

RM1579 - Predictive Control & Intelligent Sensors

LabWindows/CVI Tutorial

What is LabWindows?

LabWindows/CVI is an integrated ANSI-C programming environment designed for engineers and scientists to create virtual instrumentation applications. It has many built-in libraries to perform data acquisition, instrument control, networking etc. It allows the user to design customized measuring systems to monitor or automate processes.

Getting Started

1) Start a new project

Once CVI has been launched, the project window appears. The project window is used to open new or previously created projects and to compile, build, and run the loaded project. Save your project, for example, *Temperature Control.prj*. This is where you will manage all your files attached to this project. The file extensions are as follows:

- *.uir (graphical user interface file)
- *.c (the main C-code file)
- *.h (any header files)
- *.fp (any instrument specific files required)

2) Design the user interface

CVI has the ability to create sophisticated, yet user-friendly graphical user interfaces (GUI's). To open the **User Interface Editor**, select **File >> New >> User Interface (*.uir)**. Save the interface file. I generally use the same filename as my project, to keep everything organized and simple. What you now have is a **Untitled Panel**. This is where you place various control objects. You can create panels within the parent panel, called child panels and grandchild panels. CVI has "drag and drop" capabilities for users to easily design and customize interfaces.

For this example, we will be acquiring temperature from one thermocouple. Before starting the interface, you need to determine how you want to display your temperature. Add a few command buttons, such as *ACQUIRE*, *STOP*, *SAVE*, *QUIT*. To edit the properties of each button, double click on it, bringing up the edit screen as shown in Figure 1. Assign the buttons a constant name as well as a callback function name.

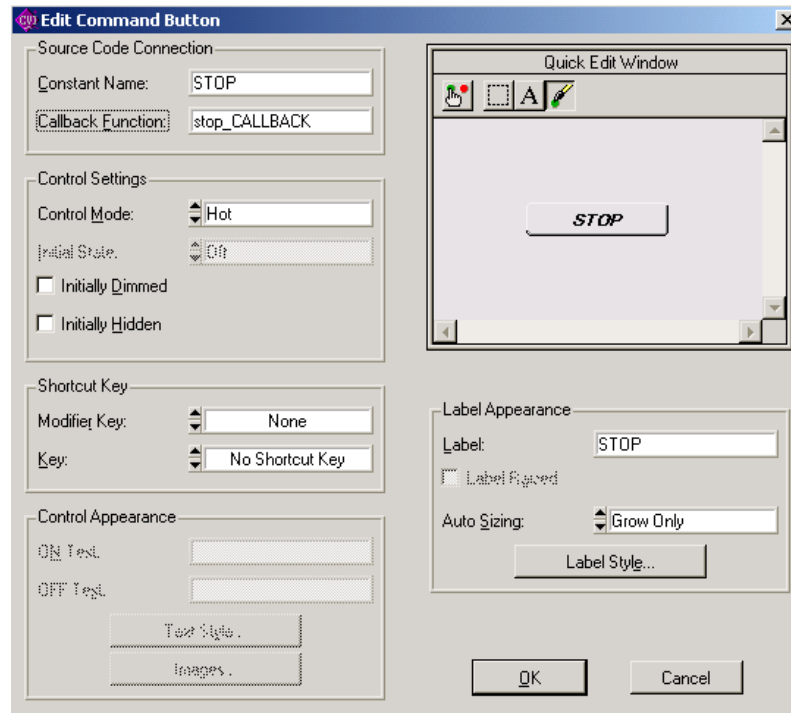


Figure 1 - Command Button properties

Plotting can be done with a Strip Chart or a Graph. A Strip Chart displays graphical data in real time, whereas a Graph plots curves, points, shapes defined as arrays. A Strip Chart is commonly used for monitoring purposes, and will be used in this example. Select **Creat -> Graph -> StripChart**. Size the chart object and double click to edit its properties. A Strip Chart doesn't need a callback Function name. The properties panel for a Strip Chart is shown in Figure 2.

