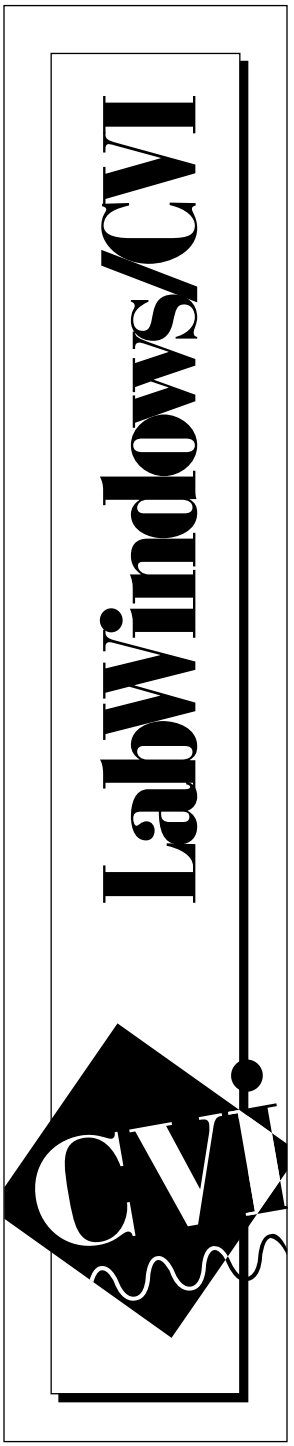


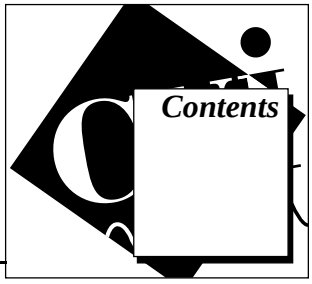
LabWindows®/CVI™ Evaluation Guide

January 1997 Edition
Part Number 350322A-01

Copyright

Trademarks

DISCLAIMER OF WARRANTY AND LIMITATION OF LIABILITY



Chapter 5
Adding a Temperature Monitor to Your Program

Chapter 6
LabWindows/CVI: The Complete Solution

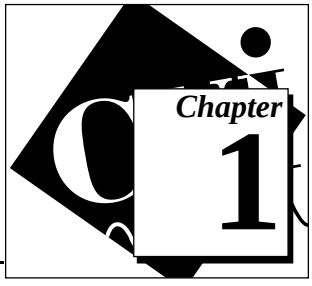
Chapter 7
National Instruments' Commitment to You

Conventions

Remember the following conventions as you use this document:

< >	Angle brackets enclose the name of a key on the keyboard—for example, <Option>. Angle brackets also enclose names of constants in the HiQ-Script language.
»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Save As** directs you to pull down the **File** menu and select **Save As**.

C/C++ Designed for Scientists and Engineers



Virtual instrumentation used to be rare in the test and measurement and industrial automation industries. Today, it has freed many scientists and engineers from the limitations of traditional instruments. Now more than ever, scientists and engineers recognize the benefits of integrating hardware and software components with PCs and workstations to create customized instrumentation solutions.

For traditional programmers who develop virtual instrumentation, LabWindows/CVI has become the most popular software development environment. Using this development tool, anyone with an elementary understanding of a text-based programming language can build virtual instruments. LabWindows/CVI combines the power of ANSI C and the flexibility of Microsoft Windows with easy-to-use tools for building virtual instrumentation systems.

Even novice programmers can use the unique interactive code generation utilities in LabWindows/CVI to unleash the power of ANSI C and speed development in the following areas.

- Creating and controlling graphical user interfaces under Windows
- Controlling instruments and plug-in data acquisition hardware from your PC
- Developing and debugging ANSI C programs for instrumentation

Furthermore, LabWindows/CVI for Windows 95/NT is now compatible with standard 32-bit C/C++ development environments from Microsoft, Borland, Symantec, and WATCOM. So whether you are a professional C programmer or a casual developer, you will be able to design and build virtual instruments quickly and easily with LabWindows/CVI.

About the Evaluation Software

Minimum System Requirements

- 386/25 MHz CPU (486/33 MHz recommended)
- Coprocessor
- 8 MB RAM
- 30 MB disk space
- VGA or Super VGA video adapter
- Windows 95/NT or Windows 3.1
- Mouse

Functionality of the LabWindows/CVI Evaluation Software

This evaluation copy of LabWindows/CVI is a fully functioning version of LabWindows/CVI with the following exceptions:

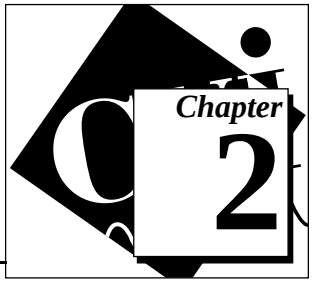
- You cannot build executable files (`.exe` or `.dll`).
- Your programs can run for only 10 minutes.
- After 30 days, you will not be able to launch the application.
- You cannot print your source and user interface files.

To prevent loss of your work during the evaluation period, you can save all programs that you build and open them later with a fully functional version of LabWindows/CVI.

Installing the Software

This evaluation guide is designed for use with the Labwindows/CVI version 4.0.1 evaluation software. You can get this software from the Software Showcase CD-ROM or from the InstrumentationWeb at `www.natinst.com/CVI`.

To install Labwindows/CVI from the Software Showcase, go to the Labwindows/CVI section in *Virtual Instrumentation Software* and click on the **Demo** icon.



Designing a User Interface

Although based in ANSI C, LabWindows/CVI is a highly visual tool for developing instrumentation applications. The user interface drives the design of your program. Because LabWindows/CVI is visual and intuitive, you can begin building a virtual instrument immediately.

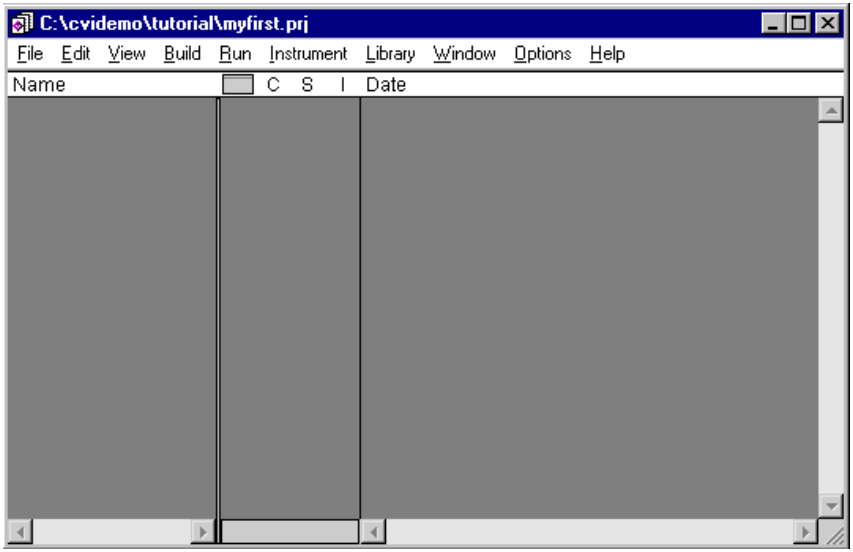
This chapter describes how to use the User Interface Editor to build the graphical user interface for your instrument. The following chapter contains instructions for creating source code directly from the user interface you build in this chapter.

A Sample Project

The graphical user interface (GUI) is the foundation of any project in LabWindows/CVI. You will open a project and create a GUI, using the LabWindows/CVI User Interface Editor. When complete, your project will perform simulated instrument control and data acquisition operations.

1. Select **Open Project** from the **File** menu. From the **Open File** dialog box, select `myfirst.prj` from the `cvidemo\tutorial` directory. Click on **Load**.

2. As shown in the following graphic, the `myfirst.prj` project file should contain no files at this time. If this project list is empty, you are ready to begin. If it is not empty, use the **Remove File** option from the **Edit** menu to remove all files from the project.



