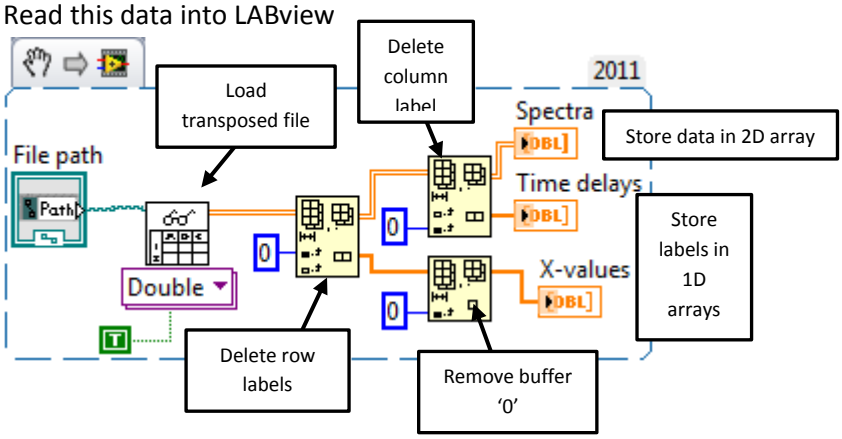
LABview Tutorial Contact: michael.grubb@bris.ac.uk

Project goal: Design and create a LABview program to read, baseline, and save a set of transient absorption data.

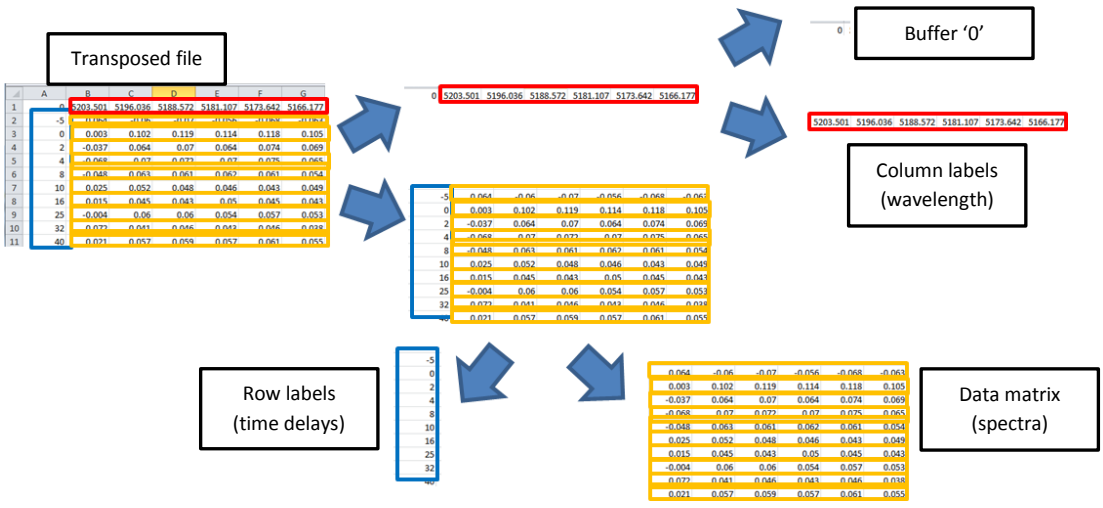
1. The data file of interest has the following format:
   Time delay in ps, wavelength in nm (common x-axis to all plots), absorption in mOD (for each time delay)

	A	B	C	D	E	F	G	H	I
1	0	-5	0	2	4	8	10	16	25
2	5203.501	0.064	0.003	-0.037	-0.068	-0.048	0.025	0.015	-0.004
3	5196.036	-0.06	0.102	0.064	0.07	0.063	0.052	0.045	0.06
4	5188.572	-0.07	0.119	0.07	0.072	0.061	0.048	0.043	0.06
5	5181.107	-0.056	0.114	0.064	0.07	0.062	0.046	0.05	0.054
6	5173.642	-0.068	0.118	0.074	0.075	0.061	0.043	0.045	0.057
7	5166.177	-0.063	0.105	0.069	0.065	0.054	0.049	0.043	0.053
8	5158.712	-0.073	0.113	0.062	0.063	0.058	0.043	0.039	0.056
9	5151.248	-0.041	0.109	0.072	0.071	0.059	0.047	0.046	0.059
10	5143.783	-0.077	0.112	0.065	0.065	0.059	0.04	0.038	0.055
11	5136.318	-0.049	0.114	0.069	0.07	0.053	0.054	0.046	0.055
12	5128.853	-0.057	0.11	0.066	0.073	0.06	0.051	0.044	0.057
13	5121.389	-0.054	0.113	0.071	0.067	0.059	0.054	0.051	0.06

