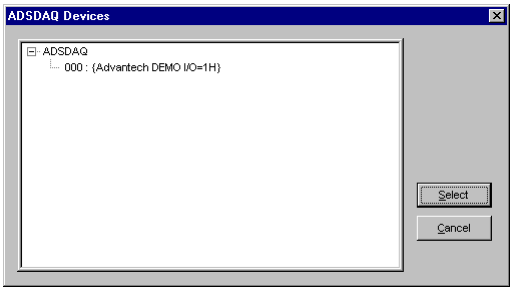


Figure 3-27: Delphi example dialog window

3. Select the 000:{Advantech DEMO I/O=1H} item and press the Select button. The selected device is shown in the Device field. When you press the Read button, the data will appear as shown below.

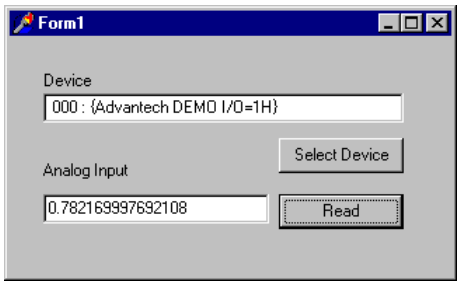


Figure 3-28: Delphi example dialog window

3.4 ActiveDAQ Tutorial for Visual C++ Applications

3.4.1 Step 1: Add Demo Board With DEVINST.EXE

1. The same as step 1 in the Visual Basic example. Please see page 30.

3.4.2 Step 2: Load ActiveDAQ Controls into the VC Controls Toolbar

1. Click on the Microsoft Visual C++ 5.0 icon in the Microsoft Visual C++ 5.0 folder in the Windows Start menu.

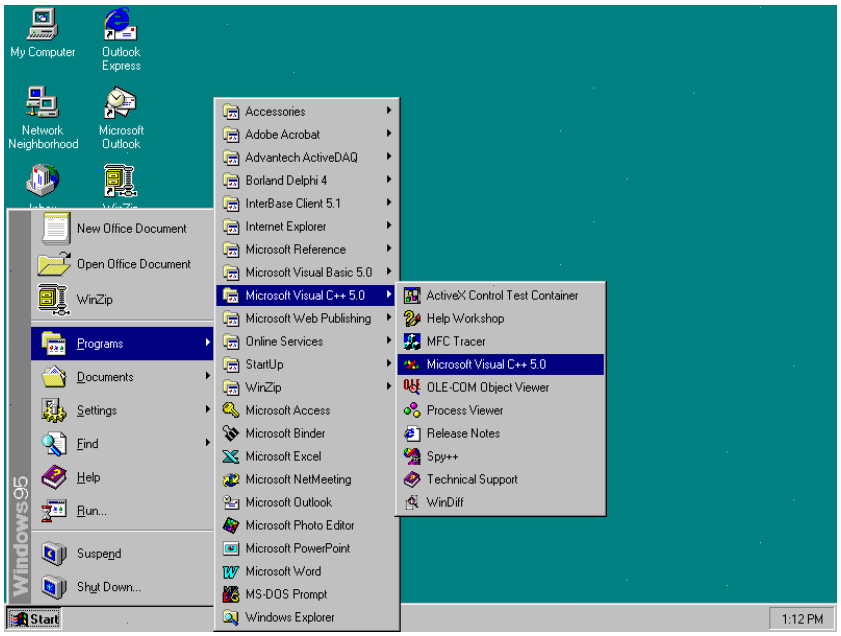


Figure 3-29: Starting Microsoft Visual C++

2. The Visual C++ program will launch as shown below:

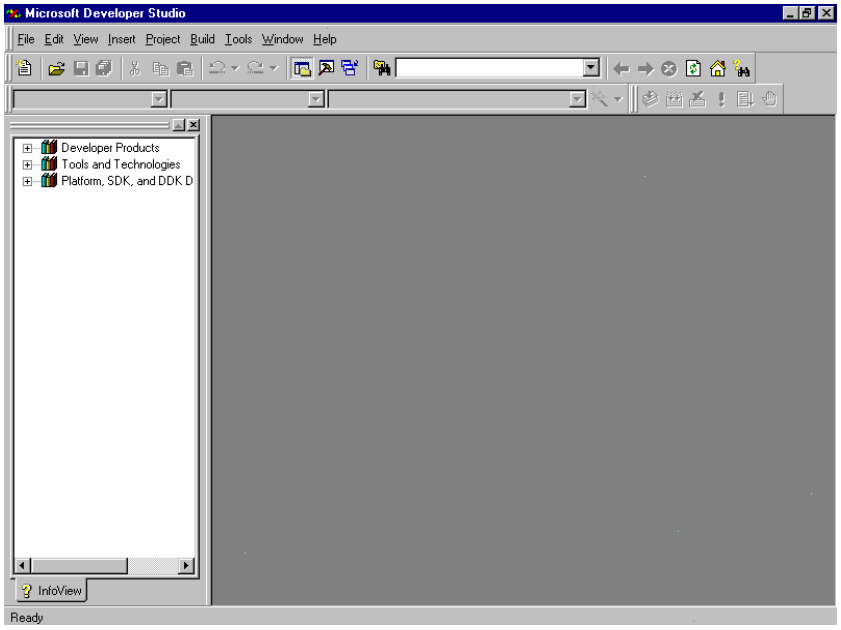


Figure 3-30: The main screen of the Microsoft Visual C++ IDE

3. Select **New…** from the **File** menu. A dialog box is shown as below:

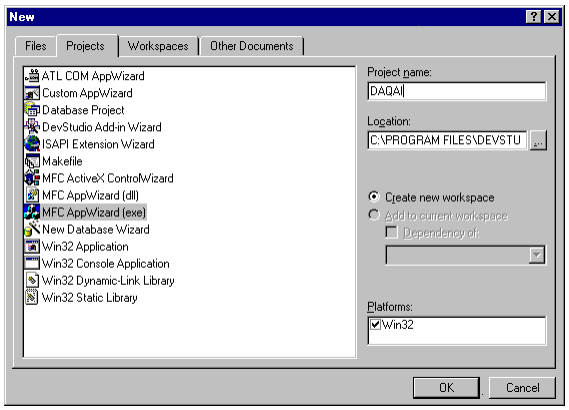


Figure 3-31: Click File | New form the Visual C++ main menu

4. Click on the MFC AppWizard (exe) entry in the list and enter DAQAI in the Project name field. Then press the **OK** button. A dialog box is displayed as below:

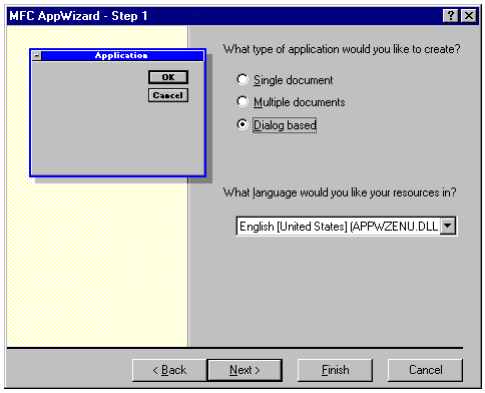


Figure 3-32: The Visual C++ MFC AppWizard

5. Select the *Dialog-based* entry and press the **Next>** button. The *MFC AppWizard – Step 2 of 4* dialog box is shown as below. Leave the default values and press **Finish** button.

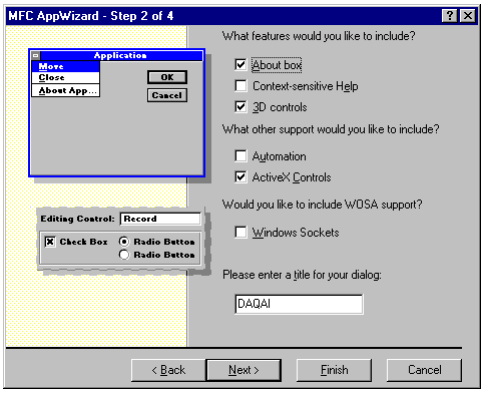


Figure 3-33: The Visual C++ MFC AppWizard

6. After completing the MFC Wizard, some skeleton code and class are created.

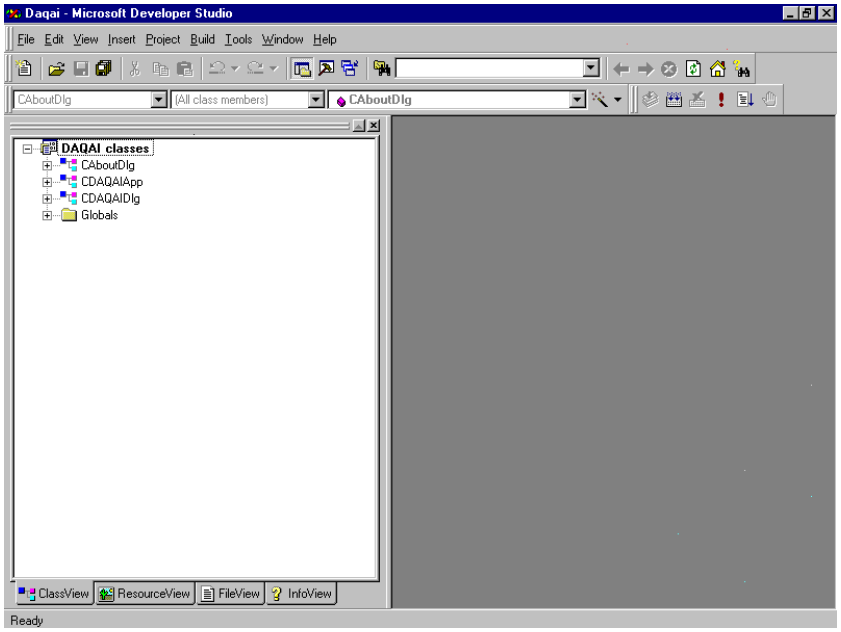


Figure 3-34: The skeleton program after running the Visual C++ MFC AppWizard

7. Click on the *Resource View* tab from the Workspace, and expand the DAQAI resources as below:

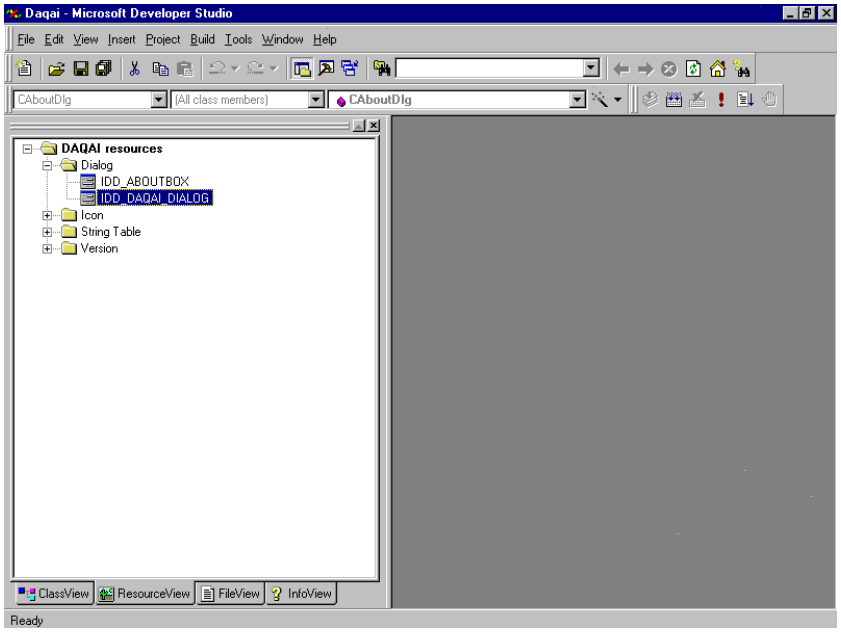


Figure 3-35: Viewing the DAQAI resources in the workspace

8. Double-click on the IDD_DAQAI_DIALOG entry in the resource tree. A dialog-based form is displayed as below:

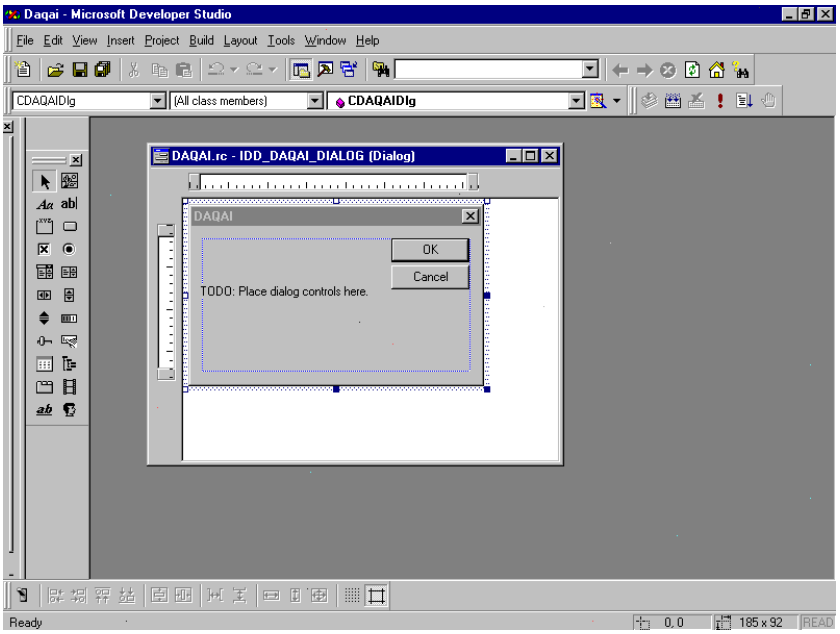


Figure 3-36: Double-click the IDD_DAQAI_DIALOG entry in the resource tree

9. Select **Add to Project…**-> **Components and Controls** from the **Project** menu, and double-click on *Registered ActiveX Controls.* The result should be as below:

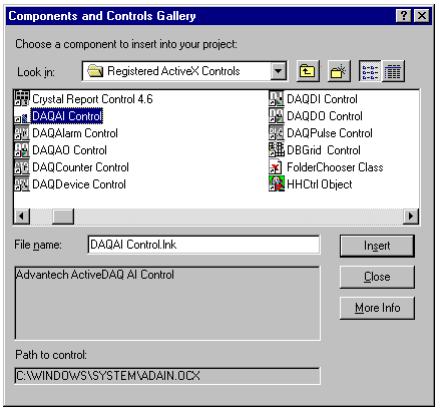


Figure 3-37: VC++ Components and Controls Gallery dialog box

10. Scroll down to the DAQAI Control and press the **Insert** button in the *Components and Controls Gallery*. Again, scroll down to the DAQDevice Control and press the **Insert** button.

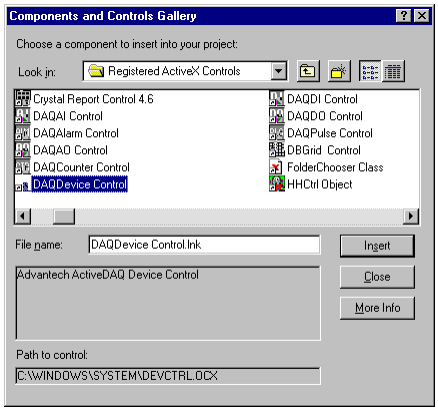


Figure 3-38: Inserting the DAQDevice control into your project

11. The VC++ toolbar will show the the DAQAI and DAQDevice controls in your Visual C++ application and toolbar:

Figure 3-39: DAQAI and DAQDevice controls loaded into VC++ application

3.4.3 Step 3: Design the form

1. Place a DAQDevice control and a DAQAI control from the Controls toolbar on the dialog-based form.
2. Place two Edit Box controls from the Controls toolbar on the form. Then right-click on them on the form and select the Properties, and enter IDC_DEVICENAME and IDC_AIVALUE in their ID fields:

Figure 3-40: Entering the ID field values in the Properties window

3. Place two Static Text controls from the Controls toolbar on the form. Then right-click on them and select **Properties**. Then enter *Device* and *Analog input* in their Caption fields.
4. Place two **Button** controls on the form. Then right-click on them and select **Properties**. Then enter Select Device and Read in the Caption fields, and enter IDC_SELECTDEVICE and IDC_READ in the ID fields.

Figure 3-41: Entering the properties for the button controls

Your form should look similar to the one shown below:

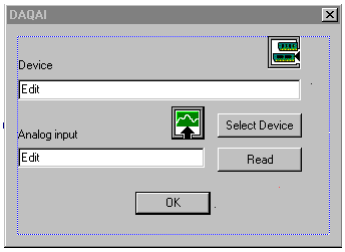


Figure 3-42: Form design in the Visual C++ example

3.4.4 Step 4: Configure the AI Control's Properties

1. Right-click on the DAQAI control on the form and select **Properties**. Select 1 – adReal in the DataType field.

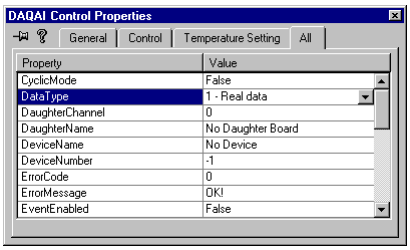


Figure 3-43: Configuring the properties of the DAQAI control

3.4.5 Step 5: Writing Code for the ActiveDAQ Controls

1. Right-click on the IDC_DEVICENAME edit box control on the form and select **ClassWizard…**. A dialog box is displayed, then select the *Member Variables* tab in the dialog box. The result is shown as below:

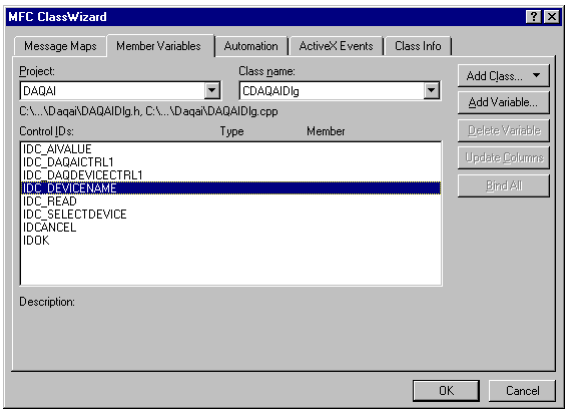


Figure 3-44: The Class Wizard dialog box (Member Variables tab)

2. Press the **Add Variable…** button and a dialog box is shown as below:

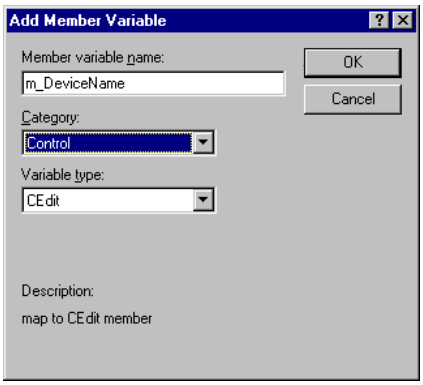


Figure 3-45: The Class Wizard dialog box (Member Variables tab)

3. Enter m_DeviceName in the Member variable name field, select Control in the Category field, and select CEdit in the Variable type field. Then press the **OK** button. The new variable in shown as below.

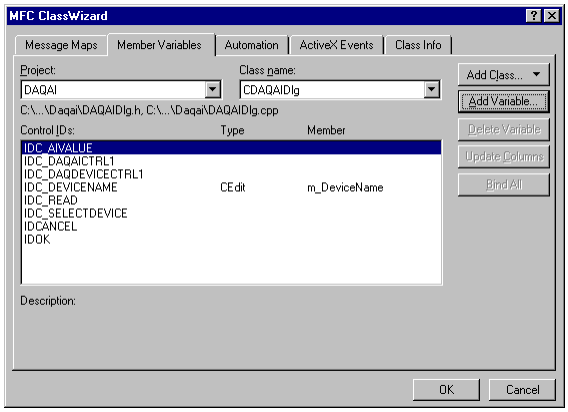


Figure 3-46: Defining a new member variable

4. Follow the previous steps to add the following variables for IDC_AIVALUE, IDC_DAQAICTRL1 and IDC_DAQDEVICECTRL1 entries: m_AIValue (type:CEdit), m_AICtrl (type:CDAQAI), and m_DeviceCtrl (type:CADDEVCtrl). The result is shown as below:

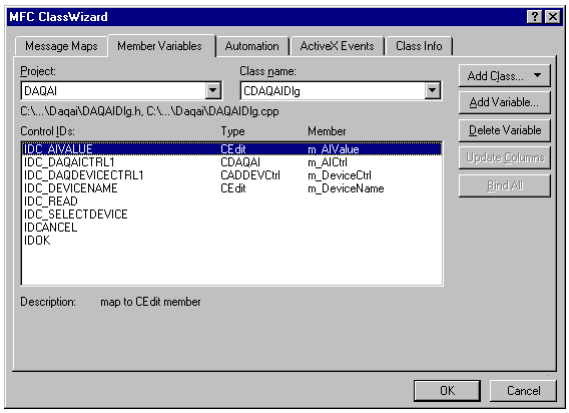


Figure 3-47: The result after configuring the member variables

5. Right-click on the Select Device button on the form, and select **Events…**. A dialog box is shown as below:

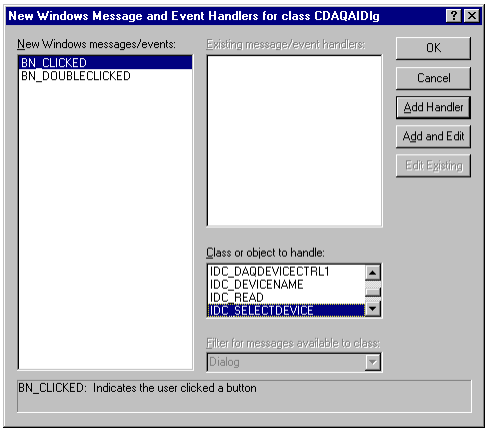


Figure 3-48: VC++ Windows Message and Event Handlers configuration

6. Select IDC_SELECTDEVICE entry in the Class or object to handle field, and select the BN_CLICKED entry in the New Windows messages/events field. Then press **Add and Edit** button to edit code. Write the following code as below:

```
void CDAQAIDlg::OnSelectdevice()
{
   // TODO: Add your control notification
   // handler code here
   m_DeviceCtrl.SelectDevice();
   m_DeviceName.SetWindowText(m_DeviceCtrl.GetDeviceName());
}
```

7. Right-click on the **Select Device** button on the form, and select **Events…**. A dialog box is shown as below:

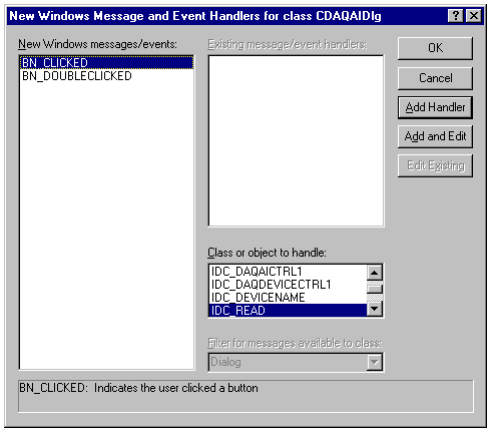


Figure 3-49: VC++ Windows Message and Event Handlers configuration

8. Select IDC_READ entry in the Class or object to handle field, and select the BN_CLICKED entry in the New Windows messages/events field. Then press the **Add and Edit** button to edit code. Write the following code:

```
void CDAQAIDlg::OnRead()
{
   // TODO: Add your control notification handler
code here
   char buffer[50];
   CString value;

  m_AICtrl.SetDeviceNumber(m_DeviceCtrl.GetDeviceNumber());
   m_AICtrl.OpenDevice();
   _gcvt(m_AICtrl.RealInput(0), 7, buffer);
   value = buffer;
   m_AIValue.SetWindowText(value);
   m_AICtrl.CloseDevice();
}
```

3.4.6 Step 6: Testing Your Program

1. Press Ctrl+F5 to execute the program. The startup screen is shown below:

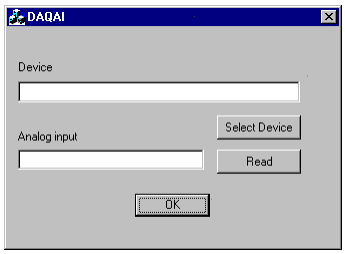


Figure 3-50: Running the example

2. Press the **Select Device** button on the form. A dialog box is displayed as follows:

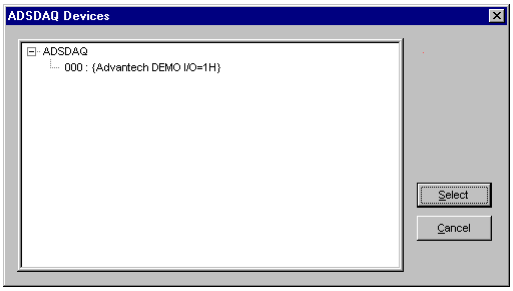


Figure 3-51: Running the example

3. Select 000:{Advantech DEMO I/O=1H} item and press the Select button. The selected device is shown in the Device field. Then press the Read button, the data will appear as below:

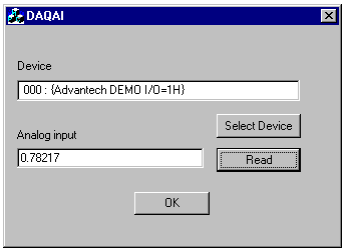


Figure 3-52: Running the example

CHAPTER 4

Using ActiveDAQ Controls

4.1 Using ActiveDAQ Controls

This chapter describes how can you use the ActiveDAQ controls to perform input or output operations with Advantech hardware. It explains each control and their most commonly used properties, methods and events. The following lists the available controls:

4.1.1 ActiveDAQ Controls and their Operations

- **DAQDevice**: Generates a dialog box to let user select a device for I/O operations
- **DAQAI**: Performs analog input or temperature measurement
- **DAQAO**: Performs analog output
- **DAQDI**: Performs digital input
- **DAQDO**: Performs digital output
- **DAQCounter**: Performs event counting or frequency measurement
- **DAQPulse**: Performs pulse output operations
- **DAQAlarm**: Performs alarm checking and monitoring

Each ActiveDAQ control contains properties, methods and events, based on the operation type and on the capabilities of the I/O device that ActiveDAQ supports. Some of the properties and methods for each control are common to all ActiveDAQ controls, such as *DeviceNumber*, *DeviceName*, *OpenDevice*, *CloseDevice*, *ErrorCode* and *ErrorMessage*. Other properties are specific to the control and the type of operations that it supports.

