

## Experiment 1: Programming and data analysis in Labwindows/CVI

In experimental physics, the main goal is always to obtain quantitative information about a physical system. This is usually done with an instrument or detector that is sensitive to some quantity of interest. Measurements of these quantities result in data that must be analyzed. Some experiments, such as some of those done by biochemists, result in so few data points that it is sufficient to write down the resulting numbers in a table, and then perhaps to do some calculations with a spreadsheet program such as MS Excel. Most of the time, however, there is enough data generated that it must be analyzed with a computer program. Sometimes these programs are obtained as commercial software designed to analyze a specific type of data. However, many researchers find that they need to write their own program in order to properly analyze experimental data. Therefore, programming knowledge is essential for any experimental physicist. In this course, we will introduce you to some simple programming techniques so that you have an intellectual basis for the development of further programming skills, such as those needed for the computational physics course offered every fall.

There are many programming languages available for data analysis, such as Fortran, C, Pascal, Basic, MathCAD, Origin, Kaleidagraph, etc. There are significantly fewer programs available that allow one to directly access data acquisition cards, which are used to obtain data directly from an instrument to a computer. The most typical programs in use for data acquisition are Labview and Labwindows (CVI), both from National Instruments, one of the main companies that makes data acquisition hardware. Labview is an entirely graphical programming language. It is the most commonly used program in experimental labs, and it can be used for many types of programs, but because of the graphical nature of the language, it becomes quite cumbersome and unwieldy when loops and other constructs are needed. It does, however, allow one to quickly set up an experiment to acquire, display, and save data without having any prior programming experience. This is done by allowing the user to create a virtual instrument (VI), which

represents the user interface. The VI contains buttons and displays that execute functions designed in the graphical programming language. Some of the experiments we will do later have prewritten programs in Labview that acquire data. However, very little if any Labview programming will be required for this course.

In Experiment 1, we will learn to use Labwindows. Labwindows has the same

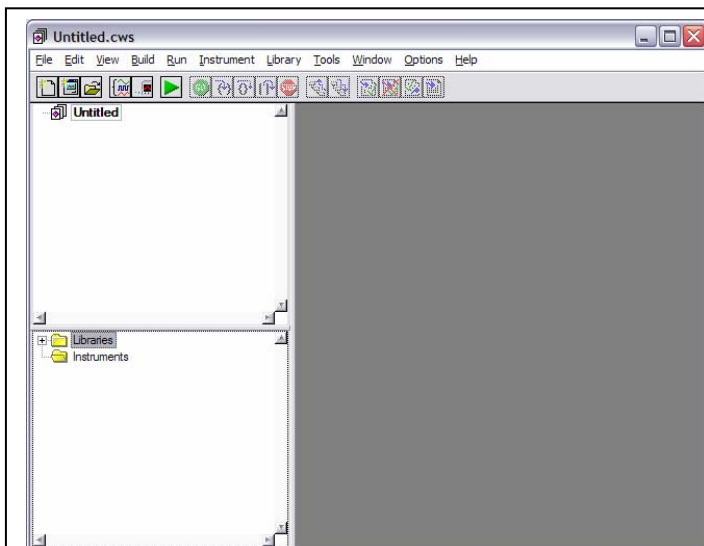


Figure 1. National Instruments CVI startup.

virtual instrument format as Labview, but the buttons run functions in C and the displays are the results of calculations done in C. Thus, Labwindows contains a nice combination of an easy to use virtual instrument interface with a powerful programming language that is commonly used by both theoretical and experimental physicists.

## 1. Programming in Labwindows

In this lab, we will learn how to analyze data using Labwindows. To do this, you will learn to create a virtual instrument panel. Start by running “National Instruments CVI” on one of the laboratory computers. You will see the window shown in Fig. 1. To create a new program, choose File -> New -> User Interface. All programs start with a user interface, and this is how each user will interact with the program. All of the user-interactive elements of the program are represented by switches, buttons, and numeric displays and controls that are first created as part of the .uir file.