2. Read this data into LABview

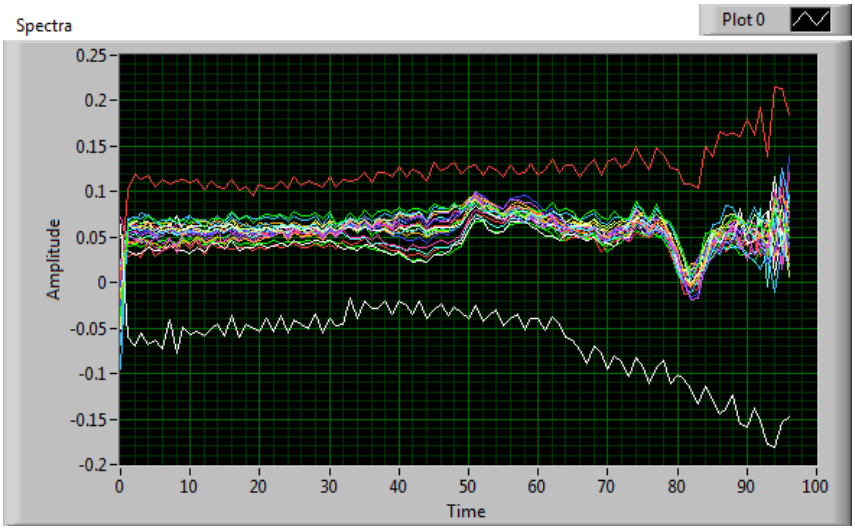


Schematic of file manipulation:

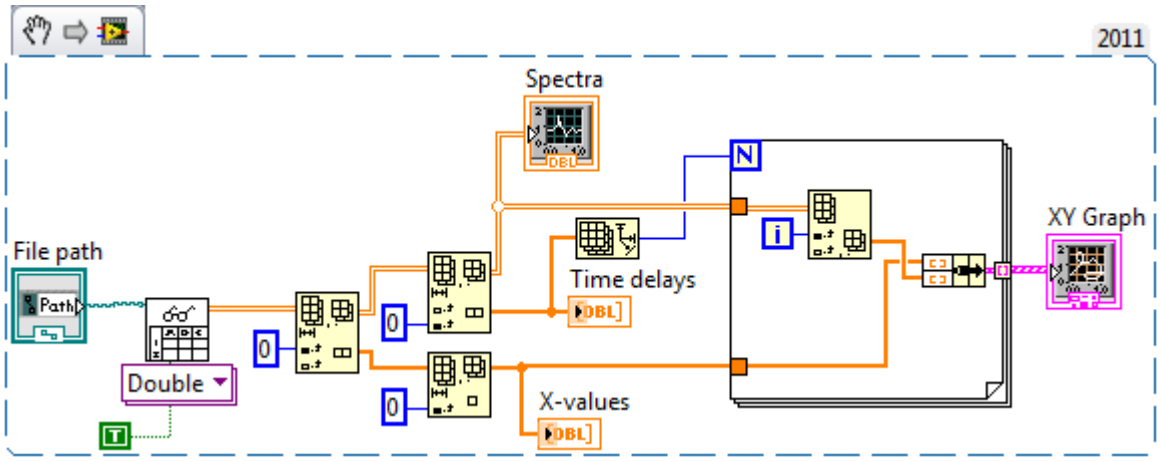


3. Now that we have read the data from the file, we want to display the data. If the data has equally spaced X-axis intervals, you can simply use a waveform graph. In fact, the data type

of a waveform graph is a 2D array of DOUBLE type (decimal numbers), so we can use a waveform graph to 'store' the data instead of a normal 2D array. This allows us to visualize the data, but note that the X-axis is only counted by array index and not wavelength.