4.2 Common Properties and Methods

4.2.1 *DeviceNumber* and *DeviceName* Properties

Each control has a *DeviceNumber* property which specifies the device that you want to perform the I/O operations. The *DeviceNumber* is initially defined through configuration using the Device Installation Utility, DEVINST.EXE. After the *DeviceNumber* is selected, the corresponding device name is returned in the property *DeviceName*.

The following is the configuration dialog box of the Device Installation Utility. It lists the installed devices. For the third entry of the device, “002:PCL-818L I/0=300H”, the *DeviceNumber* is equal to 002 and the *DeviceName* is “002:PCL-818L I/O=300H”.

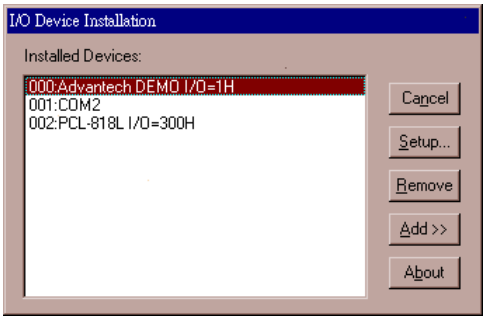


Figure 4-1: DEVINST.EXE showing DeviceName *and* DeviceNumber

You can assign the *DeviceNumber* property of each control directly. Alternatively, you can use an individual control DAQDevice to generate a dialog box for selecting the desired device, and return the selected device in the *DeviceNumber* and *DeviceName* properties. You can pass the results to the other controls. The following is a Visual Basic example. It uses the DAQDevice control to select a device, and assigns the result to the DAQAI control for analog input operation.

```
Private Sub cmdSelectDevice_Click()
    DAQDevice1.SelectDevice
    txtDeviceNum.Text = DAQDevice1.DeviceNumber
    txtDeviceName.Text = DAQDevice1.DeviceName
    DAQAI1.DeviceNumber = DAQDevice1.DeviceNumber
    DAQAI1.DeviceName = DAQDevice1.DeviceName
End Sub
```

The dialog box created by the DAQDevice control is as follows:

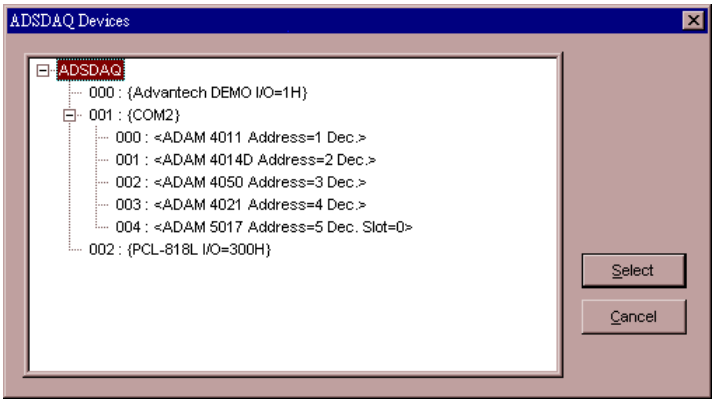


Figure 4-2: Dialog box created by the DAQDevice control

A device may contain sub-devices. If you want to view the sub-devices in the dialog box, you have to set the *GetModuleList* property of the DAQDevice control to TRUE. In the above dialog box, it displays the ADAM modules (sub-devices) for a serial port device with *GetModuleList* equal to TRUE.

4.2.2 *OpenDevice* and *CloseDevice* Methods

OpenDevice is a common method that initializes the device previously specified by the *DeviceNumber* property. This method must be called before any other methods that perform I/O operations. *CloseDevice* is the counterpart method of the *OpenDevice* method to close the device.

Note: The OpenDevice and CloseDevice methods are not supported for the DAQDevice control.

4.2.3 *ErrorCode* and *ErrorMessage* Properties

In addition to the *DeviceNumber* and *DeviceName* properties and the *OpenDevice* and *CloseDevice* methods, the *ErrorCode* is another common property. It is used for storing the result of calling any method. If the error code is equal to zero, the operation is completed normally. If the value is non-zero, the corresponding error message is returned in the *ErrorMessage* property. Appendix B lists the possible error codes and error messages.

4.3 Analog Input Control

The Analog Input Control (DAQAI) is used to perform analog input operation or temperature measurement. It can acquire single data, waveform data with specified sampling rate or temperature reading.

To acquire single data and waveform data from data acquisition and control (DA & C) cards, you may specify the input range properties *OverallInputRange* or *InputRangeList*. If all channels have the same input range, you can set the property *OverallInputRange* to an appropriate value and set the input range mode *InputRangeMode* for overall. The value of *OverallInputRange* property is the corresponding index of the input range list returned from the *GetInputRange* method. You can also use the *GetInputRange* method to get the input range support for the device specified by *DeviceNumber*. To scan multiple channels with different input ranges, you must set the *InputRangeList* property, which is an array type to appropriate values. Then set the input range mode *InputRangeMode* for different input ranges.

4.3.1 Single Data Reading

The Analog Input Control provides two methods to perform single channel reading: *RawInput* and *RealInput*. The *RawInput* method returns binary values, and *ReadInput* returns voltage data. Both of them need a channel parameter to specify the input channel. The syntax in Visual Basic is as below:

```
Voltage = DAQAI1.RealInput(channel)
```

4.3.2 Waveform Data Reading

To acquire waveform data, the Analog Input Control provides single shot (non-cyclic) and continuous acquisitions (cyclic). You can set the *CyclicMode* property of the control to TRUE for continuous acquisition. You can acquire data with the Analog Input Control with one or more channels through the properties *StartChannel* and *StopChannel*. Another property is *DataType* which specifies the returned data type, binary value or voltage data. The *NumberOfSamples* property specifies the number of samples acquired in a single-shot acquisition or

continuous acquisition. The *SampleRate* property means the sampling rate in Hz for one data item. For example, if you want to scan two channels in 1 second, then the *SampleRate* has to be 2 Hz.

The Analog Input Control provides four kinds of waveform data acquisition. They are software triggering, interrupt triggering, DMA triggering and dual DMA triggering. The *TransferMode* property specifies the data acquisition mode. Software triggering data acquisition means sampling the data based on the software timer. Interrupt, DMA and Dual DMA triggering data acquisitions use the on-board pacer to trigger the sampling operation and acknowledge the driver through a hardware interrupt.

To use hardware with FIFO feature (e.g., PCL-818HD), you should set the *FIFOEnabled* property to TRUE and set *TransferMode* for interrupt triggering. According to the count for hardware FIFO interrupt, the *NumberOfSamples* property must be a multiple of the count. For example, PCL-818HD supports half-full FIFO size for the interrupt. This means the *NumberOfSamples* property must be a multiple of half the FIFO size that is kept in the *FIFOSize* property.

Note: For DMA triggering, the NumberOfSamples property must be in excess of 4K samples.

Note: Dual DMA triggering is only supported in the PCL-1800.

The methods to perform waveform data reading are *AcquireStart* and *AcquireStop*. Use the *AcquireStart* method to start the acquisition and the *AcquireStop* method to stop the acquisition.

In contrast to single data reading, the waveform data reading needs an internal buffer to store the sample data. The size of the internal buffer is specified in the *NumberOfSamples* property. You can retrieve the buffer data either manually or by event. If you enable the event method, the control will fire an event when the number of sample data reaches the *NumberOfSamples* property. You can then retrieve the sample data in the event handler routine named *OnEventRaw* or *OnEventReal* according to the *DataType* property. To enable the event method, set the *EventEnabled* property to TRUE. Another event handler routine is *OnTerminated* which is triggered for noncyclic mode and event enabled.

To retrieve the buffer data manually, the Analog Input Control provides the *GetBufferData* method. The *GetBufferData* method accepts the starting buffer location and data count parameters and then returns the buffer data. You can also use the *AcquireStatus* method to get the current buffer location for the next incoming data.

4.3.3 Temperature Measurement

You can also use the Analog Input Control to perform temperature measurement. The properties used include *DaughterChannel*, *ThermoDasChannel*, *ThermoDasGain*, *ThermoType* and *ThermoScale*. The *ThermoDasChannel* property specifies the channel on the card. The *ThermoDasGain* property specifies the gain code on the channel of the card. The *DaughterChannel* property specifies the scanned channel on the daughter board. After you configure the property, use the *ThermoRead* method to read the temperature value. You can assign the *DaughterChannel* property directly or use the *SelectDaughter* method. It generates a dialog box for you to select the *DaughterChannel*. After you select the scanned channel of the daughter board, the *SelectDaugher* method will assign the *DaugherChannel* and *DaugherName* properties automatically.

4.3.4 Example: Waveform Analog Input With Software Triggering

Step 1: Designing the form

1. Open a new project and form.
2. Load the Advantech ActiveDAQ Device Control and Advantech ActiveDAQ AI Control from the **Project | Components** menu.
3. Place a DAQDevice and a DAQAI control on the form. Use the default names.
4. Place two TextBox controls on the form and enter txtValue and txtDeviceName in their Name fields.
5. Place three CommandButton controls on the form. Enter cmdSelectDevice, cmdAcquire and cmdStop for their name properties. Then enter Select Device, Acquire, and Stop for their caption properties.

Your form should look similar to the one shown below :

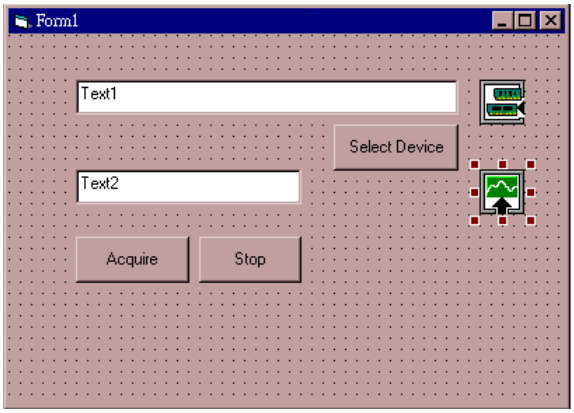


Figure 4-3: The form for the waveform analog input with software triggering example

Step 2: Configuring the properties of the DAQAI1 control

1. Enter 2 in the NumberOfSample field.
2. Select True in the CyclicMode field.
3. Select True in the EventEnabled field.
4. Select adReal in the DataType field.