Now you will need to design your virtual instrument. To do this, first think about what the user will input into the program and how to display the results. The user interacts with the program by entering numbers into boxes and by pushing buttons and changing switches. ***The first object you should create in any panel is a quit button.*** This button will end the program when activated by the user. To do this, right-click on the untitled panel and choose a command button. It should appear as shown in Fig. 3. To edit your command button, double-click on it to see the screen in Fig. 4.

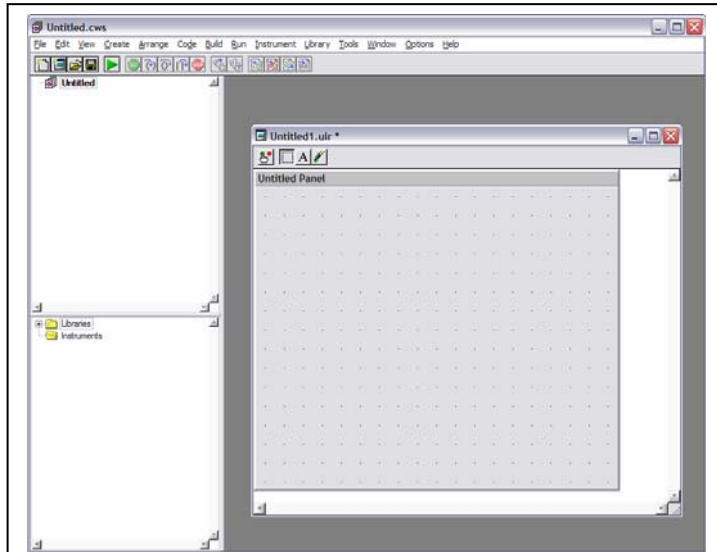


Figure 2. Blank user interface at startup.

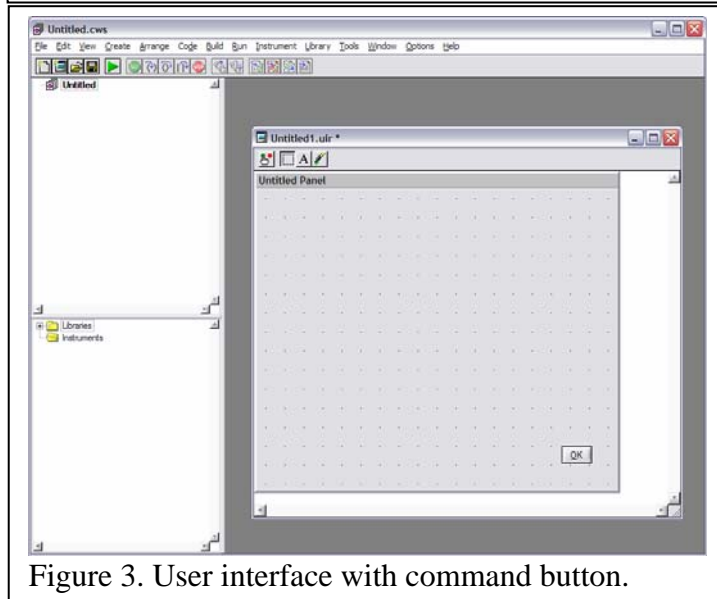


Figure 3. User interface with command button.

The screen in Fig. 4 controls the way this button will function when the program is run. You can rename it by typing in Label. Name this one Quit after its function. Note that the Constant Name is `COMMANDBUTTON`, a reasonable default name. This will work fine in the program you will write. The “Callback Function” is very important. This tells the program whether or not it will run a subroutine when the button is clicked. In this case, we want the button to terminate the program, so we will need a function to do this. Call this function “Bye”. After doing this, you can close the Edit Command Button window by choosing OK. You should now have a Panel with a Quit button. However, there is no code to execute yet.

To create execution code, we can have CVI start things out for us. First, save your new program using the File -> Save As command. Then choose Code -> Generate -> All Code. This will generate the screen in Fig. 5. The defaults are fine, but every program must have a `QuitUserInterface` Callback, which in this case is the Bye function, so this should be selected.