The User Interface Editor

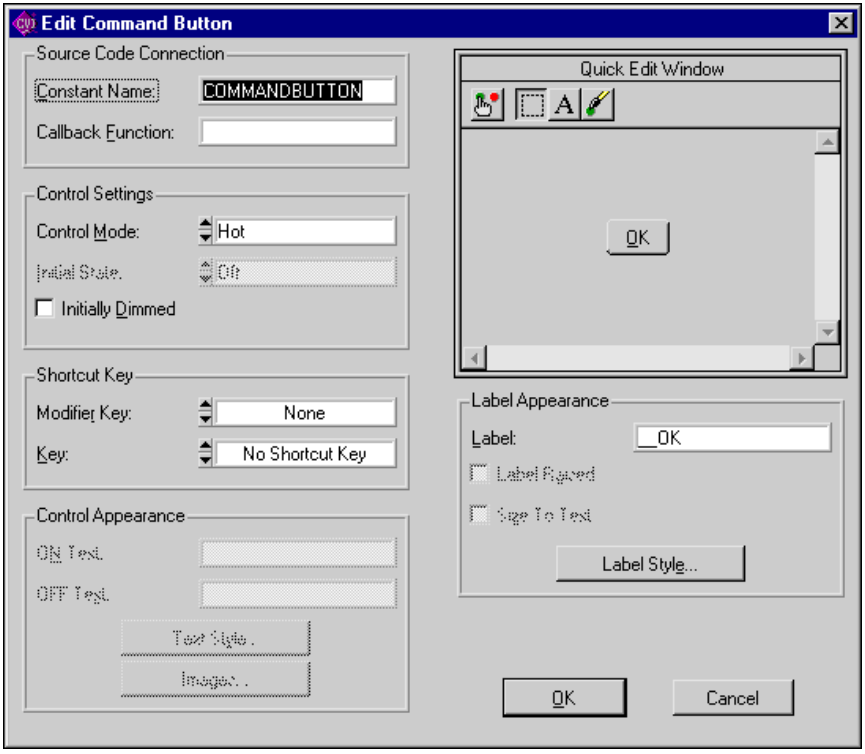
The User Interface Editor is an interactive drag-and-drop environment for the design of GUIs. You can select controls (buttons, toggles, slides, graphs, LEDs, and more) from the **Create** menu and position them on your panels. You can then use a series of dialog boxes to set attributes for the controls, such as labels, colors, and hot key connections. In the following example, you build a GUI that acquires and displays a waveform.

Task 1: Open a New User Interface Resource (.uir) File

1. Select **New»User Interface (.uir)…** from the **File** menu. The user interface editor opens an untitled file.
2. Select **Panel** from the **Create** menu to create the panel window for your GUI. Next, you can add controls and indicators to the panel window.

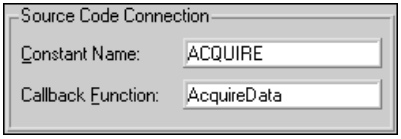
Task 2: Add a Command Button

1. Select **Command Button** from the **Create** menu and choose the **Rounded Command Button** in the palette that pops up. A button labeled **OK** appears on the panel. Use your mouse to position the button in the upper left corner of the panel.
2. To edit the button attributes, double-click on the button. A dialog box entitled Edit Command Button appears as shown in the following illustration.



3. Assign a constant name to the button. Type `ACQUIRE` in the Constant Name input control within the Source Code Connection section of the dialog box.
4. Also in the Source Code Connection section of the dialog box, type `AcquireData` in the Callback Function input control to assign a name to the function to be called when the user clicks on the button.

Be sure the Source Code Connection section of the dialog box matches the following illustration.



5. Change the label on the command button by deleting `_OK` from the **Label** input and replacing it with `Acquire`.
6. Click on **OK** at the bottom of the Edit Command Button dialog box to save your edits and close the dialog box.

Task 3: Create a Second Command Button

1. Follow the steps in Task 2 to create a second rounded command button and place it in the lower right corner of the panel. Double-click on it to open the Edit Command Button dialog box.
2. In the Edit Command Button dialog box, enter the following parameters:

Constant Name:	`QUIT`
Callback Function:	`Shutdown`
Label:	`Quit`

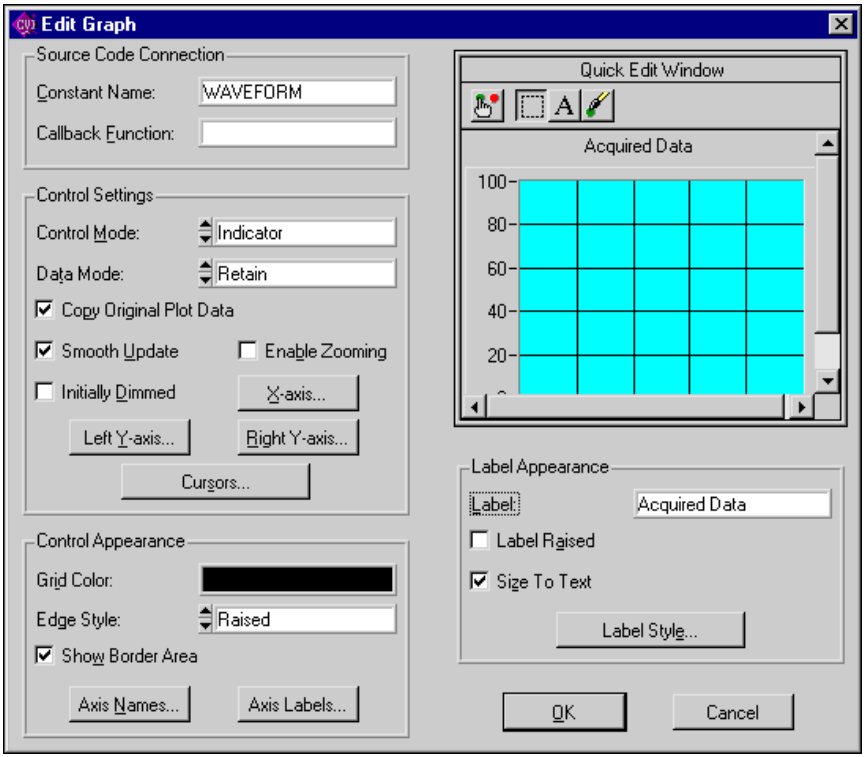
3. Click on the **OK** button at the bottom of the **Edit Command Button** dialog box to save your edits and close the dialog box.

Task 4: Add a Graph Control to the User Interface

1. Select **Create»Graph** and click on Graph control in the palette that pops up. A graph control named **Untitled Control** appears on your user interface.
2. Position and size the graph with the mouse.
3. Double-click on the graph control to display the Edit Graph dialog box.
4. Type `WAVEFORM` in the Constant Name input within the Source Code Connection section of the dialog box. Use all capital letters.

☞ **Note:** ***Because this graph does not initiate an action from the user interface, you do not need to assign a callback function to this control. Callback functions are only necessary when the operation of the control initiates an action or acts as an input.***

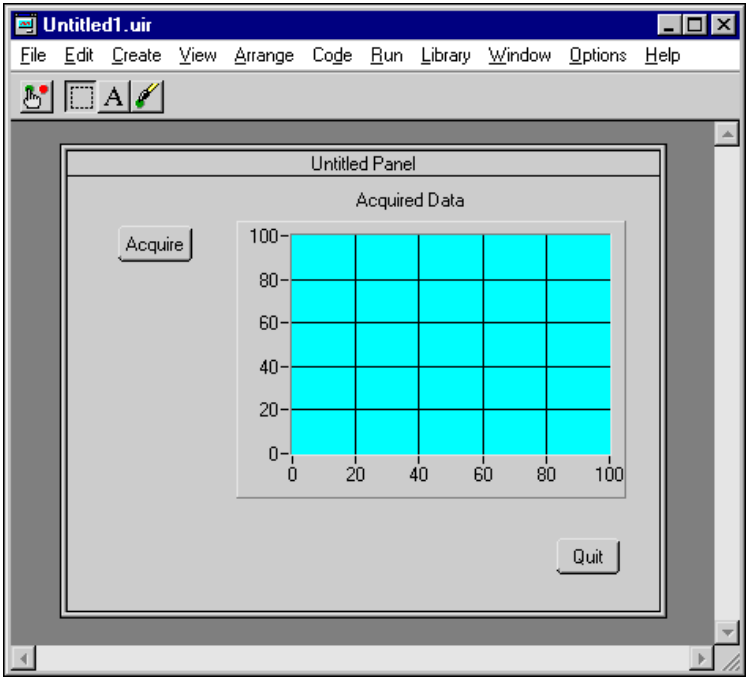
5. Type `Acquired Data` in the Label input box within the Label Appearance section of the dialog box. The dialog box should match the one shown in the following illustration.