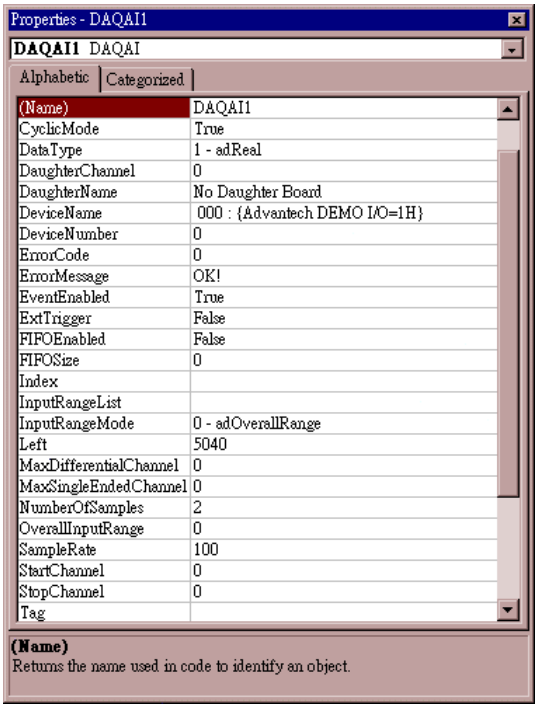


Figure 4-4: Configuring the properties of the DAQAI control

Step 3: Writing the code

1. For the Select Device button, write the following code:

```
Private Sub cmdSelectDevice_Click()
    DAQDevice1.SelectDevice
    txtDeviceName.Text = DAQDevice1.DeviceName
End Sub
```

2. For the Acquire button, write the following code:

```
Private Sub cmdAcquire_Click()
    DAQAI1.DeviceNumber = DAQDevice1.DeviceNumber
    DAQAI1.OpenDevice
    DAQAI1.EventEnabled = True
    DAQAI1.AcquireStart
End Sub
```

3. For the Stop button, write the following code:

```
Private Sub cmdStop_Click()
    DAQAI1.AcquireStop
    DAQAI1.CloseDevice
End Sub
```

4. Write the following code for the DAQAI1_OnEventReal event to display data. The data stores the returned values. The size is equal to the *NumberOfSamples* property.

```
Private Sub DAQAI1_OnEventReal(ByVal Data-
Count As Long, ByVal Data As Variant)
    txtValue.Text = Data(0)
End Sub
```

Step 4: Testing your program

1. Press the Select Device button and choose the demo board.
2. Press the Acquire button. You will view the data in the txtValue field as follows:

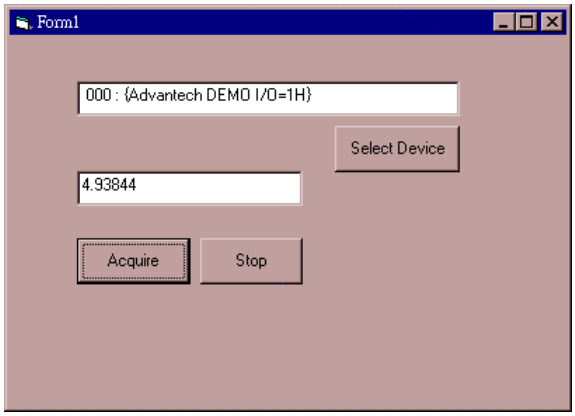


Figure 4-5: The running waveform analog input with software triggering example

3. Press the Stop button to terminate the program.

We provide some examples for the Analog Input Control in the \Examples\ path of the installation directory. These examples cover analog input and temperature measurement. Please refer to them for more information and to learn more about the Analog Input Control's advanced features.

4.4 Analog Output Control

Use the Analog Output control (DAQAO) to perform single point analog output or waveform output. The C*hannel* property specifies which channel is used to perform the operation.

4.4.1 Single Point Analog Output

The *Analog Output Control* provides two methods to perform single point analog output: *RawOutput* and *RealOutput*. The *RawOutput* method outputs a binary value to the channel specified by the *Channel* property, and *RealOutput* outputs a voltage data.

4.4.2 Waveform Analog Output

To perform waveform generation, the Analog Output Control provides finite (non-cyclic) and continuous mode (cyclic) generation. You can set the *CyclicMode* property of the control to TRUE for continuous mode. The *DataType* property specifies the output data type, binary value or real data. The *NumberOfOutputs* property specifies the number of data for output. The *OutputRate* property means the output rate in Hz.

The Analog Output Control provides three kinds of waveform generation: software triggering, interrupt triggering, and DMA triggering. The *TransferMode* property specifies the triggering mode. Software triggering means to output data based on the software timer. Interrupt and DMA triggering use the on-board pacer to trigger the output operation and acknowledge the driver through hardware interrupt.

Note: For DMA triggering, the NumberOfOutputs property must be in excess of 4K.

The methods to perform waveform generation are *OutputStart* and *OutputStop*. Use the *OutputStart* method to start the operation and the *OutputStop* method to stop the operation.

In contrast to single point output, the waveform output needs an internal buffer to store the output data. The buffer size is equal to the

NumberOfOutputs property. You have to set the buffer data before performing the output operation. According to the *DataType* property, raw or real, you have to use the *SetRawBuffer* or *SetRealBuffer* method to set the buffer data.

If you enable the event method, the control will fire an event when the number of output data reaches the *NumberOfOutputs* property. You can then set the next output data in the event handler routine named *OnCompleted*. To enable the event method, set the *EventEnabled* property to TRUE. Another event handler routine is *OnTerminated* that is triggered for non-cyclic mode and event enabled.

In addition to enabling the event to inform the output status, the Analog Output Control also provides the *OutputStatus* method. The *OutputStatus* method returns the current buffer location for the next output data.

4.4.3 Example: Single Analog Output

Step 1: Designing the form

1. Open a new project and form.
2. Load the Advantech ActiveDAQ Device Control and Advantech ActiveDAQ AO Control from the **Project | Components** menu.
3. Place a DAQDevice and a DAQAO control on the form. Use the default names.
4. Place two TextBox on the form and enter txtValue and txtDevice-Name as their Name properties.
5. Place two CommandButton controls on the form. Enter cmdSelect-Device and cmdWrite as their Name properties. Then enter Select Device and Write as their Caption properties.
6. Place two Label controls, and enter Device and Output values as their Caption properties.

Your form should look similar to the one shown below:

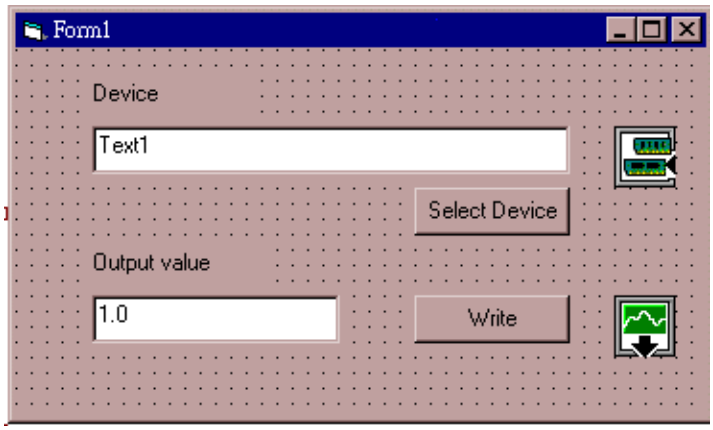


Figure 4-6: Designing the form for the single analog output example

Step 2: Configuring the properties

1. Enter 1.0 in the Text field of the txtValue control in the Properties window.

Step 3: Developing the code

1. For the Select Device button, enter the following code:

```
Private Sub cmdSelectDevice_Click()
    DAQDevice1.SelectDevice
    txtDeviceName.Text = DAQDevice1.DeviceName
End Sub
```

2. For the Write button, enter the following code:

```
Private Sub cmdWrite_Click()
    DAQAO1.DeviceNumber = DAQDevice1.DeviceNumber
    DAQAO1.OpenDevice
    DAQAO1.RealOutput Val(txtValue)
    DAQAO1.CloseDevice
End Sub
```

Step 4: Testing your program

1. Press the Select Device button and choose the device that supports analog output.
2. Enter 2.0 in the Output value field.
3. Press the Write button to output the value.

We provide additional examples for the Analog Output Control in the \Examples\ path of the installation directory. Please refer to them for more detailed information and advanced features of the Analog Output Control.

4.5 Digital Input Control

Use the Digital Input Control (DAQDI) to perform digital input operations. The digital input lines (bits) on each data acquisition device are grouped into logical units called ports. Each port has eight bits or lines. The *Port* and *Bit* properties specify the digital input line or lines. For example, if Port = 1 and Bit =3, then the digital input line starts from the eleventh one. The Digital Input Control provides the following three kinds of functions:

4.5.1 Single Point Digital Input

Use the *BitInput* or *ByteInput* method to read single line or eight-line data. The syntax in Visual Basic is as below:

```
Value = DAQDI1.BitInput
```

4.5.2 Waveform Digital Input

Waveform digital input allows you to scan the digital lines with a fixed period. The *ScanTime* property specifies the scan period. The *EnableByteScan* or *EnableBitScan* method is used to start the waveform operation. When each scan period expires, the Digital Input Control fires an event called *OnByteScan* or *OnBitScan* along with the input data. After the operation is complete, call the *EnableByteScan* or *EnableBitScan* method with a FALSE value as input to stop the operation.

4.5.3 Digital Input with Event

This function supports the devices with DI interrupt. You configure the count for triggering an event in the *EventTrigCount* property. Then call the *EnableEvent* method to start the operation. When the count reaches the *EventTrigCount* property, the control will fire an event named *OnEvent*.

4.6 Digital Output Control

Use the digital output control (DAQDO) to perform the digital output operations. The digital output lines (bits) on each data acquisition device are grouped into logical units called ports. Each port has eight bits or lines. The *Port* and *Bit* properties specify the digital output line or lines. For example, if Port = 1 and Bit =3, then the digital input line starts from the eleventh one. After you configure the *Port* and *Bit* properties, then call the *BitOutput* or *ByteOutput* methods to perform the digital output operation. For the *ByteOutput* method, the *Mask* property is used to mask some of the digital lines when performing digital output. The masked digital lines will not change the states at output. The *Mask* property is bit-wise. For example, if you want to mask bit 3 and bit 5, then set the property to 00101000 which is equal to 40.

In addition, the Digital Output Control provides the *BitReadBack* and *ByteReadBack* methods to read back current states of the digital output lines.

4.6.1 Example: Waveform Digital Input/Digital Output

Step 1: Designing the form

1. Open a new project and form.
2. Load the Advantech ActiveDAQ Device Control, DI Control and DO Control from the **Project | Components** menu.
3. Place a DAQDevice, a DAQDI and a DAQDO control on the form. Use the default names.
4. Place three TextBox controls and enter txtDiValue, txtDoValue and txtDeviceName as their Name properties.

5. Place four CommandButton controls on the form. Enter cmdSelectDevice, cmdStartScan, cmdStopScan, and cmdWrite as their Name properties. Then enter Select Device, Scan, Stop and Write as their Caption properties.
6. Place three Label controls on the form. Enter Device, Digital input and Digital output in their Caption fields.

Your form should look similar to the one shown below :

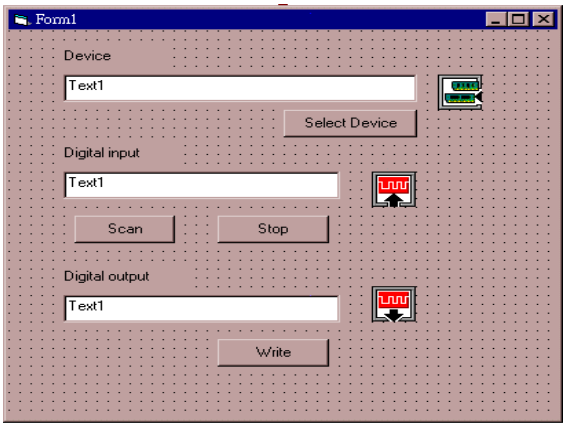


Figure 4-7: Designing form for waveform digital input/digital output example

Step 2: Configuring the properties of the DAQDO1 control

1. Enter 255 in the Mask field of the DAQDO1 control in the Properties window.

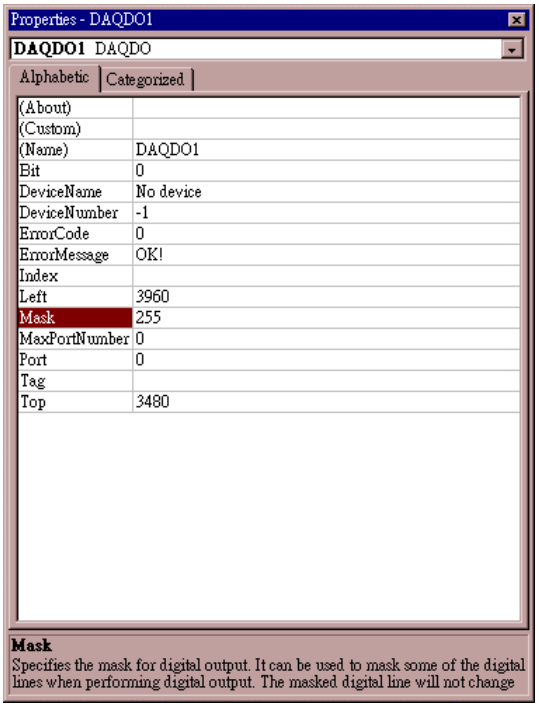


Figure 4-8: Configuring the properties of the DADDO control

Step 3: Developing the code

1. For the Select Device button, enter the following code:

```
Private Sub cmdSelectDevice_Click()
    DAQDevice1.SelectDevice
    txtDeviceName.Text = DAQDevice1.DeviceName
End Sub
```

2. For the Scan button, enter the following code:

```
Private Sub cmdStartScan_Click()
    DAQDI1.DeviceNumber = DAQDevice1.DeviceNumber
    DAQDI1.OpenDevice
    DAQDI1.EnableByteScan True
End Sub
```

3. For the Stop button, enter the following code:

```
Private Sub cmdStopScan_Click()
    DAQDI1.EnableByteScan False
    DAQDI1.CloseDevice
End Sub
```

4. Write the following code for the DAQDI1_OnByteScan event to display data:

```
Private Sub DAQDI1_OnByteScan(ByVal Data As
Integer)
    txtDiValue.Text = Data
End Sub
```

5. For the Write button, write the following code:

```
Private Sub cmdDoWrite_Click()
    DAQDO1.DeviceNumber = DAQDevice1.DeviceNumber
    DAQDO1.OpenDevice
    DAQDO1.ByteOutput Val(txtDoValue.Text)
    DAQDO1.CloseDevice
End Sub
```

Step 4: Testing your program

1. Press the Select Device button and choose the device that supports digital input.
2. Press the Scan button. You will then view the data in the digital input field.
3. Press the Stop button to terminate the digital input operation.
4. Press the Select Device button again and choose the device with digital output.
5. Enter the output data in the digital output field, then press the Write button. The result should be like below:

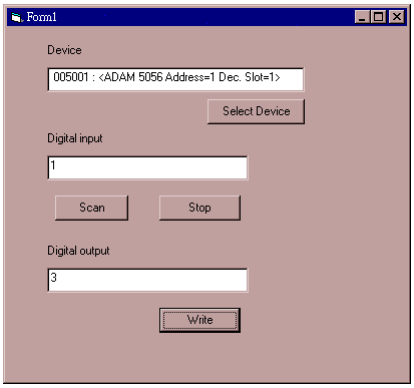


Figure 4-9: Running the waveform digital input and digital output example

We provide examples for using the Digital Input Control and Digital Output control in the \Examples\ path of the installation directory. Please refer to them for more information about the controls advanced features.