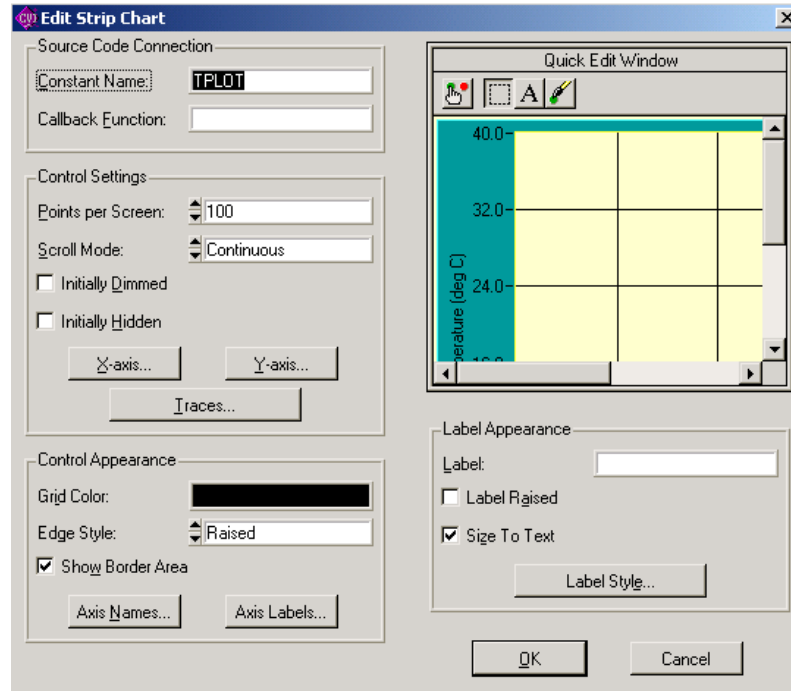


Figure 2 - Stripchart properties

Numerical values, for example temperature and or time, can be displayed on the interface by using a 'numeric control'. This field can be used to collect user input or to display acquired or calculated values. When setting the properties for the numeric make sure the *Data Type* is determined correctly. (ie. int, double etc.) Like graphs, numerics do not need a callback Function name.

The final product may look something like that shown in Figure 3.

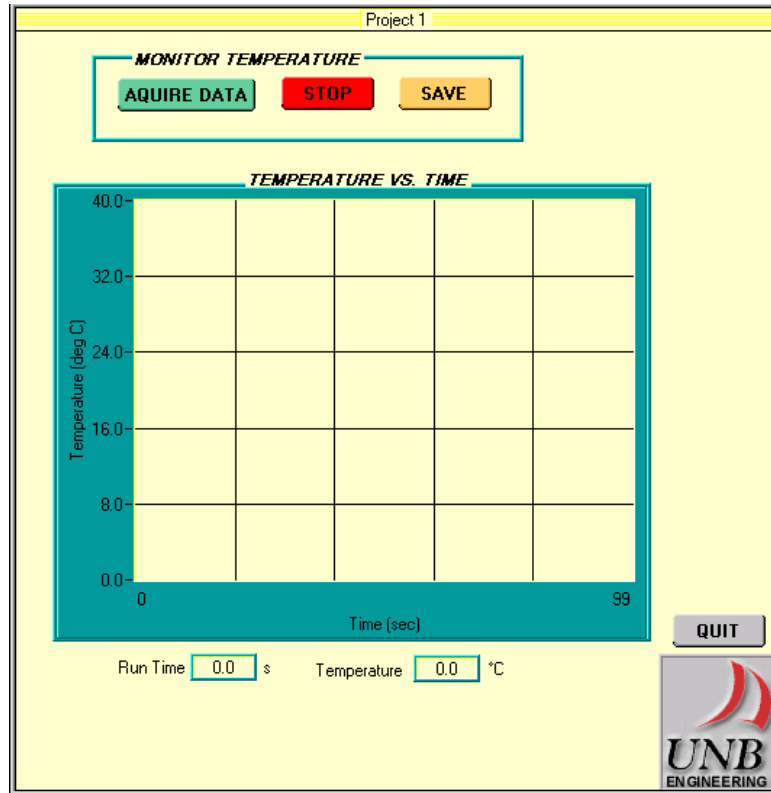


Figure 3 - Example graphical user interface

3) Generate the code

After the GUI is complete and you are satisfied with the interface layout, it is time to generate the skeleton code, which will link the data acquisition to the interface. The *CodeBuilder* creates syntactically and programmatically correct code. It gathers all the callback function names defined in your interface and generates the skeleton code for those functions. Thereafter, all you need to do is write the C-code to the callback functions to perform the actions you wish. To generate the code select **Code >> Generate >> All code**. The *Generate All Code* panel appears. If you have more than one panel associated to your interface you need to select the one to be displayed when launching your program. The callback function associated with terminating the program must also be selected as displayed in Figure 4.

Now the framework for your program is generated and displayed. Save this C-file, again using the same filename as your project (if you want). *CodeBuilder* also generates a project specific header file (*.h file). DO NOT EDIT THIS HEADER FILE. It is integral to your interface management, as it is based on the layout of the user interface and your defined preferences. All the newly generated files need to be added to your project list. This is done from the project window panel. Select **Edit >> Add files to project >> All files**. Add the interface file (*.uir), the header file (*.h) and the c-source file (*.c). Here you can see why it is convenient to store all your project files under the same name!