6. Click on the **OK** button at the bottom of the Edit Graph dialog box to close the dialog box.

Task 5: Save the .uir File

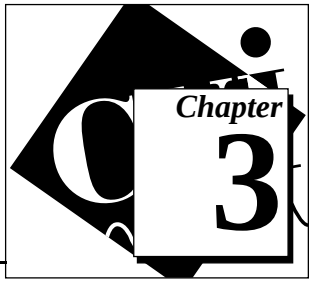
1. Ensure that your completed user interface looks like the one shown in the following illustration.



2. Select **File»Save As** to save the `.uir` file with the new controls added.
3. Type `myfirst` in the File Name input control in the **Save File As** dialog box. Click on **Save** to save the file.

☞ **Note:** ***When you save a*** `.uir` ***file, LabWindows/CVI automatically creates a header (***`.h`***) file with the same name for the user interface. This header file is stored in the same directory as the*** `.uir` ***file.***

Generating Your Program Code with CodeBuilder



In the previous chapter, you designed a simple graphical user interface. In this chapter, you will use *CodeBuilder* to automatically build the program code for this user interface. CodeBuilder creates an ANSI C program to support your user interface objects. You can compile and run this program immediately. You will need to add functions to the program that CodeBuilder generates so that it can acquire and display a waveform.

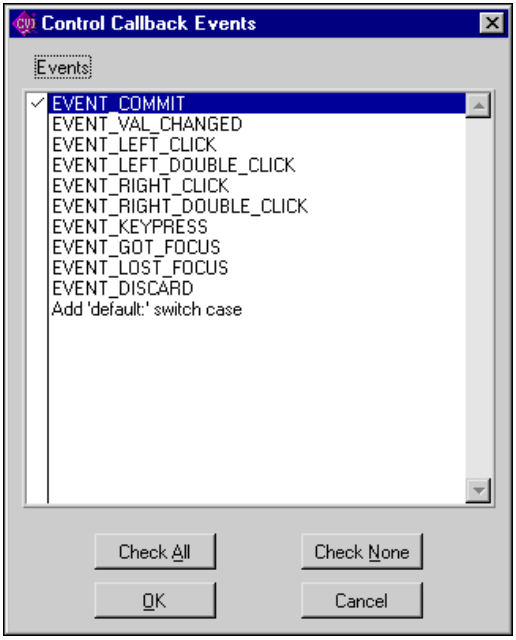
You perform the tasks in this chapter using the GUI file, `myfirst.uir`, that you built in Chapter 2, *Building a User Interface*.

Automatic Code Generation with CodeBuilder

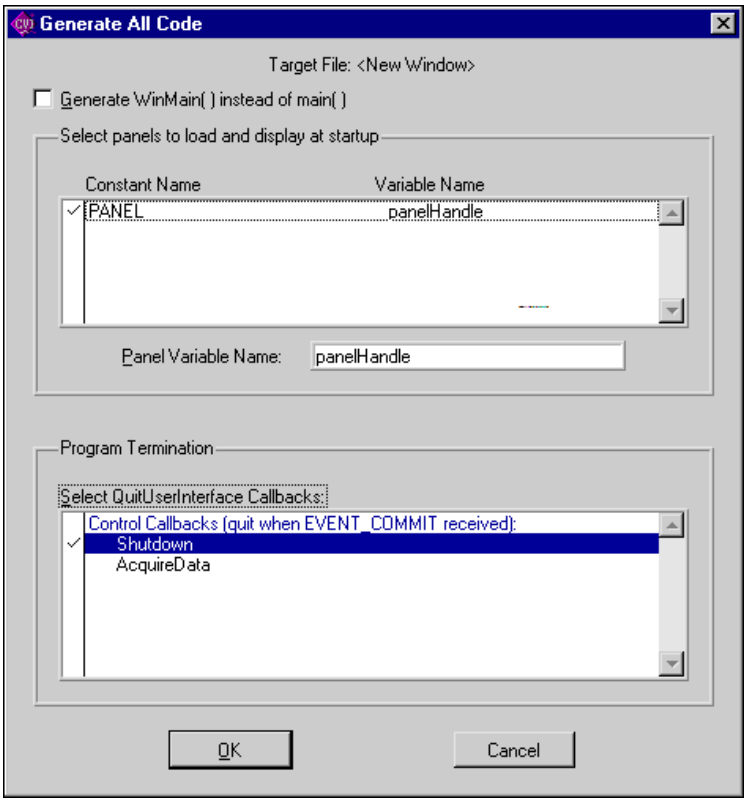
At this point, you have created a GUI with the user interface objects to start acquisition of the waveform, display the waveform, and terminate program execution. Now you need to use CodeBuilder to create source code for your GUI.

1. Specify the type of events to which your program will respond. Open `myfirst.uir` if it is not already open. Select **Code»Preferences»Default Control Events.**
2. In the Control Callback Events dialog box, you set the type of event that a control can receive. In this project, the controls should respond to one type of event: a commit event (left-click or

<Enter>). Verify that only `EVENT_COMMIT` is selected and then click on **OK**.



3. Select **Code»Generate»All Code** to display the following dialog box.



4. In general, you must set which panels you want to display at program startup. Because there is only one panel in this example you only need to verify that the Panel Variable Name is `panelHandle` and that this item is selected (checked).

☞ **Note:** ***The variable name for your GUI must be*** `panelHandle` ***to match settings you will make later in this exercise.***

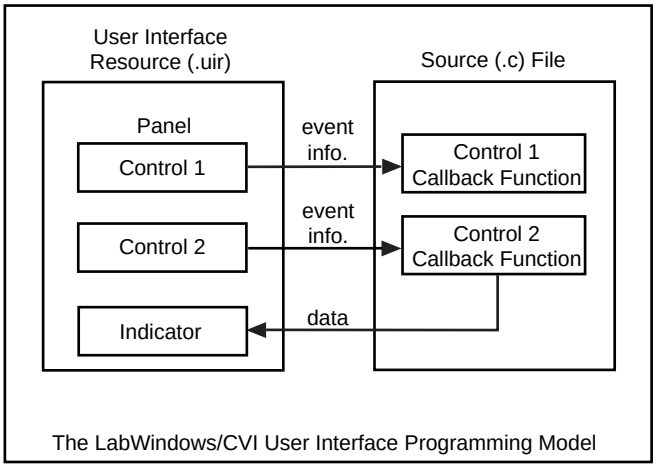
5. The lower half of the dialog box lists callback functions for your `.uir` file. You need to select the `Shutdown` function in the dialog box, so that when you click on **Quit** in your GUI, the program terminates execution. (LabWindows/CVI will insert the `QuitUserInterface` function into the `Shutdown` callback.)
6. Click on **OK**. CodeBuilder builds the source code for your program. A new Source window appears containing the new source code.
7. Select **File»Save** in the Source window and name the source file with the name `myfirst.c`.

Reviewing the Source Code

The following code should appear in your source window. Be sure your code matches the code below. Notice that the source code contains three principal functions: `main`, `AcquireData`, and `Shutdown`. You have used the User Interface Editor and the CodeBuilder feature to create these functions.

```
#include <cvirte.h>/* needed if linking executable in external compiler; harmless
otherwise */
#include <userint.h>
#include "myfirst.h"

static int panelHandle;

int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)/* needed if linking executable in external
compiler; harmless otherwise */
        return (-1);/* out of memory */
    if ((panelHandle = LoadPanel (0, "myfirst.uir", PANEL)) < 0)
        return -1;
    DisplayPanel (panelHandle);
    RunUserInterface ();
    return 0;
}

int CVICALLBACK AcquireData (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:

            break;
    }
    return 0;
}

int CVICALLBACK Shutdown (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
            QuitUserInterface (0);
            break;

    }
    return 0;
}
```

Each of the two CVICALLBACK functions in the preceding source code corresponds to a control in your user interface. The following illustration shows the structure of the LabWindows/CVI programming model.