4.7 Counter Control

Use the Counter control (DAQCounter) to perform event-counting or frequency measurement. The *Channel* property specifies which *Channel* is used to perform the operation.

4.7.1 Event-Counting

Before starting the operation, you have to configure the *PresetValue* and *Direction* properties. The *PresetValue* property sets the initial value when the program starts running and the *Direction* property determines whether the counter counts up or down. You can then call the *EnableCounter* method to start the counting as follows:

```
DAQCounter1.EnableCounter TRUE
```

You can retrieve the counter value by accessing the *CounterValue* property. After you finish the operation, you have to stop and reset the counter by calling the following methods:

```
DAQCounter1.EnableCounter FALSE
DAQCounter1.ResetCounter
```

4.7.2 Frequency Measurement

To start the operation, call the *EnableFrequency* method as follows:

```
DAQCounter1.EnableFrequency TRUE
```

You can retrieve the measurement value of the frequency by accessing the *FrequencyValue* property. After you finish the operation, you have to stop and reset the counter by calling the following methods:

```
DAQCounter1.EnableFrequency FALSE
DAQCounter1.ResetCounter
```

4.7.3 Example: Event Counting

Step 1: Designing the form

1. Open a new project and form.
2. Load the Advantech ActiveDAQ Device Control and Counter Control from **the Project | Components** menu.
3. Place a DAQDevice, a DAQCounter and a Timer control on the form. Use the default names.
4. Place two TextBox controls on the form. Enter txtCounter and txtDeviceName as their Name properties.
5. Place three CommandButton controls on the form. And enter cmdSelectDevice, cmdStart, and cmdStop as their Name properties. Then enter Select Device, Start, and Stop in their Caption fields.
6. Place two Label controls on the form. Enter Device and Count in the Caption fields.

Your form should look similar to the one shown below :

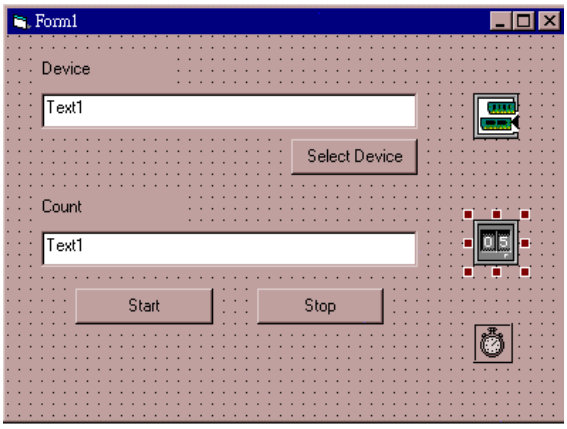


Figure 4-10: Form design in the event counting example

Step 2: Configuring the properties

None

Step 3: Developing the code

1. For the Select Device button, write the following code:

```
Private Sub cmdSelectDevice_Click()
    DAQDevice1.SelectDevice
    txtDeviceName.Text = DAQDevice1.DeviceName
End Sub
```

2. For the Start button, write the following code:

```
Private Sub cmdStart_Click()
    DAQCounter1.DeviceNumber = DAQDevice1.DeviceNumber
    DAQCounter1.OpenDevice
    DAQCounter1.EnableCounter True
    Timer1.Interval = 100
    Timer1.Enabled = True
    cmdStart.Enabled = False
    cmdStop.Enabled = True
End Sub
```

3. For the Stop button, write the following code:

```
Private Sub cmdStop_Click()
    Timer1.Enabled = False
    DAQCounter1.EnableCounter False
    DAQCounter1.ResetCounter
    DAQCounter1.CloseDevice
    cmdStart.Enabled = True
    cmdStop.Enabled = False
End Sub
```

4. Write the code of the Timer event to display data:

```
Private Sub Timer1_Timer()
    txtCounter.Text = DAQCounter1.CounterValue
End Sub
```

Step 4: Testing your program

1. Press the Select Device button and choose the device that supports event counting.
2. Press the Start button. You will then view the data in the Counter field as follows:

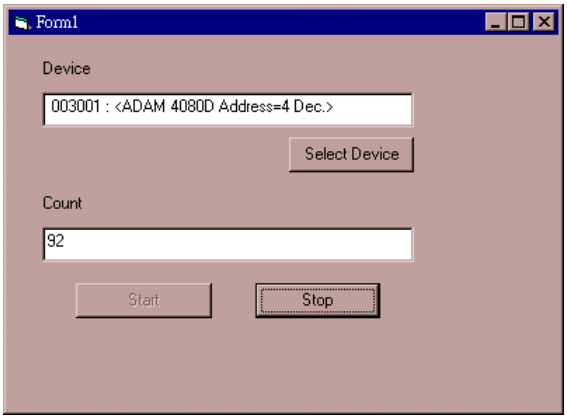


Figure 4-11: Running the event counting example

We provide examples for the Counter Control and Pulse Output Control in the \Examples\ path of the installation directory. Please refer them for more information about the controls' advanced features.

4.8 Pulse Output Control

Use the Pulse Output control (DAQPulse) to perform pulse output operations. The *Channel* property specifies the pulse output channel.

The programming method depends on the counter/timer chip on the board. There are two kinds of chips: Intel 8254 and AMD Am9513A.

For the AMD Am9513A chip, all counter channels from 0 to 9 can perform the pulse generation with an arbitrary duty cycle. The *PulsePeriod* and *PulseUpCycle* properties specify the total period and first 1/2 duty cycle. The Intel 8254 chip always generates a square wave. Hence, it does not use the *PulseUpCycle* property. For the *GateMode* property, you can configure it with gating mode. Then the pulse output operation is started by separate external hardware input.

After configuring the properties, you call the *EnablePulseOut* method to start the operations of pulse output as follows:

```
DAQPulse1.EnablePulseOut TRUE
```

When the operations are complete, you have to stop and reset it as follows:

```
DAQPulse1.EnablePulseOut FALSE
DAQPulse1.ResetPulse
```

4.9 Alarm Control

The Alarm Control (DAQAlarm) performs alarm monitoring for the analog input channel. When the input falls outside of the alarm limits, it will fire events to inform you to handle the alarm. The *Channel* property specifies the channel for alarm monitoring, and the *ScanTime* property sets the rate of alarm checking. The *HiLimit* and *LoLimit* properties specify the high limit and low limit of the alarm threshold.

There are two alarm mode options: *Momentary* and *Latched*. If the alarm is in Latched mode, the alarm will stay on even when the input value returns within limits. Setting the *RetriggerAlarm* property to TRUE can turn OFF an alarm in Latched mode. When the alarm is in Momentary mode, the alarm will be turned ON when the input value is outside of alarm limits and OFF while the input value remains within alarm limits. The *AlarmMode* property specifies the alarm mode.

After configuring the properties, you call the *EnableAlarm* method with TRUE input to start the alarm monitoring. The *Value* property returns the input value. When the input of the monitoring channel gets outside of the alarm limits, it will fire events. The *OnHiAlarm* event is triggered when the input gets higher than the high limit. The *OnLow-Alarm* event is triggered when the input goes lower than the low limit. The *OnHiToNormal* event is triggered when the input gets within the limit from high alarm state to normal state. The *OnLoToNormal* event is triggered when the input gets within the limit from low alarm state to normal state.

After the alarm operations are complete, you have to call the *En-ableAlarm* method with FALSE input and *ResetAlarm* to stop and reset the alarm.

Note: The Alarm control performs the alarm check by software instead of firmware. It will thus support any devices with analog input features, in addition to ADAM modules.

4.9.1 Example: Alarm Monitoring for Analog Input

Step 1: Designing the form

1. Open a new project and form.
2. Load the Advantech ActiveDAQ Device Control and Alarm Control from the **Project | Components** menu.
3. Place a DAQDevice, a DAQAlarm and a Timer control on the form. Use the default names.
4. Place four TextBox controls and enter txtValue, txtLoLimit, txtHiLimit and txtDeviceName as their Name properties.
5. Place three CommandButton controls on the form. Enter cmdSelectDevice, cmdStart, and cmdStop as their Name properties. Then enter Select Device, Start, and Stop in the Caption fields.
6. Place four Label controls on the form. Enter Device, Low limit, High limit and Input value in the Caption fields.
7. Place three CheckBox controls on the form. Enter chkHigh, chkNormal, and chkNormal in the Name fields. Then enter High, Normal and Low in the Caption fields.

Your form should look similar to the one shown below:

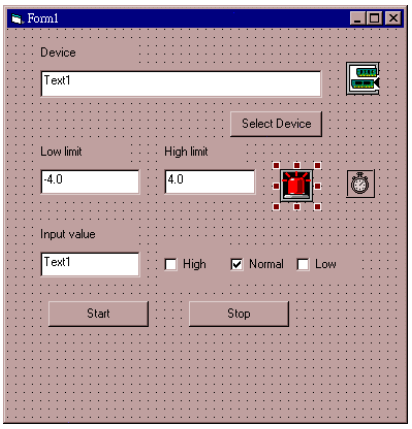


Figure 4-12: Form design for the alarm monitoring for analog input example

Step 2: Configuring the properties of the DAQAlarm1 control

1. Set the *ScanTime* property of the DAQAlarm1 control to 500.
2. Set the *Text* property of txtLoLimit control to –4.0.
3. Set the *Text* property of txtHiLimit control to 4.0.