When you select OK, the CVI program will create your new C program, which will look like that shown in Fig. 6. Note that this is automatically formatted correctly for C (or C++) and you can simply insert any more text directly into the program between the curly brackets in `main { }`. Any additional items that you add to your user interface can also have their own callback functions. This is a menu-driven system, so just look for something that seems like it will do what you want, and try it out. As you create more

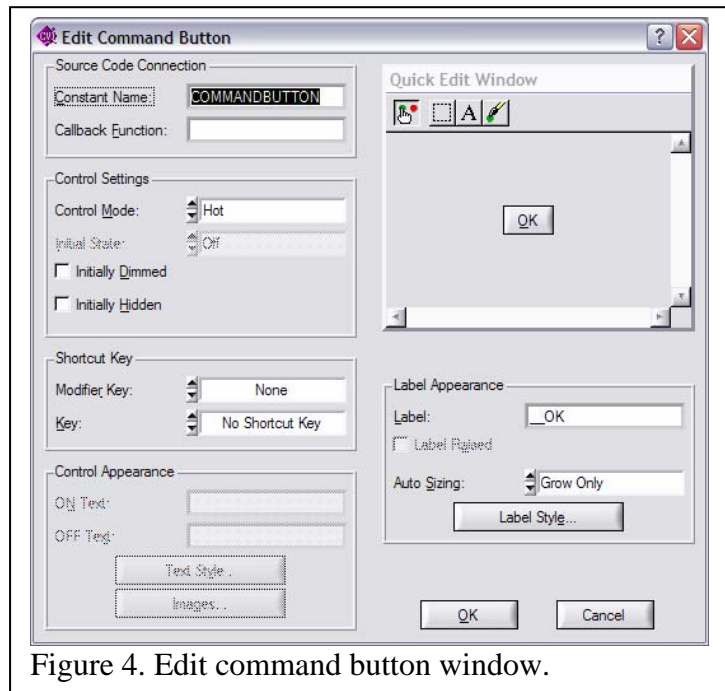


Figure 4. Edit command button window.

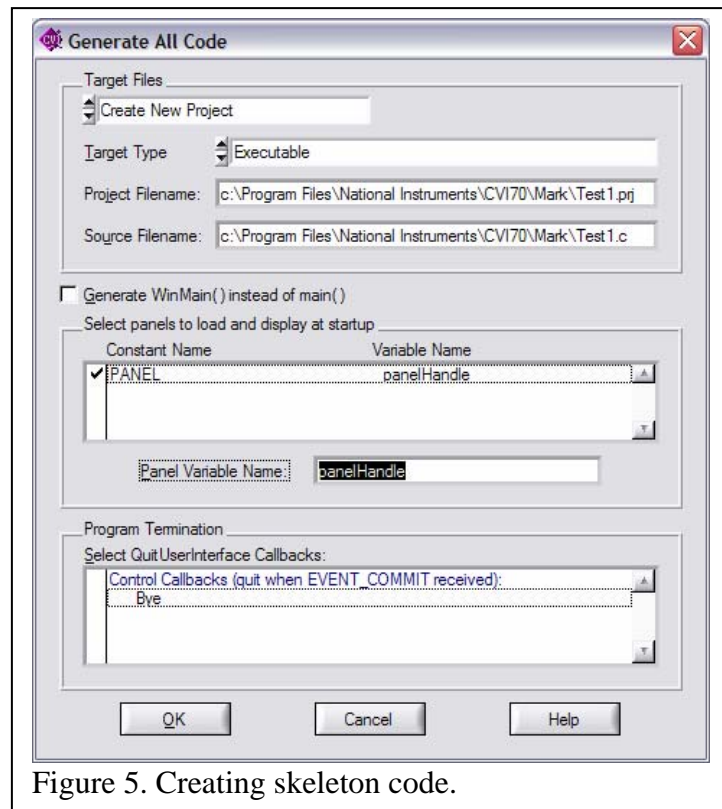


Figure 5. Creating skeleton code.

callback functions, they will appear below main. Very little CVI programming is done in the main section of the program. Most of what will be done in the program will be done by a function that is activated upon choosing a button.