The LabWindows/CVI User Interface Programming Model

The functions within each of the three principal functions of this project (main, AcquireData, and Shutdown) deserve further attention.

The main Function

The main function is the first component you need when you build any application. The main function for this project consists of the following lines of code:

```
int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)   /* Needed if linking in external compiler;
harmless otherwise */
        return -1;  /* out of memory */
    if ((panelHandle = LoadPanel (0, "myfirst.uir", PANEL)) < 0)
        return -1;
    DisplayPanel (panelHandle);
    RunUserInterface ();
    return 0;
}
```

The functions within the `main` function must prepare your user interface for use as follows:

- The `LoadPanel` function loads the panel from the `.uir` file into memory.
- The `DisplayPanel` function displays the panel on the screen.
- The `RunUserInterface` function prepares LabWindows/CVI to send events from the user interface to the C program you are developing.

The AcquireData Function

The `AcquireData` function shown in this section automatically executes whenever a user clicks on the **Acquire** button in your GUI. Currently, the `AcquireData` function contains no functions to execute. Your task in the next chapter is to add data acquisition functions within the `AcquireData` function.

```
int CVICALLBACK AcquireData (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:

            break;
    }
    return 0;
}
```

The Shutdown Function

The `Shutdown` function shown in this section automatically executes whenever a user clicks on the **Quit** button in your GUI. This function disables the user interface from sending event information to the callback function and halts execution of the program.

```
int CVICALLBACK Shutdown (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
            QuitUserInterface (0);
            break;
    }
    return 0;
}
```

Constructing the LabWindows/CVI Project File

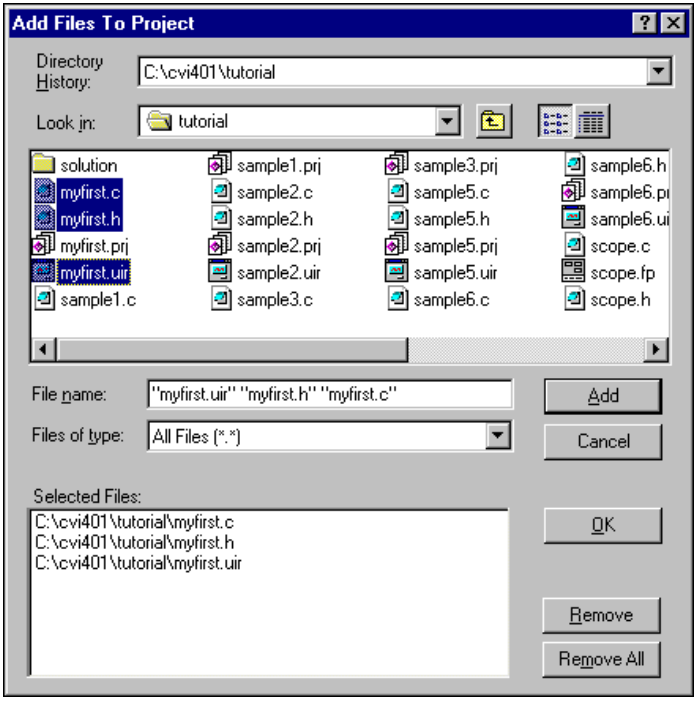
Now that you have created the source files for your program, you can construct a LabWindows/CVI project file to contain the other files. In this case, there are three files to store in the project file.

- `myfirst.uir`—the user interface file that you built in the last session.
- `myfirst.h`—the include file that was generated automatically when you saved the `.uir` file in the last session. The include file declares all of the constant names and callback functions that you assigned in the User Interface Editor (Tasks 2, 3, and 4 in Chapter 2, *Building a User Interface*).
- `myfirst.c`—the source file created in this session.

To construct a Project file, perform the following steps:

1. Close all LabWindows/CVI windows except the Project window.

2. From the menu bar, select **Edit»Add Files to Project»All Files (*.*)**. A dialog box appears, listing all the files in working directory.

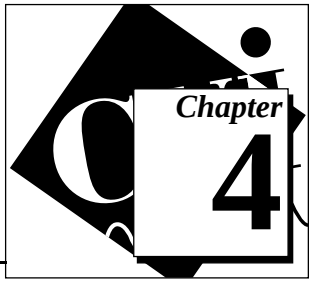


3. While pressing the <Ctrl> key, click on `myfirst.c`, `myfirst.h`, and `myfirst.uir` in the dialog box to select all three files. Click on **Add** to place these files in the Selected Files section of the dialog box.
4. Click on **OK** to add the files to the project.

Running the Project

You now have a completed LabWindows/CVI project. You can run the project by selecting **Run Project** from the **Run** menu. Currently, the only active control in your GUI is the **Quit** button. In the next chapter, you will add functions to this program to make the **Acquire** button acquire data to display. Then you will have a complete GUI that acquires and displays waveforms.

Completing Your Program with an Instrument Driver



In the last chapter, you created `myfirst.prj`, your first project in the LabWindows/CVI environment. However, the program is just a program shell. In this chapter, you will use a simple instrument driver to acquire data and you will plot the data to the graph control on your GUI.

An instrument driver is a set of functions that programs an instrument or a group of related instruments. The high-level functions in an instrument driver incorporate many low-level operations, including GPIB, VXI, or RS-232 read and write operations, data conversion, and scaling. The sample module in this chapter communicates with a simulated instrument and shows you how to use an instrument driver to acquire a waveform from an oscilloscope.

Acquiring Waveforms with an Instrument Driver

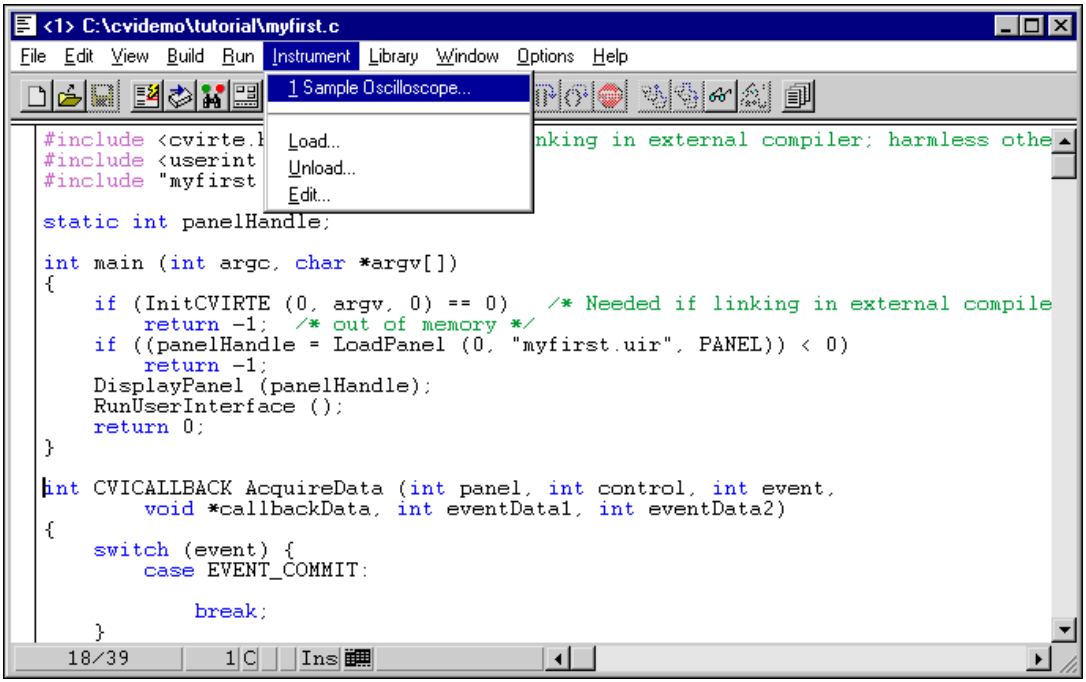
To be complete, your program needs to read an array of simulated data from an instrument driver and plot the array on the graph control. You must modify the `AcquireData` function in the `myfirst.c` source file as described in this section. Then, when a user clicks on the **Acquire** button, the program will read the data from the instrument and plot it to the graph.

Loading the Instrument Driver

The instrument driver that you need consists of several files that reside on disk. You have access to these files when you load the instrument (`.fp`) file.