Figure 4-13: Configuring the properties of the DAQAlarm1 control

Step 3: Writing the code

1. For the Select Device button, write the following code:

```
Private Sub cmdSelectDevice_Click()
    DAQDevice1.SelectDevice
    txtDeviceName.Text =
DAQDevice1.DeviceName
End Sub
```

2. For the Start button, write the following code:

```
Private Sub cmdStart_Click()
    chkNormal.Value = Checked
    chkHigh.Value = Unchecked
    chkLow.Value = Unchecked
    DAQAlarm1.DeviceNumber = DAQDevice1.DeviceNumber
    DAQAlarm1.OpenDevice
    DAQAlarm1.HiLimit = Val(txtHiLimit.Text)
    DAQAlarm1.LoLimit = Val(txtLoLimit.Text)
    DAQAlarm1.EnableAlarm True
    Timer1.Enabled = True
End Sub
```

3. For the Stop button, write the following code:

```
Private Sub cmdStop_Click()
    Timer1.Enabled = False
    DAQAlarm1.EnableAlarm False
    DAQAlarm1.ResetAlarm
    DAQAlarm1.CloseDevice
End Sub
```

4. Write the code of the alarm events to check the alarm state:

```
Private Sub DAQAlarm1_OnHiAlarm()
    chkHigh.Value = Checked
    chkLow.Value = Unchecked
    chkNormal.Value = Unchecked
End Sub

Private Sub DAQAlarm1_OnHiToNormal()
    chkHigh.Value = Unchecked
    chkLow.Value = Unchecked
    chkNormal.Value = Checked
End Sub
```

```
Private Sub DAQAlarm1_OnLoAlarm()
    chkHigh.Value = Unchecked
    chkLow.Value = Checked
    chkNormal.Value = Unchecked
End Sub

Private Sub DAQAlarm1_OnLoToNormal()
    chkHigh.Value = Unchecked
    chkLow.Value = Unchecked
    chkNormal.Value = Checked
End Sub
```

5. Write the code of the timer to display input data:

```
Private Sub Timer1_Timer()
    txtValue.Text = DAQAlarm1.Value
End Sub
```

Step 4: Testing your program

1. Press the Select Device button and choose the device that supports event counting.
2. Press the Start button. You will then view the alarm status and input value as follows:

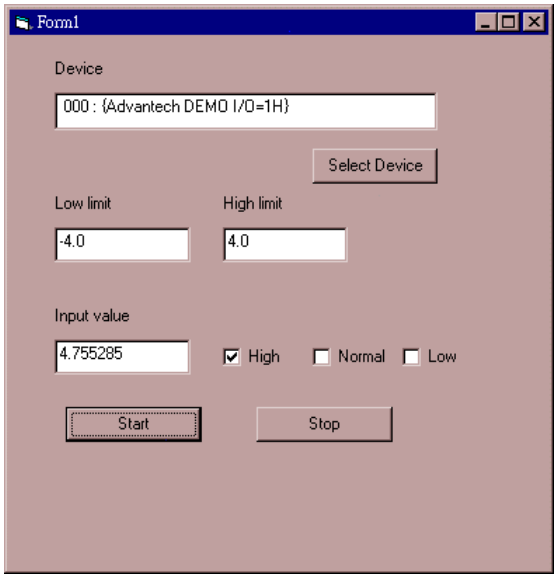


Figure 4-14: Running the alarm monitoring for analog input example

We provide examples for the Alarm Control in the \Examples\ path of the installation directory. Please refer to them for more information about the control's advanced features.

APPENDIX A

Properties, Methods and Events Reference

A.1 Device Control (DAQDevice)

A.1.1 Property List

Name	Type	Description
DeviceNumber	Long	Specifies the Device Number, previously defined through configuration using the Device Installation Utility, DEVINST.EXE.
DeviceName	String	The device name corresponding to the DeviceNumber property.
SubDeviceNumber	Long	Specifies the sub-device number, previously defined through configuration using the Device Installation Utility, DEVINST.EXE. For example, the serial port is a device, and the ADAM modules attached to the serial port are its sub-devices.
SubDeviceName	String	The sub-device name corresponding to the SubDeviceNumber property.
ErrorCode	Long	Used for storing the result of calling any methods. If it is completed normally, it is equal to zero, otherwise non-zero. See Appendix B for error code listing.
ErrorMessage	String	The error message for the ErrorCode property. It is “OK” if successful. See Appendix B for error message listing.
NumOfDevice	Short	Number of devices.
NumOfSubDevice	Short	Number of sub-devices.
GetModuleList	Boolean	Enables (TRUE) to retrieve sub-devices.

Table A-1: DAQDevice Control Property List

A.1.2 Methods

Name	Arguments	Returned Type	Description
SelectDevice	None	Long	Opens a dialog box for selecting the desired device to perform I/O operations and returns the result in the DeviceNumber property.
DeviceGetNumOfList	None	Short	Returns the number of installed devices in the NumOfDevices property
DeviceGetFirstList	None	Boolean	Returns the device number, device name, and number of sub-devices for the first device of the installed device list in the DeviceNumber, DeviceName, NumOfSubDevices properties.
DeviceGetNextList	None	Boolean	Returns the device number, name and number of sub-devices for the next device of the installed device list from last retrieving in the DeviceNumber, DeviceName, NumOfSubDevices properties.
DeviceGetFirstSubList	None	Boolean	Retrieves the first device number and name of the sub-device list for current selected device specified in the DeviceNumber property. Puts the results in the SubDeviceNumber and SubDeviceName properties.
DeviceGetNextSubList	None	None	Retrieves the next device number and name of the sub-device list from last retrieving for current selected device specified in the DeviceNumber property, and puts the results in the SubDeviceNumber and SubDeviceName properties.

Table A-2: DAQDevice Control Methods

A.2 Analog Input Control (DAQAI)

A.2.1 Property List

Name	Type	Description
DeviceNumber	Long	Specifies the Device Number, previously defined through configuration using the Device Installation Utility, DEVINST.EXE.
DeviceName	String	The device name for the DeviceNumber property.
ErrorCode	Long	Used for storing the result of calling any methods. If it is completed normally, it is equal to zero, otherwise nonzero. See Appendix B for error code listing.
ErrorMessage	String	The error message for the ErrorCode property. It is “OK” if successful. See Appendix B for error message listing.
InputRangeMode	Short	Specifies overall input range (adOverallRange or 0) or different input ranges (adDifferentRange or 1) for the scanned channels.
OverallInputRange	Short	Specifies the overall input range when the InputRangeMode property is configured for overall input range.
InputRangeList	Short Array	Specifies the input ranges for the scanned channels when the InputRangeMode property is configured for different input ranges.
StartChannel	Short	Specifies the start channel for multiple channel reading. The available channels are specified in the MaxSingledEndedChannel property or MaxDifferentialChannel property.
StopChannel	Short	Specifies the end channel for multiple channel reading. Refer to the StartChannel property.
NumberOfSamples	Long	Specifies number of samples for reading.
SampleRate	Double	Specifies sample rate in Hz.
CyclicMode	Boolean	Specifies cyclic (TRUE) or non-cyclic (FALSE).
DataType	Short	Specifies the returned data type: raw (adRaw or 0) or converted (adReal or 1).
TransferMode	Short	Specifies the data transfer mode: Software triggering (adSoftTrig or 0), Interrupt (adINTTrig or 1), DMA (adDMATrig or 2), Dual DMA (adDDMATrig or 3).

Table A-3a: DAQAI Analog Input Control Properties

Name	Type	Description
FIFOEnabled	Boolean	Enable (TRUE), or disable (FALSE) the hardware FIFO interrupt. It depends on I/O hardware.
EventEnabled	Boolean	Enable (TRUE), or disable (FALSE) to fire events.
ExtTrigger	Boolean	Specifies external trigger (TRUE), or pacer trigger (FALSE).
DaughterName	String	The name of the daughter board which is attached on the channel of the card specified by the ThermoDasChannel.
DaughterChannel	Short	Specifies the scanned channel on the daughter board.
ThermoDasChannel	Short	Specifies the scanned channel on the card.
ThemoDasGain	Short	Specifies the gain code of the scanned channel on the card.
ThermoType	Short	Specifies the thermocouple type: J (0), K (1), S (2), T (3), B (4), R (5), E (6)
ThermoScale	Short	Specifies the temperature unit: Celsius (0), Fahrenheit (1), Rankine(2), Kelvin (3)
MaxSingleEndedChannel	Short	Returns number of available analog input channels for single ended mode. The available channels of the device depend on the configuration (whether singled-ended or differential mode). It is configured by the Device Installation Utility, DEVINST.EXE.
MaxDifferentialChannel	Short	Returns the number of available analog input channels for differential mode. Refers to MaxSingleEndedChannel.
FIFOSize	Long	Returns the hardware FIFO size.

Table A-3b: DAQAI Analog Input Control Properties

A.2.2 Methods

Name	Arguments	Returned type	Description
OpenDevice	None	Boolean	Initializes the device previously specified with the DeviceName property. This method must be called before any other methods that perform I/O operations.
CloseDevice	None	Boolean	Resets the device previously used by OpenDevice. This method must be called when all I/O operations are complete.
RawInput	(IN) Short Channel	Short	Reads one sample in binary data format.
RealInput	(IN) Short Channel	Floating point	Reads one sample in voltage or converted data format.
AcquireStart	None	Boolean	Starts the reading and returns FALSE if successful
AcquireStatus	None	Long	Returns the buffer location for the current incoming data.
GetBufferData	(IN) Long BufferStart (IN) Long Count (OUT) Variant ReturnedData	Boolean	Retrieves sample data from the internal buffer, and returns FALSE if successful.
AcquireStop	None	Boolean	Terminates the reading, and returns FALSE if successful.
GetInputRange	(OUT) Short Count (OUT) Variant InputRanges	Boolean	Gets the list of the input range for the specified device, and returns FALSE if successful.
SelectDaugher	None	Short	Generates a dialog box for selecting the channel on card for temperature measurement, and returns the selected channel and sets it to the ThermoDasChannel property.
ThermoRead	None	Floating point	Reads the temperature value.
GetFirstInputRange	(OUT) String InputRange	Boolean	Get first InputRange data item from device. User has to allocate a string buffer of a minimum of 30 bytes.
GetNextInputRange	(OUT) String InputRange	Boolean	Get the other InputRange data item. User has to allocate a string buffer of a minimum of 30 bytes.

* **Delphi users**: Please see important note on page 99.

Table A-4: DAQAI Analog Input Control Methods

Note: For Delphi applications, you should use the *GetFirstInputRange* and *GetNextInputRange* methods to get the list of the input range, instead of the *GetInputRange* method.

An example is shown below:

```
var
    strGain : WideString;
    bRet : Boolean;
begin
    ...
    SetLength(strGain, 30);
    bRet := DAQAI1.GetFirstInputRange (strGain);
    while bRet = False do
    begin
        bRet := DAQAI1.GetNextInputRange (strGain);
    end;
    ...
end;
```

A.2.3 Events

Name	Arguments	Description
OnEventRaw	Long DataCount Variant Data	Triggered when the sample count reaches the NumberOfSamples property and requested returned data type is raw or binary.
OnEventReal	Long DataCount Variant Data	Triggered when the sample count reaches the NumberOfSamples property and requested returned data type is real or voltage.
OnTerminated	None	Triggered when the sample count reaches the NumberOfSamples property for noncyclic mode.

Table A-5: DAQAI Analog Input Control Events

A.3 Analog Output Control (DAQAO)

A.3.1 Property List

Name	Type	Description
DeviceNumber	Long	Specifies the Device Number, previously defined through configuration using the Device Installation Utility, DEVINST.EXE.
DeviceName	String	The device name for the DeviceNumber property.
ErrorCode	Long	Used for storing the result of calling any methods. If it is completed normally, it is equal to zero, otherwise nonzero. See Appendix B for error code listing.
ErrorMessage	String	The error message for the ErrorCode property. It is “OK” if successful. See Appendix B for error message listing.
Channel	Short	Specifies the output channel. The available output channels are specified in the MaxChannel property.
NumberOfOutputs	Long	Specifies number of output data.
OutputRate	Double	Specifies output rate in Hz.
CyclicMode	Boolean	Specifies cyclic (TRUE) or noncyclic (FALSE)
DataType	Short	Specifies the output data type, binary (0 or adRaw) or real (1 or adReal).
TransferMode	Short	Specifies the data transfer mode: Software triggering (adSoftTrig or 0), Interrupt (adINTTrig or 1), and DMA (adDMATrig or 2).
EventEnabled	Short	Enable (1), or disable (0) to fire events.
ExtTrigger	Boolean	Specifies external trigger (TRUE), or pacer trigger (FALSE).
OutputType	Short	Specifies voltage or current output. Current output is for PCI-1720.
MaxChannel	Short	Returns number of the available output channels for the specified device.

Table A-6: DAQAO Analog Output Control Properties

A.3.2 Methods

Name	Arguments	Returned type	Description
OpenDevice	None	Boolean	Initializes the device previously specified with the DeviceName property. This method must be called before any other methods that perform I/O operations.
CloseDevice	None	Boolean	Resets the device previously used by OpenDevice. This method must be called when all I/O operations are complete.
RawOutput	(IN) Short Data	Boolean	Outputs one binary data to the channel specified by the Channel property, and returns FALSE if successful.
RealOutput	(IN) Float Data	Boolean	Outputs one real data, voltage or current, to the channel specified by the Channel property. Returns FALSE if successful. See also the OutputType property and the RawOutput method.
OutputStart	None	Boolean	Starts the waveform output operation.
OutputStatus	None	Long	Returns the buffer location for the current output data.
OutputStop	None	Boolean	Terminates the waveform output operation. See also the OutputStart method.
SetRawBuffer	(IN) Variant DataArray	Boolean	Sets the output data with binary format to the internal buffer and returns FALSE if successful.
SetRealBuffer	(IN) Variant DataArray	Boolean	Sets the output data with real format to the internal buffer and returns FALSE if successful.
SetSynchronous	Boolean Enabled	Boolean	Enables synchronization of the multiple channel output. Only used for PCI-1720.
SynchronousOutput	None	Boolean	Synchronizes the outputs of the multiple channels. Only used for PCI-1720.

Table A-7: DAQAO Analog Output Control Methods

A.3.3 Events

Name	Arguments	Description
OnCompleted	None	Triggered when the output count reaches the NumberOfOutputs property.
OnTerminated	None	Triggered when the output count reaches the NumberOfOutputs property for noncyclic mode.