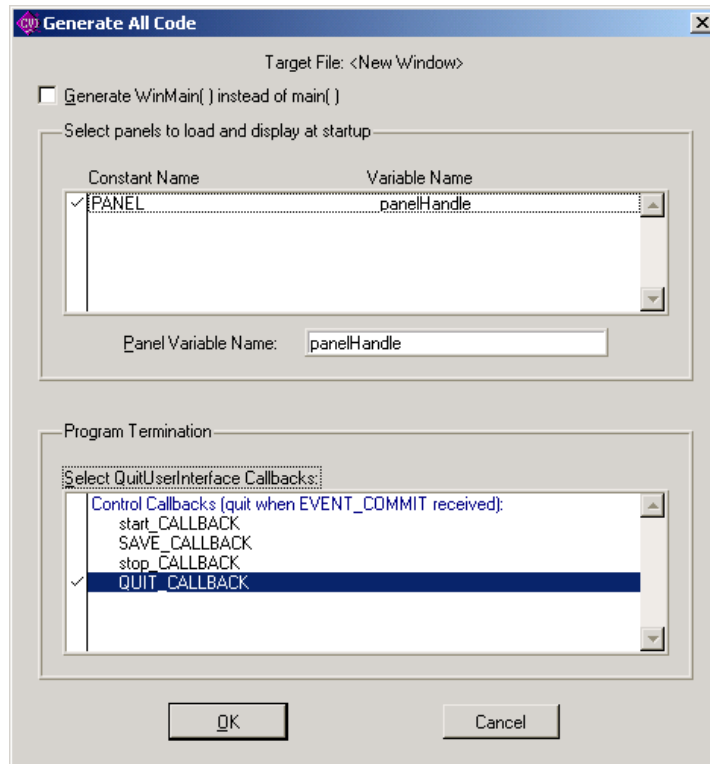


Figure 4 - Generate all Code

Main Program

1) Load Instrument

For this example, we are using a thermocouple. The data acquisition board senses a voltage differential from the thermocouple. This analog voltage needs to be converted to temperature. To do so you need to load the instrument file. Select **Instrument >> Load >>** browse to `o:\nal\programs\cvi\toolslib\daqUtil` and chose `convert.fp`. Add this file to the project list.

2) Variable header file

It is common practice to define all your variables in a header file. Select a new file and save it as a *.h file, for example `Variables.h`. Include this file in your project list. You will also need to link the variable file to your main C-program by typing the following at the top (using the filename you chose):

```
#include "Variables.h"
```

3) Initializing DAQ

The DAQ must be initialized and configured each time the program is launched. This basically tells the program which channel to sense the voltage signals from. You can write a separate function to perform the initialization. This function call is typically located in the main c-program before the `RunUserInterface()` call is made. I will call this function `Initialize()`. The main program now looks like:

```

int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1;          /* out of memory */
    if ((panelHandle = LoadPanel (0, "Temperature Control.uir", PANEL)) < 0)
        return -1;
    Initialize();
    DisplayPanel (panelHandle);
    RunUserInterface ();
    DiscardPanel (panelHandle);
    return 0;
}

```

Now we need to define the Initialize() function. The return type is not a number, just a state, therefore the function declaration is of return type *void*. CVI has many built in Data Acquisition functions and these are found in **Library >> Data Acquisition**. Each function in the libraries have an extensive *Help* to guide you in the right direction. First you need to initialize the board, and then configure it to accept the type of signal it will be receiving. For example, a thermocouple sends an analog voltage. To initialize the board, select **Library >> Data Acquisition >> Initialize Board**. To configure, select **Library >> Data Acquisition >> Analog Input >> Single Input >> Configure Analog Input**. Example code used to initialize and configure the DAQ is:

```

/***** Initialize and Configure DAQ *****/
void Initialize(void)
{
    Init_DA_Brds (1, &board);
    //Configure channel 1
    AI_Configure (1, 1, 0, 10, 0, 0);
}

```

4) Callback Functions

Now you need to program the action you that you want to occur when any of the button are activated. For example, you want to acquire temperature data until the user selects stop or quit. One way of doing this is by flagging commands. When the start button is activated, a start flag is set to a value, typically 1, enters a loop until the start flag is set to another value, say 0, when the stop of quit button is activated. Below is an example code fragment for a start callback function. The program_Run function call contains the actual data acquire, and convert.

```

int CVICALLBACK start_CALLBACK (int panel, int control, int event,
                               void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            RunTime = 0;
            while(StartFlag!=1) // Run as long as the flag is not set
            {
                program_Run();
            }
            break;
    }
    return 0;
}

```

Final Note: LabWindows/CVI is very user friendly, however, like any programming environment there is a learning curve. You **will not** be able to write your first acquisition program in one day. Be patient, use the *Help*, and it will come together. Being able to use LabWindows/CVI brings control into a digital world.