1. Select **Add Files To Project»Instrument (*.fp)** from the **Edit** menu in the Project window.
2. Select the `scope.fp` file from the `tutorial` directory. Click on **Add** and then click on **OK** to add the driver to the project.

3. In the Project window, double-click on `myfirst.c` to open the Source window.



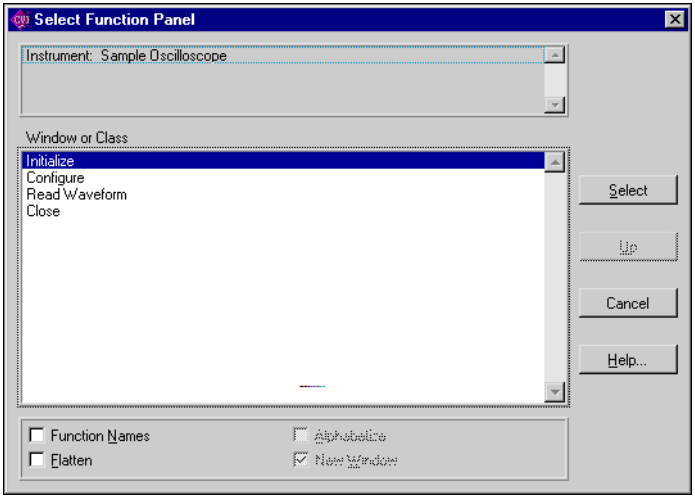
4. Place your mouse cursor on the blank line just after the `Event_Commit:` statement in the `AcquireData` function, as shown in the illustration above.

Initializing the Instrument

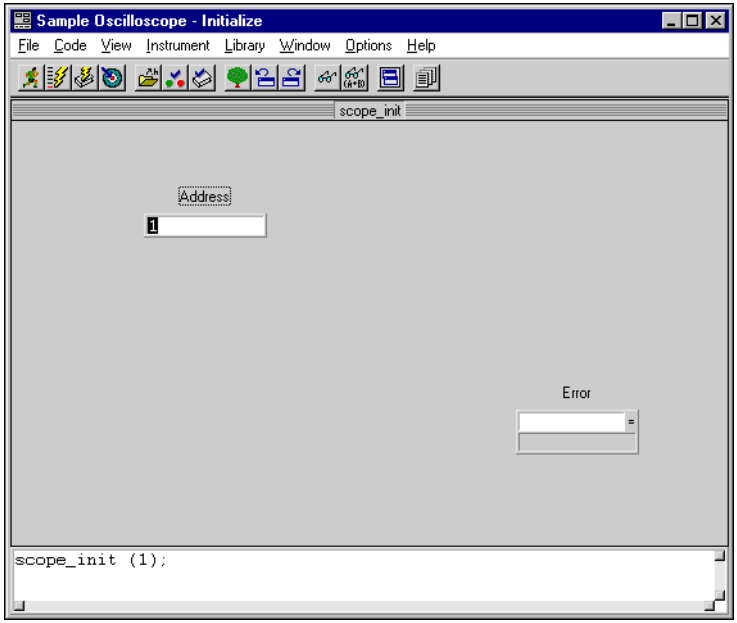
You will now use the sample oscilloscope instrument driver to generate code for acquiring a waveform. The generated code will appear in this `AcquireData` function, which is triggered by clicking on the **Acquire** button.

Task 1: Open the Function Panel

1. Select the **Sample Oscilloscope** item from the **Instrument** menu. The sample oscilloscope driver contains four simple functions for communicating with a scope, as shown in the following illustration.



2. Highlight the `Initialize` function and click on the **Select** button. The scope initialization function panel appears.



Function panels are interactive tools for building and testing your library function calls. Function panels have complete online help for the function. Right-click on the function panel to see the online help

for the Initialize function. Function panels also generate code. When you enter values in the function panel controls, the function call is built automatically at the bottom of the screen.

Task 2: Initialize the Instrument Driver Function

1. Enter the value `1` in the Address control.
2. Enter `err` in the Error control.
3. Declare the `err` variable by selecting **Declare Variable** from the **Code** menu.
4. Click on the following checkbox items: Execute declaration and Add declaration to the top of target file. Then click on **OK**.
5. Select **Run Function Panel** from the **Code** menu. Click on **Yes** when you see a dialog box prompting you to save changes before running. This dialog box appears every time you make a change.

If no errors are detected during execution, the value in the Error control is `0`. If the value is not `0`, you can right-click on the Error control to view related online help.

When your function panel is complete, perform the following steps to insert the function call into the code and remove the function panel from the screen.

Task 3: Insert the Function into the Source File

1. Select **Insert Function Call** from the **Code** menu to copy the generated code to the Source window.
2. Select **Close** from the **File** menu to close the function panel.

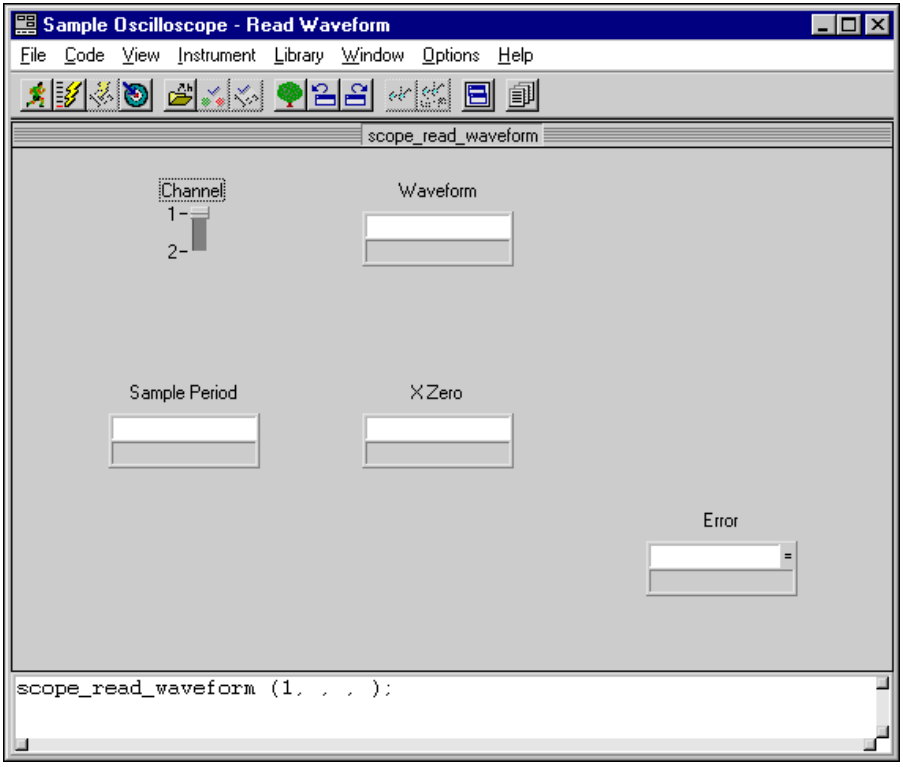
The function call to initialize the instrument driver should now appear in your source code, below the `Event_Commit` code, as follows:

```
err = scope_init (1);
```

Reading Data from the Instrument

Perhaps the most important function of an instrument driver is to read data from an instrument and convert the raw data into a format your program can use. For example, a digital oscilloscope returns a waveform as a string of comma-separated ASCII numbers. The instrument driver parses the string, scales the data to voltage, and places them into an array in memory for you.

1. Select **Read Waveform** from the **Sample Oscilloscope** instrument driver in the **Instrument** menu. The Read Waveform function panel appears as shown in the following illustration.



2. Set the Channel control to 2. Channel 1 is sine wave data and Channel 2 is random data.

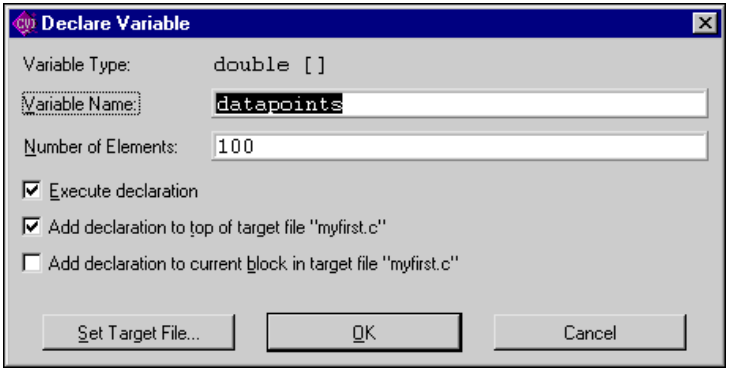
Declaring Variables from Function Panels

The `Read Waveform` function places the waveform data into an array. Before you can execute the function, you must declare an array for the waveform. You can declare variables, such as an array, from a function panel. To declare an array from the function panel, perform the following steps.