Table A-8: DAQAO Analog Output Control Events

A.4 Digital Input Control (DAQDI)

A.4.1 Property List

Name	Type	Description
DeviceNumber	Long	Specifies the Device Number, previously defined through configuration using the Device Installation Utility, DEVINST.EXE.
DeviceName	String	The device name for the DeviceNumber property.
ErrorCode	Long	It is used for storing the result of calling any methods. If it is completed normally, it is equal to zero, otherwise nonzero. See Appendix B for error code listing.
ErrorMessage	String	The error message for the ErrorCode property. It is “OK” if successful. See Appendix B for error message listing.
Port	Short	Specifies the digital input port. It’s 8 bits. Refers to the MaxPortNumber property.
Bit	Short	Specifies the digital input line or bit.
MaxPortNumber	Short	Returns number of the available ports for the specified device.
ScanTime	Double	Specifies the scan time for digital input in milliseconds.
EventTrigCount	Long	Specifies the count for triggering events.

Table A-9: DAQDI Digital Input Control Properties

A.4.2 Methods

Name	Arguments	Returned type	Description
OpenDevice	None	Boolean	Initializes the device previously specified with the DeviceName property. This method must be called before any other methods that perform I/O operations.
CloseDevice	None	Boolean	Resets the device previously used by OpenDevice. This method must be called when all I/O operations are complete.
BitInput	None	Boolean	Reads single line data on the digital line specified in the Port and Bit properties.
ByteInput	None	Short	Reads eight-line data starting from the digital line specified in the Port and Bit properties.
EnableByteScan	(IN) Boolean Enabled	Boolean	Enable (TRUE), or disable (FALSE) digital waveform scan for byte data.
EnableBitScan	(IN) Boolean Enabled	Boolean	Enable (TRUE), or disable (FALSE) digital waveform scan for bit data.
EnableEvent	(IN) Boolean Enabled	Boolean	Enable (TRUE), or disable (FALSE) to fire events.

Table A-10: DAQDI Digital Input Control Methods

A.4.3 Events

Name	Arguments	Description
OnByteScan	(OUT) Short Data	It is triggered for every scan period specified by the ScanTime property and it is started with the EnableByteScan method, Then returns the byte data.
OnBitScan	(OUT) Boolean Data	It is triggered for every scan period specified by the ScanTime property and it is started with the EnableBitScan method, Then returns the bit data.
OnEvent	None	Triggered when the digital input count reaches the EventTrigCount property. It is used for the device with DI interrupt.

Table A-11: DAQDI Digital Input Control Events

A.5 Digital Output Control (DAQDO)

A.5.1 Property List

Name	Type	Description
DeviceNumber	Long	Specifies the Device Number, previously defined through configuration using the Device Installation Utility, DEVINST.EXE.
DeviceName	String	The device name for the DeviceNumber property.
ErrorCode	Long	It is used for storing the result of calling any methods. If it is completed normally, it is equal to zero, otherwise nonzero. See Appendix B for error code listing.
ErrorMessage	String	The error message for the ErrorCode property. It is “OK” if successful. See Appendix B for error message listing.
Port	Short	Specifies the digital output port. It’s 8 bits. Refers to the MaxPortNumber property.
Bit	Short	Specifies the digital output line or bit.
Mask	Short	Specifies the mask for digital output. It can be used to mask some of the digital lines when performing digital output. The masked digital lines will not change the states at output. The Mask property is bit-wise. For example, you want to mask the bit 3 and bit 5, then set the property to 00101000, that is equal to 40.
MaxPortNumber	Short	Returns number of the available ports for the specified device.

Table A-12: DAQDO Digital Output Control Properties

A.5.2 Methods

Name	Arguments	Returned type	Description
OpenDevice	None	Boolean	Initializes the device previously specified with the DeviceName property. This method must be called before any other methods that perform I/O operations.
CloseDevice	None	Boolean	Resets the device previously used by OpenDevice. This method must be called when all I/O operations are complete.
ByteReadBack	None	Short	Reads the value of the digital output port (8-bit) back. Then returns the value.
ByteOutput	(IN) Short Data	Boolean	Outputs one byte data to the specified port. Returns FALSE if successful.
BitOutput	(IN) Boolean Data	Boolean	Outputs one bit value to the digital line specified by the Port and Bit properties. Returns FALSE if successful.
BitReadBack	None	Boolean	Reads one bit value back from the digital line specified by the Port and Bit properties. Returns the value.

Table A-13: DAQDI Digital Output Control Methods

A.6 Counter Control (DAQCounter)

A.6.1 Property List

Name	Type	Description
DeviceNumber	Long	Specifies the Device Number, previously defined through configuration using the Device Installation Utility, DEVINST.EXE.
DeviceName	String	The device name for the DeviceNumber property.
ErrorCode	Long	Used for storing the result of calling any methods. If it is completed normally, it is equal to zero, otherwise nonzero. See Appendix B for error code listing.
ErrorMessage	String	The error message for the ErrorCode property. It is “OK” if successful. See Appendix B for error message listing.
Channel	Short	Specifies the channel to perform event counting or frequency measurement.
Direction	Short	Determines whether the counter counts up (0) or down (1). It depends on the hardware.
PresetValue	Long	Sets the value of the counter at starting counting.
CounterValue	Long	Stores the current value of the counter.
FrequencyValue	Float	Stores the current measurement for frequency.
GateMode	Short	Specifies the gating mode: no gating (0), high level gating (1), low level gating (2), rising edge (3), and falling edge (4) for AMD Am9513A chip. If the mode is with gating, the counter may be started by separate external hardware input. It uses an external device to trigger the gate input of the counter.
GatePeriod	Short	Specifies gating period in seconds for AMD Am9513A chip and frequency measurement.
MaxCounterNumber	Short	Returns number of the available counters for the specified device.

Table A-14: DAQCounter Counter Control Properties

A.6.2 Methods

Name	Arguments	Returned Type	Description
OpenDevice	None	Boolean	Initializes the device previously specified with the DeviceName property. This method must be called before any other methods that perform I/O operations.
CloseDevice	None	Boolean	Resets the device previously used by OpenDevice. This method must be called when all I/O operations are complete.
EnableCounter	(IN) Boolean Enabled	Boolean	Starts or stops the operation of event counting.
EnableFrequency	(IN) Boolean Enabled	Boolean	Starts or stops the operation of the frequency measurement.
ResetCounter	None	Boolean	Resets the counter for event counting or frequency measurement.

Table A-15: DAQCounter Counter Control Methods

A.7 Pulse Output Control (DAQPulse)

A.7.1 Property List

Name	Type	Description
DeviceNumber	Long	Specifies the Device Number, previously defined through configuration using the Device Installation Utility, DEVINST.EXE.
DeviceName	String	The device name for the DeviceNumber property.
ErrorCode	Long	Used for storing the result of calling any methods. If it is completed normally, it is equal to zero, otherwise nonzero. See Appendix B for error code listing.
ErrorMessage	String	The error message for the ErrorCode property. It is “OK” if successful. See Appendix B for error message listing.
Channel	Short	Specifies the channel to perform pulse output.
GateMode	Short	Specifies the gating mode: no gating (0), high level gating (1), low level gating (2), rising edge (3), and falling edge (4). If the mode is with gating, the pulse output may be started by separate external hardware input.
PulsePeriod	Float	Specifies total period in seconds for AMD Am9513A chip.
PulseUpCycle	Float	Specifies the first 1/2 cycle length in seconds for AMD Am9513A.

Table A-16: DAQPulse Pulse Output Control Properties

A.7.2 Methods

Name	Arguments	Returned Type	Description
OpenDevice	None	Boolean	Initializes the device previously specified with the DeviceName property. This method must be called before any other methods that perform I/O operations.
CloseDevice	None	Boolean	Resets the device previously used by OpenDevice. This method must be called when all I/O operations are complete.
EnablePulseOut	(IN) Boolean Enabled	Boolean	Starts (TRUE) or stops (FALSE) the operation of pulse output.
ResetPulse	None	Boolean	Resets the counter for pulse output.

Table A-17: DAQPulse Pulse Output Control Methods

A.8 Alarm Control (DAQAlarm)

A.8.1 Property List

Name	Type	Description
DeviceNumber	Long	Specifies the Device Number, previously defined through configuration using the Device Installation Utility, DEVINST.EXE.
DeviceName	String	The device name for the DeviceNumber property.
ErrorCode	Long	It is used for storing the result of calling any methods. If it is completed normally, it is equal to zero, otherwise nonzero. See Appendix B for error code listing.
ErrorMessage	String	The error message for the ErrorCode property. It is “OK” if successful. See Appendix B for error message listing.
Channel	Short	Specifies the channel to perform alarm monitoring.
ScanTime	Float	Specifies the rate of the alarm checking in milliseconds.
AlarmMode	Short	Specifies the alarm mode: momentary or latched. If the alarm is in Latched mode, the alarm will stay on even when the input value returns within limits. An alarm in Latched mode can be turned OFF by setting the RetriggerAlarm property to TRUE. When the alarm is in Momentary mode, the alarm will be turned ON when the input value is outside of alarm limits and OFF while the input value remains within alarm limits.
HiLimit	Float	Specifies the high limit of the alarm threshold.
LoLimit	Float	Specifies the low limit of the alarm threshold.
Value	Float	Returns current input value.
GainCode	Short	Specifies the gain code.
RetriggerAlarm	Boolean	Re-trigger the alarm monitoring after the alarm is latched. Refers to the AlarmMode property.
ExtTrigger	Boolean	Specifies external trigger (TRUE), or pacer trigger (FALSE).

Table A-18: DAQAlarm Alarm Control Properties

A.8.2 Methods

Name	Arguments	Returned type	Description
OpenDevice	None	Boolean	Initializes the device previously specified with the DeviceName property. This method must be called before any other methods that perform I/O operations.
CloseDevice	None	Boolean	Resets the device previously used by OpenDevice. This method must be called when all I/O operations are complete.
EnableAlarm	(IN) Boolean Enabled	Boolean	Enables (TRUE) or disables (FALSE) alarm operation.
ResetAlarm	None	Boolean	Resets alarm.
GetInputRange	(OUT) Short Count (OUT) Variant InputRanges	Boolean	Gets the list of the input range for the specified device, and returns FALSE if successful.

Table A-19: DAQAlarm Alarm Control Methods

A.8.3 Events

Name	Arguments	Description
OnHiAlarm	None	Triggered when the input gets higher than the high limit.
OnLoAlarm	None	Triggered when the input gets lower than the low limit.
OnHiToNormal	None	Triggered when the input gets within the limit from high alarm state to normal state.
OnLoToNormal	None	Triggered when the input gets within the limit from low alarm state to normal state.

Table A-20: DAQAlarm Alarm Control Events

APPENDIX B

Error Messages

B.1 Driver Error Messages

This section lists the error codes and error messages returned by these controls. Each control contains the properties, ErrorCode and ErrorMessage, that indicate whether the control's method was performed successfully. When ErrorCode is not zero, it means that the method performed failed. ActiveDAQ will return the corresponding error message in the property ErrorMessage automatically. The error messages can be classified into two categories: one is generated from DLL drivers, the other one is from ActiveDAQ controls. The errors from DLL drivers are listed below.

The ErrorCode is 32-bit. Its format is described in Table B-1.