After creating the Quit button, you can create other buttons that will have new program functions.

Create a new button on the user interface and name it `add_one`. Name the control callback `add_one` as well. Then right-click on it and choose Generate Control Callback. This creates the `add_one` function in your C code that will be executed. For now, leave the function blank. Now choose Debug from the Run menu and your program will run. If you click on the new button (the one you just created), it will do nothing because there is no code in that function. If you click on Quit, the program will terminate. You have now written a working (and useless) C program.

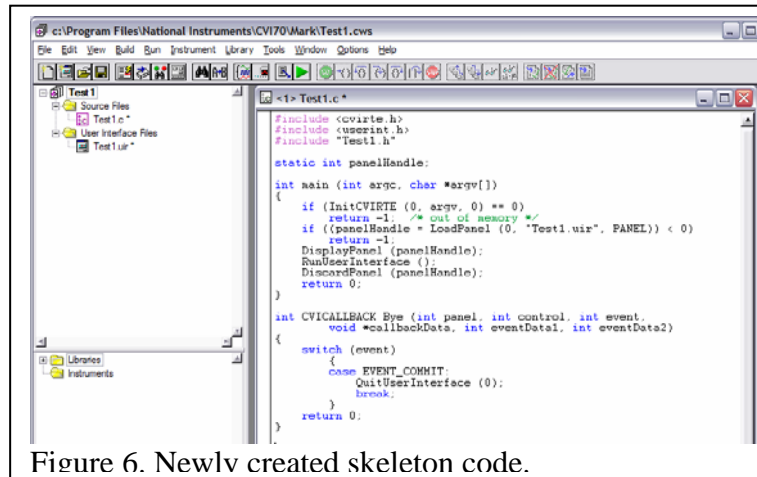


Figure 6. Newlv created skeleton code.

## 2. Elements of the C code.

The C code you have generated consists of several parts. At the top, we have several “#include” statements. These statements “include” other C code that is not explicitly written in your code. If we didn’t have these include statements, C code would be very long and cumbersome. CVI already included the files it needs, such as `cvirte.h` (CVI runtime environment) and `userint.h` (the user interface header). We do not need to worry about these “header files”. Header files are just files containing more C code, but which end in “.h”. If you are curious you can open these files in CVI and read the C code that makes up the user interface.

The next section reads “`static int panelHandle;`” This line declares the variable `panelHandle`. Here `static` means that this is a global variable – it can be used in the main program and in other functions. “`int`” means that this variable is an integer. This is one of several variable types such as `char` (text), `short` (short integer, same as `int`), `long` (long integer), `float` (single precision decimal number), and `double` (double precision decimal). Most data will be float or double. CVI functions (found in the function library) often require specific types of variable, and these will be found in the function description.

The next section of the program is shown below:

```

int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1; /* out of memory */
    if ((panelHandle = LoadPanel (0, "Test1.uir", PANEL)) < 0)
        return -1;
    DisplayPanel (panelHandle);
    RunUserInterface ();
    DiscardPanel (panelHandle);
    return 0;
}

```

This is the main program. The LoadPanel function loads the user interface you have created. Displaypanel displays it on the desktop and RunUserInterface runs the event handler routine. This routine responds only when a specific item on the interface panel is clicked, at which point it runs the appropriate function, such as Bye. When the Bye function is called, it returns to main and DiscardPanel closes the interface and terminates the program.