1. Click on the Waveform control and enter `datapoints`.
2. Select **Declare Variable** from the **Code** menu to declare the `datapoints` variable in memory. A dialog box appears with the

name `datapoints` automatically entered in the Variable Name input control.

3. Enter `100` in the Number of Elements input control.
4. Verify that the first Add declaration box is checked. The dialog box should look like the one in the following illustration.



5. Click **OK** to declare the datapoints array.

Completing the Read Waveform Function Panel

Complete the configuration of the function panel and run it as follows.

1. Click on the Sample Period control so that it is highlighted.
2. Select **Declare Variable** from the **Code** menu.
3. Enter `delta_t` in the Variable Name input control. Verify that the first Add declaration box is checked and then click **OK**.
4. Click on the X Zero control so that it is highlighted.
5. Select **Declare Variable** from the **Code** menu.
6. Enter `x_zero` in the Variable Name input control. Verify that the "Execute declaration" box and the first of two "Add declaration..." boxes are checked and then click on **OK**.
7. Enter `err` in the Error control.
8. Select **Run Function Panel** from the **Code** menu to execute the function panel. Save changes before running. If you executed the initialize function correctly, the Read Waveform Error control indicates `0`. After the function executes, a row of boxes in the Waveform control signifies that the data is in the waveform array.

9. [optional] You can double-click on row boxes to display two windows of data returned by the function call: one showing variable values and the other showing the data collected in the array. If you choose to review these windows, you should close both of them before moving on to the next step.
10. Select **Insert Function Call** from the **Code** menu to copy the generated code to the Source window.
11. Close the function panel. The following line of code now exists within the `AcquireData` callback function:

```
err = scope_read_waveform (2, datapoints, &delta_t, &x_zero);
```

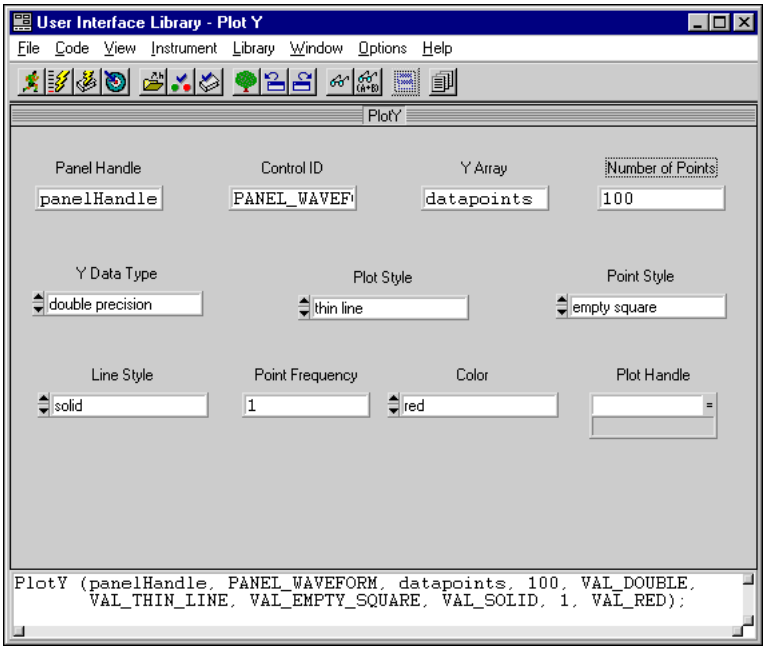
Displaying the Waveform on a Graph Control

At this point, you have completed the steps necessary to acquire data. Now you need to plot this data using the Plot Y function, which can plot the acquired data array to the graph control on the user interface.

To display and select the Plot Y function, perform the following steps.

1. Open the Plot Y function panel by selecting the following nested items. In the menu, select **Library»User Interface**. In the resulting dialog box, select **Controls/Graphs/Strip Charts»Graph and Strip Charts»Graph Plotting and Deleting»Plot Y**.
2. Fill in the function panel controls to match the following settings:

Panel Handle:	`panelHandle`
Control ID:	`PANEL_WAVEFORM`
Y Array:	`datapoints`
Number of Points:	`100`



3. From the `PlotY` function panel, select **Insert Function Call** from the **Code** menu to paste the `PlotY` function call into your source code.
4. Close the function panel. The changes you have made to the `AcquireData` callback function should match the source code shown in the following section.
5. In the Source window, select **Save** from the **File** menu to save your changes. Your program is complete.

Reviewing the Program

Your completed source file should match the following code block:

```
static double x_zero;
static double delta_t;
static double datapoints[100];
static int err;
#include <cvirte.h> /* Needed if linking in external compiler; harmless otherwise */
#include <userint.h>
#include "myfirst.h"

static int panelHandle;

int main (int argc, char *argv[])
{
if (InitCVIRTE (0, argv, 0) == 0)/* Needed if linking in external compiler;
harmless otherwise */
       return -1;/* out of memory */
    if ((panelHandle = LoadPanel (0, "myfirst.uir", PANEL)) < 0)
       return -1;
    DisplayPanel (panelHandle);
    RunUserInterface ();
    return 0;
}

int CVICALLBACK AcquireData (int panel, int control, int event,
       void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
       case EVENT_COMMIT:
           err = scope_init (1);
           err = scope_read_waveform (2, datapoints, &delta_t, &x_zero);
           PlotY (panelHandle, PANEL_WAVEFORM, datapoints, 100, VAL_DOUBLE,
               VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);

           break;
    }
    return 0;
}

int CVICALLBACK Shutdown (int panel, int control, int event,
       void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
       case EVENT_COMMIT:
           QuitUserInterface (0);
           break;
    }
    return 0;
}
```

Running the Completed Project

You now have a completed project, saved as `myfirst.prj`. You can view the status of each file associated with the project in the Project window and edit each file by double-clicking the file name in the project list.

Select **Run Project** from the **Run** menu to execute the code. During the compile process, LabWindows/CVI will recognize that your program is missing the `scope.h` header file statement and offer to insert it into your source code. Click on **Yes** to add this include file to your program and then save the file when prompted. While your program executes, the following actions occur:

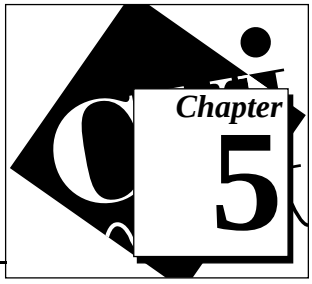
- LabWindows/CVI compiles the source code from `myfirst.c` and links with the appropriate libraries in LabWindows/CVI.
- The user interface appears, ready for keyboard or mouse input.
- When you click on the **Acquire** button, LabWindows/CVI passes the event information generated by the mouse click directly to the `AcquireData` callback function.
- The `AcquireData` function reads data from the simulated instrument and plots it on the graph control on the user interface.

☞ **Note:** ***When you click on* Acquire *more than once, the data from previous acquisitions remains on display with the new data plot. You can modify the*** `AcquireData` ***callback to include the*** `DeleteGraphPlot` ***function as as shown in the following code excerpt. This allows only one acquisition to display each time you run the program.***

Hint: you can find the DeleteGraphPlot function panel in the Library menu under User Interface»Controls/Graphs/Strip Charts»Graphs and Strip Charts»Graph Plotting and Deleting.

```
int CVICALLBACK AcquireData (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_COMMIT:
            DeleteGraphPlot (panelHandle, PANEL_WAVEFORM, -1, VAL_IMMEDIATE_DRAW);
            err = scope_read_waveform (chan_num+1, datapoints, &delta_t, &x_zero);
            PlotY (panelHandle, PANEL_WAVEFORM, datapoints, 100, VAL_DOUBLE,
                VAL_THIN_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
            break;
    }
    return 0;
}
```

Adding a Temperature Monitor to Your Program



In the last chapter, you used an instrument driver to add code to your program to gather waveform data. In this chapter, you will modify your program to gather temperature data using a timer callback function to perform synchronous data acquisition. You will use a simulated data acquisition driver to acquire and display a temperature value on a thermometer.