ErrorCode (32-bit)		
Bit 31-28	Bit 27-16	Bit 15-0
serial port used	base address occupied	Code

Table B-1: ErrorCode Format

A summary of the ErrorCode is listed in the following three tables:

Code	Description (ErrorMessage)
1	Not Enough Memory
2	Configuration Data Lost
3	Invalid Device Handle
4	Analog Input Failure On I/O=%XH
5	Invalid Scaled Value On I/O=%XH
6	Section Not Supported On I/O=%XH
7	Invalid Channel On I/O=%XH
8	Invalid Gain Code On I/O=%XH
9	Data Not Ready On I/O=%XH
10	Invalid Input Parameter On I/O=%XH
11	No Expansion Board Configuration in Registry/Configuration File On I/O=%XH
12	Invalid Analog Output Value On I/O=%XH
13	Configure DIO Port Failure On I/O=%XH
14	Open COM %d Failure
15	Unable to Transmit to COM %d Address %XH
16	Unable to Receive from COM %d Address %XH
17	Invalid Data Received from COM %d Address %XH
18	Configure Communication Port Falied on COM %d
19	Checksum Error from COM %d Address %XH
20	Initialization Failure On I/O=%XH
21	No Buffer Allocated for DMA
22	The Sample Rate Exceeds the Upper Limit On I/O=%XH
23	Background Operation Is Still Running On I/O=%XH
24	Board ID Is Not Supported On I/O=%XH
25	Time Interval For Frequency Measurement Is Too Small On I/O=%XH
26	Call CreateFile() Failed
27	Function Not Supported
28	Load Library Failed
29	Call GetProcAddress() Failed
30	Invalid Driver Handle
31	Module Type Not Existence On I/O=%XH
32	The Value is Out of Range On I/O=%XH
33	Invalid Windows Handle of Destination on I/O=%XH
34	Invalid Numver of Conversion On I/O=%XH
35	Invalid Number of Interrupt Count On I/O=%XH
36	Invalid Number of Event Count On I/O=%XH
37	Create or Open Event Failed On I/O=%XH
38	Interrupt Process Failed On I/O=%XH
39	Invalid digital output direction setting COM %d Address %XH
40	Invalid Event Type On I/O=%XH
41	The Time-out Interval Elapsed in Milliseconds Parameter On I/O=%XH

Table B-2a: ErrorCode Summary

Code	Description (ErrorMessage)
100	An error occured while starting the device
101	The device has not been created
102	The handle passed to the function is not a valid
103	The logic commands have created an apparent endless loop
104	Passed to the driver contains an invalid parameter
105	Attempts to access a port which has not been defined in DEVINST
106	The operation was not successful
107	The driver connects interrupt failure on I/O=%XH
108	The driver creates notification event failure On I/O=%XH
109	The system resource is insufficient OnI/O=%XH
110	An adapter object could not be created On I/O=%XH
111	The driver opens notification event failure On I/O=%XH
112	Allocate DMA buffer failure On I/O=%XH
113	Allocate MDL for DMA buffer failure On I/O=%XH
114	The buffer of requisition must be bigger that PAGE_SIZE On I/O=%XH

Table B-2b: ErrorCode Summary

Code	Description (ErrorMessage)
201	DeviceNet Initialization Failed
202	Send Message Failed On Port %d MACID %XH
203	Run Out of Message ID
204	Invalid Input Parameters
205	Error Response On Port %d MACID %XH
206	No Response On Port %d MACID %XH
207	Busy On Network On Port %d MACID %XH
208	Unknown Response On Port %d MACID %XH
209	Message Length Is Too Long on Port %d MACID %XH
210	Fragment Response Error On Port %d MACID %XH
211	Too Much Data Acknowledge On Port %d MACID %XH
212	Fragment Request Error On Port %d MACID %XH
213	Event Enable/Disable Error On Port %d MACID %XH
214	Device Net Driver Create/Open Event Failed On Port %d MACID %XH
215	IO Message Request Error On Port %d MACID %XH
216	Get Event Name From CAN Driver Failed On Port %d MACID %XH
217	Wait For Message Time Out Error On Port %d MACID %XH
218	Open CAN Card Failed
219	Close CAN Card Failed
220	DeviceNet Reset Failed

Table B-2c: ErrorCode Summary

B.2 ActiveDAQ Error Messages

Besides the error messages from DLL drivers, there are some error messages that are caused from the invalid call or improper configuration generated by ActiveDAQ controls. They are listed below:

Error Code	Description (ErrorMessage)
10001	Create the control failed.
10002	The OpenDevice method must be called before performing any I/O operations.
10003	The I/O operation is still running at background. You can not perform other I/O operations.
10004	Retrieve or set buffer data failed.
10005	The number of outputs must equal to the size of output data buffer.
10006	Retrieve or set buffer data failed at stopping running.
10007	Requested data range is out of the buffer range.
10008	Allocate or free memory failed.
10009	The high alarm limit can not be less than the low alarm limit.
10010	The number of samples must be multiple of half of FIFO size.
10011	The number of samples or outputs must be more than 4K size.
10012	The sample rate is too high.
10013	Invalid number of samples or outputs.

Table B-3: ActiveDAQ control's internal error listing

CHAPTER C

Hardware Support Listing

C.1 Hardware Support Listing

	Device							
Controls (Methods)	PCL-818 Series	PCL-818HG	PCL-1800	PCL-816	PCL-812PG	PCL-711B	MIC-2718	PCM-3718
Analog input								
RawInput	√	√	√	√	√	√	√	√
RealInput	√	√	√	√	√	√	√	√
AcquireStart	√	√	√	√	√	√	√	√
AcquireStatus	√	√	√	√	√	√	√	√
GetBufferData	√	√	√	√	√	√	√	√
AcquireStop	√	√	√	√	√	√	√	√
GetInputRange	√	√	√	√	√	√	√	√
SelectDaughter	√	√	√	√	√	√	√	√
ThermoRead	√	√	√	√	√	√	√	√
Analog output								
RawOutput	√	√	√	√	√	√		
RealOutput	√	√	√	√	√	√		
OutputStart	√	√	√	√	√	√		
OutputStatus	√	√	√	√	√	√		
OutputStop	√	√	√	√	√	√		
SetRawDataBuffer	√	√	√	√	√	√		
SetRealDataBuffer	√	√	√	√	√	√		
SynchronousOutput								
Digital input								
BitInput	√	√	√	√	√	√		√
ByteInput	√	√	√	√	√	√		√
EnableByteScan	√	√	√	√	√	√		√
EnableBitScan	√	√	√	√	√	√		√
EnableEvent	√	√	√	√	√	√	√	√
Digital output								
ByteReadBack	√	√	√	√	√	√		√
ByteOutput	√	√	√	√	√	√		√
BitOutput	√	√	√	√	√	√		√
BitReadBack	√	√	√	√	√	√		√
Counter								
EnableCounter	√	√	√	√	√	√	√	√
EnableFrequency	√	√	√	√	√	√	√	√
ResetCounter	√	√	√	√	√	√	√	√
Pulse output								
EnablePulseOut	√	√	√	√	√	√	√	√
ResetPulse	√	√	√	√	√	√	√	√
Alarm								
EnableAlarm	√	√	√	√	√	√	√	√
ResetAlarm	√	√	√	√	√	√	√	√

Table C-1: ActiveDAQ Hardware Support Listing

	Device	
Control (Method)	PCI-1710	PCI-1713
Analog input		
RawInput	√	√
RealInput	√	√
AcquireStart	√	√
AcquireStatus	√	√
GetBufferData	√	√
AcquireStop	√	√
GetInputRange	√	√
SelectDaughter		
ThermoRead		
Analog output		
RawOutput	√	
RealOutput	√	
OutputStart	√	
OutputStatus	√	
OutputStop	√	
SetRawDataBuffer	√	
SetRealDataBuffer	√	
SynchronousOutput		
Digital input		
BitInput	√	
ByteInput	√	
EnableByteScan	√	
EnableBitScan	√	
EnableEvent	√	√
Digital output		
ByteReadBack	√	
ByteOutput	√	
BitOutput	√	
BitReadBack	√	
Counter functions		
EnableCounter	√	
EnableFrequency	√	
ResetCounter	√	
Alarm		
EnableAlarm	√	√
ResetAlarm	√	√

Table C-2: ActiveDAQ Hardware Support Listing

	Device							
Control (Method)	PCIA-71A/B/C	PCL-813B	PCL-726/727	PCL-728 MIC2728	Demo Board	PCL-725/730	PCL-733 MIC-2730/2732	PCL-722/724/731 PCM-3724
Analog input								
RawInput	√	√			√			
RealInput	√	√			√			
AcquireStart	√	√			√			
AcquireStatus	√	√			√			
GetBufferData	√	√			√			
AcquireStop	√	√			√			
GetInputRange	√	√						
SelectDaughter	√							
ThermoRead	√							
Analog output								
RawOutput			√	√				
RealOutput			√	√				
OutputStart			√	√				
OutputStatus			√	√				
OutputStop			√	√				
SetRawDataBuffer			√	√				
SetRealDataBuffer			√	√				
SynchronousOutput			√	√				
Digital input								
BitInput	√		√			√	√	√
ByteInput	√		√			√	√	√
EnableByteScan	√		√			√	√	√
EnableBitScan	√		√			√	√	√
EnableEvent								√
Digital output								
ByteReadBack	√		√			√		√
ByteOutput	√		√			√		√
BitOutput	√		√			√		√
BitReadBack	√		√			√		√
Alarm								
EnableAlarm	√	√			√			
ResetAlarm	√	√			√			

Table C-3: ActiveDAQ Hardware Support Listing

	Device
Control (Methods)	PCI-1720
Analog input	
RawInput	
RealInput	
AcquireStart	
AcquireStatus	
GetBufferData	
AcquireStop	
GetInputRange	
SelectDaughter	
ThermoRead	
Analog output	
RawOutput	√
RealOutput	√
OutputStart	√
OutputStatus	√
OutputStop	√
SetRawDataBuffer	√
SetRealDataBuffer	√
SynchronousOutput	√
Digital input	
BitInput	
ByteInput	
EnableByteScan	
EnableBitScan	
EnableEvent	
Digital output	
ByteReadBack	
ByteOutput	
BitOutput	
BitReadBack	

Table C-4: ActiveDAQ Hardware Support Listing

	Device							
Control (Methods)	PCL-734/735 MIC-2750/ 2752/ 2760	PCL-833	PCL-720	PCL-721 /723	PCL-836	PCI-1750	PCI-1751	PCI-1760
Digital input								
BitInput		√	√	√	√	√	√	√
ByteInput		√	√	√	√	√	√	√
EnableByteScan		√	√	√	√	√	√	√
EnableBitScan		√	√	√	√	√	√	√
EnableEvent					√	√	√	
Digital output								
ByteReadBack	√		√	√			√	√
ByteOutput	√		√	√	√	√	√	√
BitOutput	√		√	√	√	√	√	√
BitReadBack	√		√	√		√	√	√
Counter								
EnableCounter			√		√	√	√	
EnableFrequency			√		√	√	√	
ResetCounter			√		√	√	√	
Pulse output			√		√	√	√	
EnablePulseOut			√		√		√	
ResetPulse			√		√		√	

Table C-5: ActiveDAQ Hardware Support Listing

	Device							
Control (Methods)	ADAM-4011/ 4011D	ADAM-4012	ADAM-4014D	ADAM-4018/ 4018M/ 5018	ADAM-4017/ 4013/ 5017	ADAM-4021/ 5024	ADAM-4016	ADAM-4052/ 4053/5051/5052
Analog input								
RawInput								
RealInput	√	√	√	√	√		√	
AcquireStart	√	√	√	√	√		√	
AcquireStatus	√	√	√	√	√		√	
GetBufferData	√	√	√	√	√		√	
AcquireStop	√	√	√	√	√		√	
GetInputRange								
SelectDaughter								
ThermoRead	√			√				
Analog output								
RawOutput						√		
RealOutput						√		
OutputStart						√		
OutputStatus						√		
OutputStop						√		
SetRawDataBuffer						√		
SetRealDataBuffer						√		
SynchronousOutput								
Digital input								
BitInput	√	√	√					√
ByteInput	√	√	√					√
EnableByteScan	√	√	√					√
EnableBitScan	√	√	√					√
EnableEvent								
Digital output								
ByteReadBack	√	√	√				√	
ByteOutput	√	√	√				√	
BitOutput	√	√	√				√	
BitReadBack	√	√	√				√	
Counter								
EnableCounter	√	√	√					
EnableFrequency								
ResetCounter	√	√	√					
Alarm								
EnableAlarm	√	√	√	√	√		√	
ResetAlarm	√	√	√	√	√		√	

Table C-6: ActiveDAQ Hardware Support Listing

Control (Methods)	ADAM-4060/5056/5060	ADAM-4080D	ADAM-4530	ADAM-4521	ADAM-5050	ADAM-4050
Digital input						
BitInput					√	√
ByteInput					√	√
EnableByteScan					√	√
EnableBitScan					√	√
EnableEvent						
Digital output						
ByteReadBack	√	√			√	√
ByteOutput	√	√			√	√
BitOutput	√	√			√	√
BitReadBack	√	√			√	√
Counter						
EnableCounter		√				
EnableFrequency						
ResetCounter		√				

Table C-7: ActiveDAQ Hardware Support Listing

C.2 Notice About Support For Advantech Products

Advantech ActiveDAQ controls are based on the standard Advantech DLL drivers. The ActiveDAQ controls provide support for new hardware as long as the hardware feature is supported by the controls. You simply have to install the DLL driver shipped with the hardware (it should be higher than DLL Driver version 1.11).

Hardware not supported in Advantech ActiveDAQ version 1.0 are:

- PCL-833 quadratic counter
- PCL-1800 watchdog functions
- PCI-1760 PWM/DI with event extension functions
- PCL-1800 for Windows 95
- PCL-816 for Windows 95

Notice About System Performance

Software performance may not be able to match the stated level of hardware performance, and is affected by the performance of the computer that is being used to run the application. In our testing, on a computer with an Intel Pentium® processor running at 233 MHz with 64 MB RAM memory, using the PCL-1800 analog input function with DMA transfer supports a scan rate of up to 60 kHz.

Index