All of the functions are listed below this, with the Bye and add\_one function shown here:

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

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

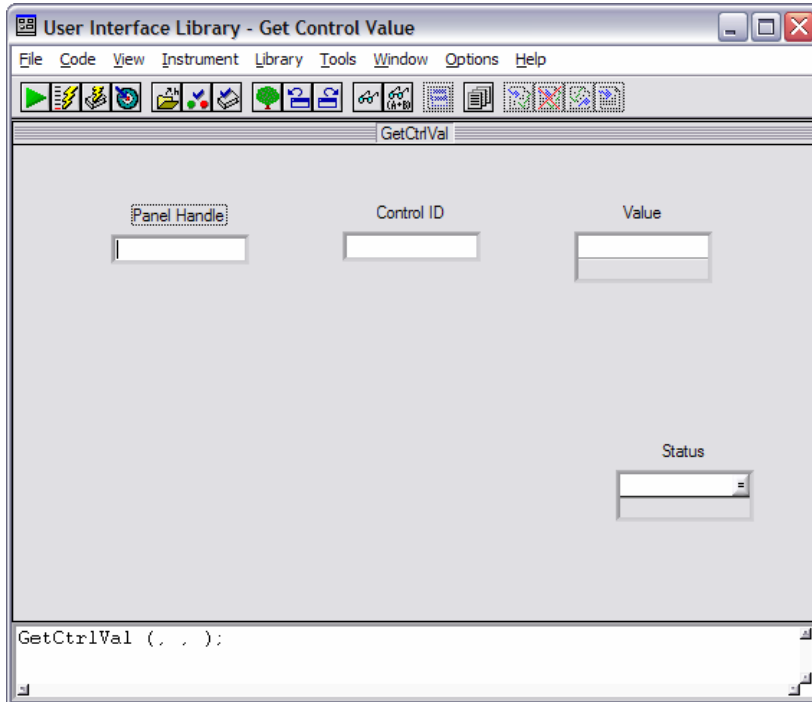
The Bye function will execute whenever the Quit button is clicked on. It runs the function QuitUserInterface, which terminates the program. The add\_one function does nothing yet, but we will use it later.

### 3. Writing a simple program

Now we will make use of our new powers to create a slightly less useless program. The “add\_one” button will serve to add one to a number that is being displayed. To do this, in addition to the command button, we also need to display a number. We can add a display to the panel by right-clicking and choosing numeric (the first choice). Double-click on this to set its properties. By default it is a control, but we want it to be a display, so change its Control Mode to indicator. Our program needs to start with a number, so for our input we need a numeric control whose Control Mode is “Hot”. Add this to your panel, so we have a numeric control, a numeric display, and the “add\_one” button. The variable types should be int for both the control and display. We want to input a number (in this case an integer), then click on add\_one and have the computer add one to the input number. Clearly the input number needs to have a variable assigned to it. This should be done whenever a user types in a number, so the control needs a callback function. We’ll call it input. Name the callback function and generate the control callback in the C code, as we did for the command button.

NOTE: Whenever you write a section of code, it is very useful to insert a comment indicating the purpose of the code. This is done by first typing “/\* comment comment comment \*/”. CVI will change commented characters green to indicate that this section will not be executed.

Now we have our user interface and skeleton code. However, we have to tell the program what to do when a user enters a number. When a number is entered, the “input” function will execute. We can enter code in between “case EVENT\_COMMIT:” and “break;”. How do we obtain the information from the front panel and assign it to the variable x? We need a special function from the CVI library. On the left column, you will see a folder named “Libraries”. Under Controls/Graphs/StripCharts and General Functions, you will find many useful user interface functions, including “GetCtrlVal”, which we can now use to get the value that the user has input to the control. When you select GetCtrlVal from the library, you will get a new window with boxes for variable names, as shown below.