☞ **Note:** ***You need to build the project*** `myfirst.prj` ***in order to perform the exercises in this chapter. You can build*** `myfirst.prj` ***by completing the exercises in the second, third, and fourth chapters of this evaluation guide.***

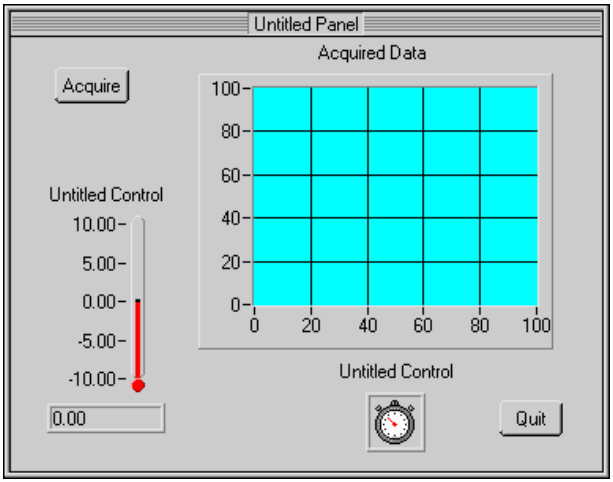
Timer Controls

In the last example, you used an **Acquire** button to generate an event. Timer controls are used to automatically generate events at specified intervals. In this example, you will use a Timer control to trigger a data acquisition operation once every second for reading temperature. You must add a Timer control and a thermometer indicator to display the acquired temperature on your user interface.

1. Open the `myfirst.prj` project.
2. Double-click on `myfirst.uir` in the Project window. From the **Create** menu, select **Timer**. A timer control appears on your user interface. You can place the timer anywhere on your panel (even on top of another control) because the timer will not be visible on the GUI when you run the program.
3. Double-click on the Timer control to display the **Edit Timer** dialog box. Modify the following items in the dialog box and then click on **OK** to save these settings:

Constant Name:	`ACQUIRE_SYNC`
Callback Function:	`SyncAcquire`
Interval:	`0.100`

4. From the **Create** menu, select **Numeric»Thermometer** to add a thermometer to your user interface. Place the thermometer in an open area as shown in the following illustration.



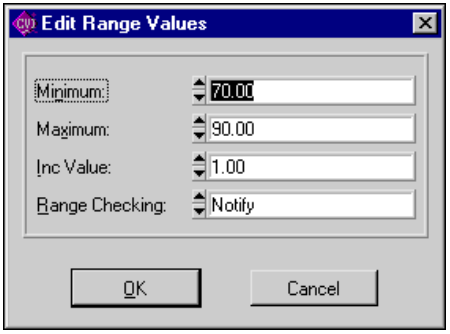
5. Double-click on the Thermometer control to display the **Edit Thermometer** dialog box. Modify the following items in the dialog box:

   Constant Name: `THERMOMETER`

   Default Value: `70` (You can click on **OK** when you see the out-of-range message)

   Label: `Temperature`

6. In the **Edit Thermometer** dialog box, you also need to change the range on the thermometer. Click on **Range Values** to display the following dialog box. Set the values in the dialog box as shown and then click on **OK**.



7. After you have set the Constant Name, Default Value, Range Values, and Label, click on **OK** to close and save your settings in the **Edit Thermometer** dialog box.
8. Save the user interface file by selecting **Save** from the **File** menu. Close the User Interface Editor Window.

Your changes to the user interface are now complete.

Adding the Timer Callback to Your Program

Now that you have made the appropriate additions to your user interface, you need to add the timer callback to your source code. First, open the `myfirst.c` Source window. In the User Interface Editor, click on the timer control so that it is highlighted. Right-click on the Timer control and select **Generate Control Callback.** The callback for the Timer control is inserted at the bottom of `myfirst.c`. Click on the Source window to view the code. The insertion should match the following code excerpt:

```
int CVICALLBACK SyncAquire (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_TIMER_TICK:

            break;
    }
    return 0;
}
```

☞ **Note:** ***Keep the following points in mind as you insert functions into a source code file:***

- ***The source code file you want to modify must be open.***
- ***When you insert a function call from a function panel, LabWindows/CVI inserts the new code where the cursor is located in your source code file.***
- ***When you insert a control callback from the User Interface Editor, LabWindows/CVI inserts the new code at the end of your source code file.***

Adding a DAQ Function to the Timer Callback

Now that the timer callback is inserted into the program, you need to add a simulated data acquisition (DAQ) function to it. Follow these steps to add the necessary functions to your program:

1. This evaluation guide has a simulated DAQ board for reading temperature. In the Project window, select **Add Files To Project»Instrument (*.fp)** from the **Edit** menu.
2. Select the `simpldaq.fp` file from the `cvi\tutorial` directory. Click on **Add** and then click on **OK** to add the driver to the project.
3. Position your cursor on the blank line after `EVENT_TIMER_TICK:` in the `SyncAcquire` callback function.
4. Select **Instrument** from the menu bar to verify that the instrument driver was loaded. Click on **Simple Data Acquisition…** to open the instrument driver.
5. The **Select Function Panel** dialog box currently contains only one function: `Read_Value`. Click on **Select** to display the dialog box for this function panel.
6. When the dialog box appears, click on the **Temperature** control and enter the variable name, `temp_value`.
7. The variable, `temp_value`, must be declared. From the menu bar, select **Code»Declare Variable…**. Verify that the Execute declaration checkbox and the Add declaration to top of target file checkbox are both checked. Then click on **OK**.
8. Place your cursor in the **Error** control and enter `err`. This variable was declared previously and requires no new declaration.

9. From the menu bar, select **Code»Insert Function Call** and then close the function panel window. Your `SyncAcquire` callback now includes a function call:

```
int CVICALLBACK SyncAcquire (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_TIMER_TICK:
            err = simpldaq_read_value (1, 1, &temp_value);

            break;
    }
    return 0;
}
```

Adding a Function to Update the Thermometer

Your program now contains a DAQ function to acquire simulated data. The program also needs a function to update the thermometer with the newly acquired data. Follow these steps to add this function and complete the program:

1. In the Source window, place your cursor on the blank line after the `simpldaq_read_value` function.
2. Open the Set Control Value function panel. From the menu bar, select **Library»User Interface**. Then in the resulting dialog box, select **Controls/Graphs/Strip Charts»General Functions»Set Control Value**.
3. Fill in the function panel controls as follows:

   Panel Handle: `panelHandle`

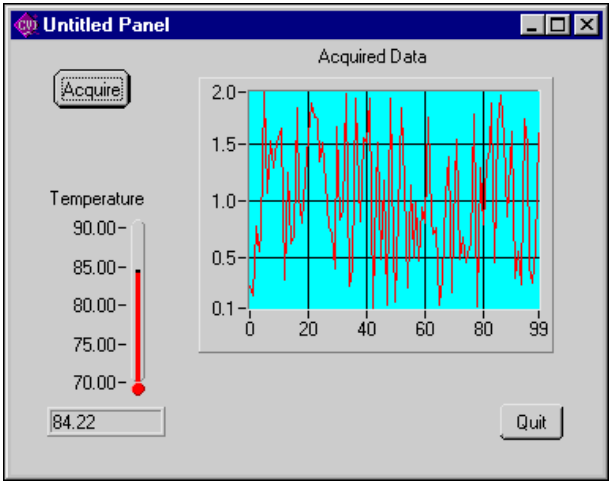
   Control ID: `PANEL_THERMOMETER`

   Value: `temp_value`
4. From the menu bar, select **Code»Insert Function Call** to insert the `SetCtrlVal` function call into your timer callback. Close the Function Panel window and return to the Source window. Your callback should match the following excerpt:

```
int CVICALLBACK SyncAcquire (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event) {
        case EVENT_TIMER_TICK:
            err = simpldaq_read_value (1, 1, &temp_value);
            SetCtrlVal (panelHandle, PANEL_THERMOMETER, temp_value);

break;
    }
    return 0;
}
```

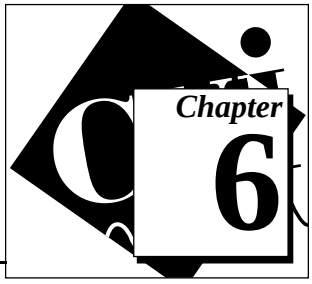
Running Your Project

Select **Run Project** from the **Run** menu. During the compile process, LabWindows/CVI will recognize that your program is missing the `simpldaq.h` header file statement. Choose **Yes** to add this include file to your program and then save the file when prompted.



When your program runs, the internal timer generates an event based on the interval setting. When the event is generated, the timer callback function is called and the thermometer indicator displays the new temperature. The timer runs the DAQ function automatically “in the background,” allowing the users to acquire and graph a waveform from the scope instrument whenever they click on **Acquire**.

LabWindows/CVI: The Complete Solution



Although you have completed this evaluation guide, you should remember that you have only scratched the surface. The capabilities of LabWindows/CVI extend far beyond the what you have seen up to this point. LabWindows/CVI offers a complete solution for your instrumentation programming needs, including an ever-improving feature set and many add-on toolkits.

More Features to Meet Your Needs

National Instruments is constantly improving and extending the capabilities of LabWindows/CVI and adding hardware support options and toolkits.

Features

LabWindows/CVI features full compatibility with standard programming tools. All LabWindows/CVI user interface, analysis and instrumentation libraries are available as standard 32-bit DLLs for use in general-purpose C/C++ programming environments from Microsoft, Borland, Symantec, and WATCOM. In addition to building standard executable (`.exe`) files, LabWindows/CVI allows you to build 32-bit DLLs that are compatible not only with your general-purpose C/C++ tools but also with Visual BASIC and LabVIEW.

As always, LabWindows/CVI gives you the most complete instrument driver solution on the market: over 600 drivers for instruments from more than 65 instrument vendors. The instrument driver library includes drivers for GPIB, VXI, serial, and CAMAC instruments. Furthermore, the VXI*plug&play* Systems Alliance has endorsed LabWindows/CVI as a core technology and a basis for industry standardization.

Platforms

LabWindows/CVI supports the following operating systems and hardware platforms.

Platform Support	System Requirements	Compatible Compilers
Windows 95/ NT/3.1	486/33 with FPU capability recommended, 25 MB disk space, VGA monitor	Microsoft Visual C++, Borland C++, WATCOM C, Symantec C (under 32-bit Windows operating systems only)
Sun Solaris	SPARCstation 1 Workstation, 24 MB main memory 32 MB disk swap space, 12 MB disk space for application files	Sun ANSI C compiler (acc), GNU C compiler (gcc)
HP-UX (Run-time Libraries only)	HP-UX 700 Series Workstation, 24 MB main memory, 8 MB free hard disk space prior to installation, HP-UX 9.05 or greater operating system	HP-UX C compiler, GNU C compiler (gcc)

Add-On Toolkits

National Instruments offers a growing collection of add-on toolkits for LabWindows/CVI that can help you in particular application areas. Add-on toolkits are additional libraries or utilities developed to meet the specific needs of specialized markets, industries, and application areas.

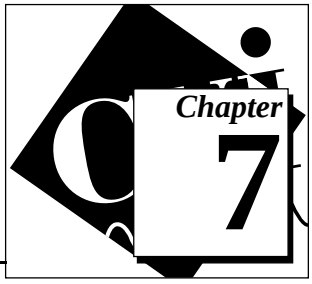
- **Test Executive**—a test management system that handles test sequencing, looping, and data logging for production test situations. You can concentrate on developing individual test modules and let the Test Executive manage and control the tests for you. See also the SPC Toolkit.
- **SQL Toolkit (Windows only)**—a collection of libraries you can use to connect to more than 30 local or remote databases. With these libraries, you can connect your test programs or the Test Executive directly to a database to store the results or download test parameters.
- **SPC Toolkit**—a library of statistical process control functions for analyzing production quality using process statistics, control charts, and Pareto analysis. See also the Test Executive toolkit.

- **PID Control Toolkit**—a set of algorithms for PID control which you can integrate into your process monitoring and control applications.
- **Internet Developers Toolkit (Windows only)**—a library of functions to help you create a Web server to display your LabWindows/CVI user interface panels on the Internet. Users of Web browsers can click on these panels to interact with your LabWindows/CVI applications. You can also send e-mail messages from your applications and transfer files to and from FTP servers.
- **Digital Filter Design Toolkit**—a general purpose design tool for signal conditioning, control systems, and digital signal processing.
- **Third-Octave Analysis Toolkit (Windows only)**—a ready-to-run, PC-based third-octave analyzer.

More Examples for LabWindows/CVI

After you complete this evaluation guide, you can use the *Getting Started with LabWindows/CVI* manual for a complete tutorial on LabWindows/CVI. This tutorial manual is available in Software Showcase CD-ROM. The National Instruments Web site offers more examples that demonstrate the basic concepts of LabWindows/CVI at `www.natinst.com/cvi`.

National Instruments' Commitment to You



LabWindows/CVI represents a long-standing commitment by National Instruments to provide tools that simplify the development of instrumentation systems for programmers using standard languages. When you choose LabWindows/CVI as your development environment, you join thousands of scientists and engineers who are taking advantage of the power of ANSI C programming and the flexibility of Microsoft Windows for their instrumentation systems.

Long-Term Compatibility

LabWindows/CVI is the culmination of years of development and continuous support and improvement of the LabWindows programming tools. LabWindows users in the DOS environment can use LabWindows/CVI to translate their programs immediately to run under Microsoft Windows 3.1, Windows 95, and Windows NT. This commitment to supporting and maintaining your development investment does not end with Windows. Today you can develop programs in LabWindows/CVI under Microsoft Windows and run them on Sun SPARCstations under Solaris or HP 700-Series Workstations running HP-UX. National Instruments will continue to support and maintain LabWindows/CVI on the major operating systems of the future as well. When LabWindows/CVI moves to these future operating systems, you can be assured that your development efforts will upgrade smoothly.

Customer Education

National Instruments offers comprehensive training courses on LabWindows/CVI to help you learn to build applications quickly. The three-day Basics course and two-day Advanced course give you concentrated instruction, as well as design tips, on the LabWindows/CVI development environment from our applications engineers. These courses take place monthly at our corporate headquarters in Austin, Texas, and also at our branches around the

world. In addition, we offer two-day hardware courses on GPIB, DAQ, and VXI to help you develop your entire system.

Alliance Program

The Alliance program is a network of third-party developers and consultants who are experts in LabWindows/CVI. The National Instruments *Solutions* directory lists additional libraries and utilities developed by our Alliance members to help you use LabWindows/CVI. In addition, the *Solutions* directory lists expert LabWindows/CVI consultants you can hire to help you develop custom applications.

Technical Support

National Instruments offers a wealth of technical support. You can use our Internet sites (Web and FTP), BBS, or fax-on-demand systems to download valuable information and product examples, question-and-answer documents, and technical development tips. A technical forum for LabWindows/CVI is available on the Internet where you can discuss issues with other LabWindows/CVI users. In addition, National Instruments has experienced applications engineers located throughout the world to assist you.

You can access technical support through the following paths:

Web Support

Web Site: `www.natinst.com`
FTP Site: `ftp.natinst.com`

E-Mail Support

`support@natinst.com`
`lw.support@natinst.com`

Bulletin Board Support

BBS United States: (512) 794-5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-On-Demand Support

(512) 418-1111

U. S. Telephone Support

(512) 795-8248
(512) 794-5678 (Fax)

International Offices

Australia 02 9874 4100, Austria 0662 45 79 90 0,
Belgium 02 757 00 20, Canada (Ontario) 905 785 0085,
Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 09 527 2321, France 01 48 14 24 24, Germany 089 741 31 30,
Hong Kong 2645 3186, Israel 03 5734815, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635,
Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886,
Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51,
Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway, Austin, TX 78730-5039, (512) 794-0100

Documentation Comment Form

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

Title: *LabWindows®/CVI™ Evaluation Guide*

Edition Date: January 1997

Part Number: 350322A-01

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

_____

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (___)_____ Fax (___) _____

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

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678