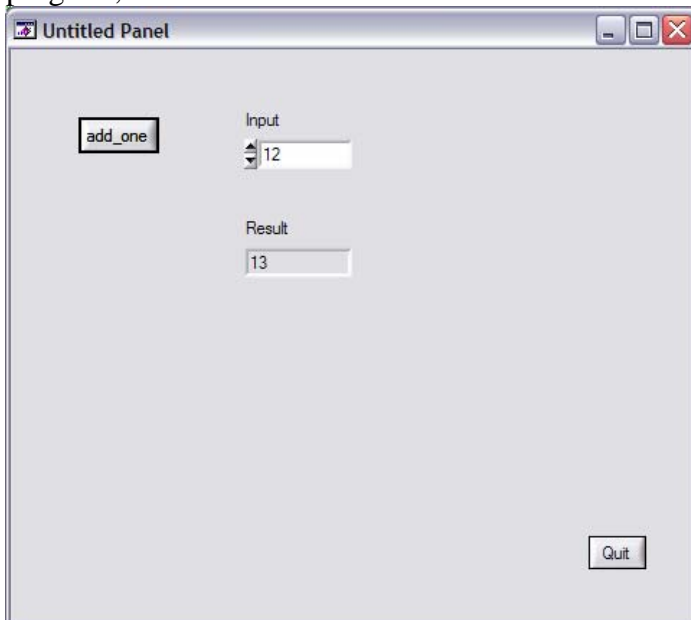
When you type variable names in the boxes, they appear in the correct syntax at the bottom of the window. This text can be copied directly into the program. Alternatively, CVI will insert it into the currently selected location in the C code if you click on the Insert Function call button (second from the left with the lightning bolt.) How do you know what variables to put in each box? Usually this can be deduced from the context or from the explanation obtained by right-clicking on each box. In addition, by clicking in a box and then choosing Code -> Select UIR constant or Select variable, one obtains a choice of currently defined variable names that can be used in the box. Thus, clicking in Control ID and choosing Select UIR constant generates a list of constants that defines the command buttons and numeric controls that are currently on the user interface.

The library can also be used to define new variables. We would like to get the value of the number in PANEL\_NUMERIC\_2 (or whatever control runs the input function) and assign it to x. We fill in the Panel Handle and control indicator from the select variable and UIR constant. Then we put x in the variable box and choose “declare variable” from the library buttons. x is declared and appears in the box as “&x”. This function requires a pointer rather than a variable. For our purposes, it is sufficient to let CVI decide whether

it needs a pointer or a variable, and this does not affect our program. Now you can insert the completed function call.

Now each time a number is typed into that control, the value of  $x$  will be updated. To add one to  $x$ , we need the `add_one` function. We can just add this code directly. It will read simply `"x=x+1;"` and this will appear after `"EVENT_COMMIT:"` once again.

Now each time the `add_one` button is clicked,  $x$  will be incremented by one. To display the resulting value, we need the `"SetCtrlVal"` function from the library. This can be inserted below `x=x+1` so that it updates the screen each time the number is incremented. You should be able to run your program and add one to any number that is input into the box. An example of a running program is shown below. It's not yet a very useful program, but it does make use of several of the functions we will need later.



#### 4. Displaying input from a file

We will now write a new program to input data from a file, display the data, and manipulate the data. The program written in section 1 should be considered practice and does not need to be described in your lab report. However, the program written now will be needed for your lab report.

To input data from a file to be analyzed, the user will need to tell the program what file to take data from, and what to do with the data. The user will probably in general know what is in the file, but the best thing to do is to immediately plot the data so that the user knows what was actually in the file. ***Therefore, in addition to your Quit button, you will need a command button to select and input a file and another command button to display the file.*** Having a separate button to display the data will be useful later when the user makes changes to the file. You have already created C code that creates a function that will be run when the user clicks on it. Now we want to have this button input data from a file. For this, use the library function `"FileSelectPopup"`. This function allows you

to select a file and saves the path to that file into a character array. 300 characters should be sufficient for the array. On the next line, you will need to read this file and save it to an array. You may do this in any way you like, as long as it works.

One simple library function that will do this is the “FileToArray” function. However, this will save your two-dimensional x-y data into a single array, with the x array first followed by the y array. To use this you must have the program count the number  $n$  of elements in the array and divide by two, then write two arrays of length  $n$ . These two arrays can be used for plotting.

After you have figured out how to input data from the file into an x and a y array, you can plot the data using the PlotXY function from the library. Note you will also need a “graph” object on your .uir panel. This object does not need a callback function, as it is just an indicator.

The instructor will provide you with a file containing data to be analyzed using your program. Make sure your program properly displays the data before continuing to make your program more complex. At this point, make sure all of the code is commented and that you have saved your application to a unique filename for your group. If it worked, while your program is active press “alt-print screen” and a picture of your results will be saved to the clipboard in Windows XP. Open an MSWord document and paste the picture into your document by choosing paste from the edit menu. The picture you have created is referred to as a “screenshot”. This is your proof that your application works. Make sure the application has your name and your partner’s name on the front panel so we know it was your application.

## 5. Subtracting baselines and normalizing data

Two common operations to perform on data are normalization and baseline subtraction. Data obtained from a detector often has an unchanging background that should be subtracted to obtain the actual signal. If this is done regularly, it is nice to build such a function into the virtual instrument panel. ***To do this, you will need to add a button that subtracts a baseline and a numerical input that determines what amount to subtract from the signal.*** Label this section of your VI “baseline”.

Data is often normalized in two ways. Sometimes it is normalized so that the maximum value of the data is one. This can be unreliable, and it is common to instead calculate the area under a measured curve so that it is equal to one. This is the normalization we will use. Note that it is important to subtract a baseline before normalization. To calculate the normalization constant, you must numerically integrate the data. The simplest numerical integration, which we will use, is to take your data, which is in intensity as a function of some variable such as position or time, and sum the product of each measurement times the bin width (which may be in meters or seconds or some other useful unit). ***Your program should have a button that normalizes the data and a numerical output that displays the factor by which the data must be divided for normalization, which***

*represents the area under the curve.* The displayed result should give a plot of data that sums to one when integrated.

## 6. Finding the centroid of the data

When analyzing image data or other data that contains peaks, it is often desirable to find the “center” of a peak. However, it is not always clear what one means by the center of a peak. Is it the highest point, or the mean value? One robust definition of the center of the peak is referred to as a centroid. This is useful in imaging because many objects that are smaller than optical resolution are detected as a larger sphere, and the location of such an object can be identified to higher than optical resolution by identifying the centroid of the object’s image. Thus, for example the position of a micron-sized polystyrene bead can be measured with an accuracy of 10 nm using image analysis.

The final capability you should add to your program is the ability to identify the position of the centroid of the data. The centroid is defined as the center of the area of the peak. In other words, the centroid is the position at which half of the total area under the data is reached. To calculate this, your program must first calculate the entire area, which is done already for normalization. Your C code must then calculate the area again, but within the loop that calculates the area, there must be a conditional statement that identifies the position at which this area reaches one half of the total area. *Your program should numerically display the value of the centroid of the data after the “calculate centroid” button is selected.*

## 7. Data extraction: writing your data to a file

When taking data in experimental physics, it is never enough to have a program that displays the raw data. You need to create a file that contains the raw data so you can take that data later and manipulate it using MS Excel or some other analysis program. Here you will modify your program by adding a “save to file” command to your virtual instrument. Find from the library a function that will allow you to save your array to a file (such as ArrayToFile). Write your program so that it outputs two columns for each data set. In Excel, each column is separated by a tab. Repeat the experiments above for one input file and plot the data in Excel. Label your axes and add a title. Save this file to import into your lab report later.

## 8. Writing your lab report

One of the most important skills for an experimental physicist is the ability to communicate your experimental results. Your lab report should describe your experiment step by step and relate the experimental results and methods quantitatively using words and graphic illustration. This report should contain an introduction describing the goals of the experiment and the methods used. This should be followed by a “Results and Discussion” section, with a description of the methods used and the data analysis program you wrote. It should contain a picture of the program and a screenshot of the virtual instrument you have created. Screenshots of the results when subtracting a

baseline, normalizing data, and calculating the centroid should be presented in your report. A graph made in Excel or some other graphing program (gnuplot, Kaleidagraph, etc.) showing the initial data and final result should also be presented. You should discuss the values obtained in the graph and how the initial and final data relate to each other. Finally, your conclusion should state what you have accomplished with the program.