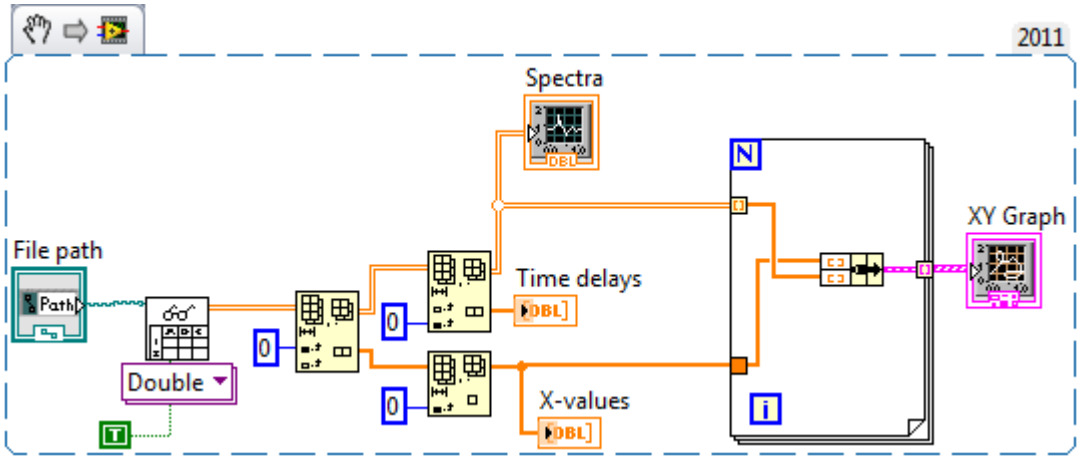
4. A more general way of displaying X-Y datasets is using an XY graph. The data-type of an XY graph is a 1D array of clusters, with each cluster composed of an array of the X and Y data respectively. This means we have to take each row of our data matrix, and 'bundle' it into a cluster with our x-axis labels. The following code achieves this:



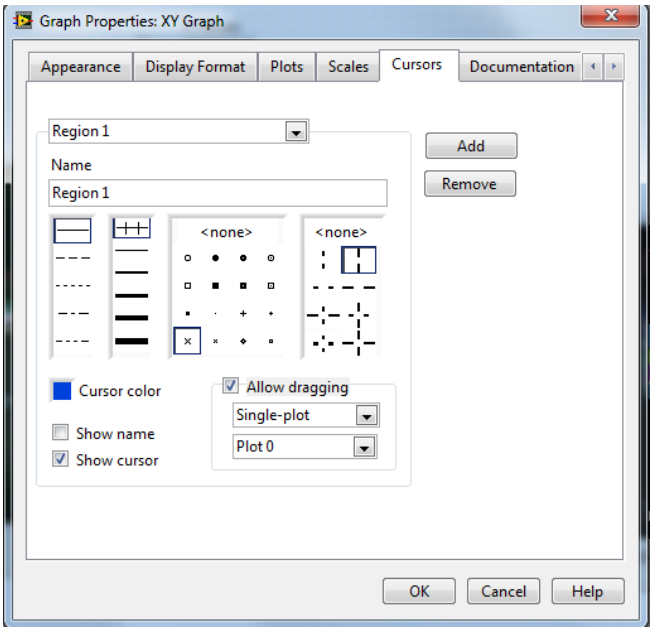
We access each row of the Spectra data matrix using a 'for' loop, which will run whatever code is placed inside of it 'N' times (in this case, the number of time delays). We then want to take each individual spectrum in our data matrix (row) and pair it with the X-labels. 'i' is the current iteration of the loop, so the first time the code is executed it returns 0, the second time 1, and so forth all the way up to N-1. Using 'i', we can access each row of our data matrix. The clusters we create during each iteration of the loop will be 'auto indexed' into an array of size N, and is now in the proper data-type to become an XY graph.

An easier way to write this same code is to auto-index the data matrix into the for loop, which will automatically set N equal to the number of rows in the matrix, and applies the

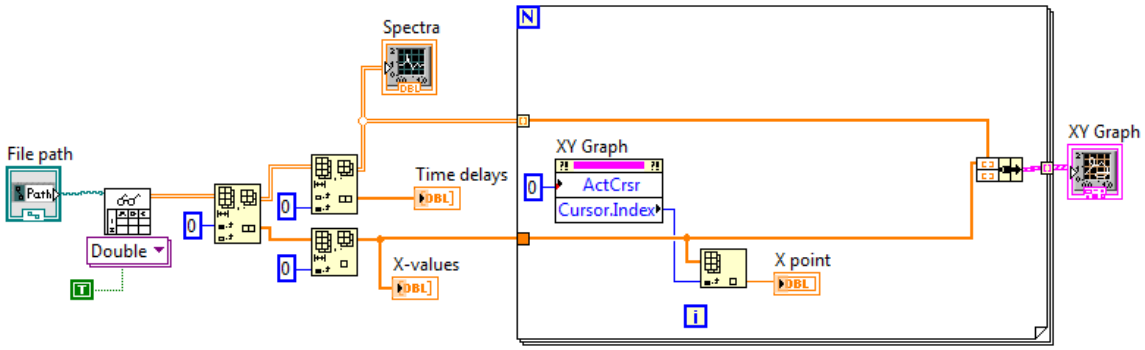
loop code to each row:



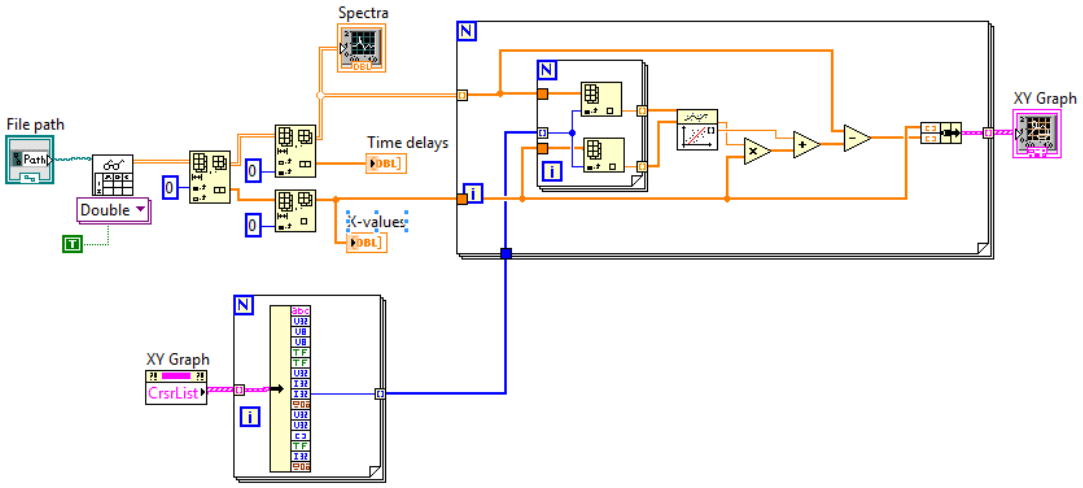
5. Next, we want to be able to baseline the spectra, which we notice are all offset from each other. We want to be able to select points on the spectra that we know should be zero, fit a line through those points, and subtract that line from the data. We will make vertical cursors to define the points. To do this, we go to the 'Cursors' tab of the XY graph properties and add a new cursor with the following parameters:



6. Make several cursors. Now we want to grab each data point the cursors are pointing to, and fit a line through them. Properties of front panel objects, such as cursor positions, can be accessed programmatically on the block diagram using 'Property nodes'. The following code reads the index of the array 'Cursor 0' is pointing to, and displays the X-axis value at that index.

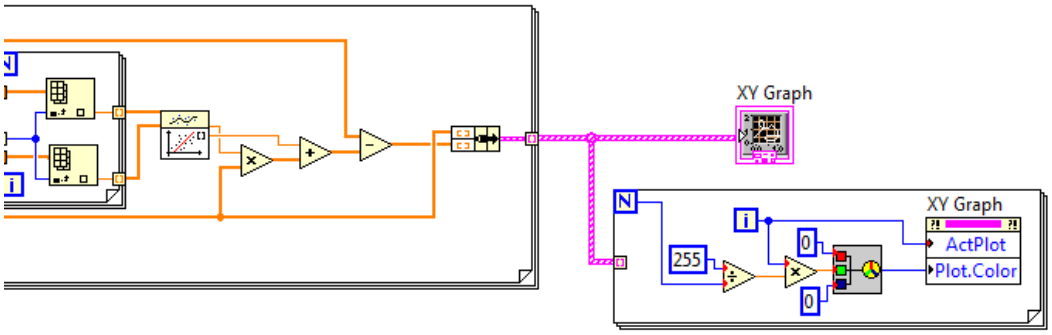


7. Next we want to read all the x-y position of all the cursors for each spectrum, fit a line through them, and subtract that line from the spectrum to re-baseline it. The following code achieves this, using the handy 'Linear Fit VI' from the mathematics tab:

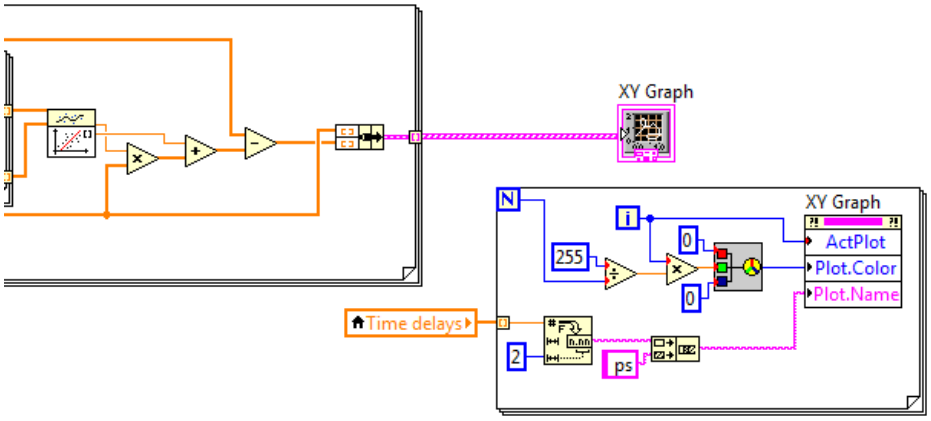


8. Running the program continuously now allows us to optimize the base-lining in real time. However, using LABview's default color scheme, it is difficult to see the time evolution of the spectrum. We now want to make the spectrum of each time delay be plotted as a different color, increasing from black to bright green as the time delay increases. Again, we will use a Property node to programmatically change this color. Colors can be expressed by RGB values, which is described by 3 unsigned 8 bit integers (0-255) describing the relative contribution of red, green, and blue to the color. Since our simple color scheme is a green-scale, the red and blue contributions will always be zero. The green contribution we want to be 0 for the first time delay, 255 for the largest time delay, and evenly distribute the

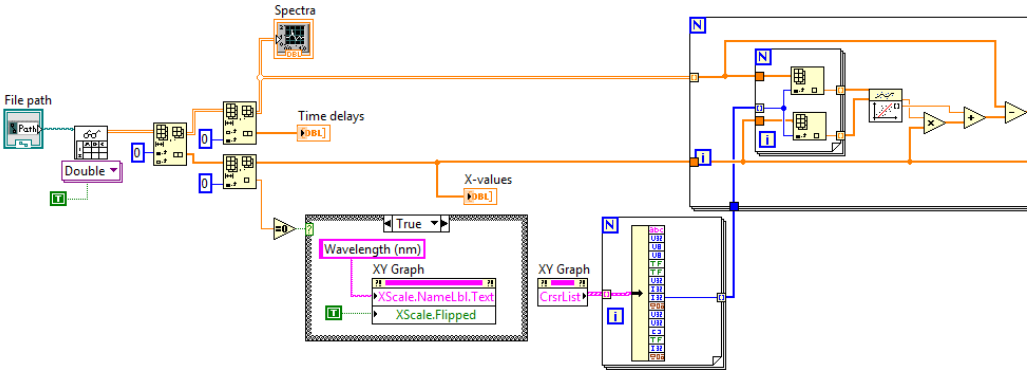
remaining time delays between those 2 values. This is accomplished by the following code:



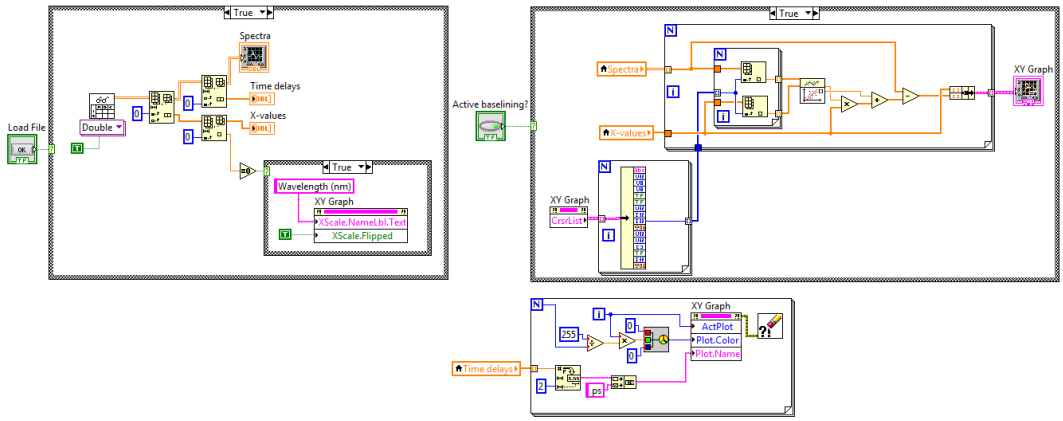
9. Next, we want the plot legend to show what color corresponds to wach time delay. Again, we will use property nodes, but first we must change our array of time delay numbers from the DOUBLE type to the STRING type.



10. Remember that buffer '0' in the first cell of the input file we loaded earlier? We are now going to use that cell in the input file as a code for what units the X-axis is saved in. If it is a '0', the X-values in the file are in wavelength in nm, and if it is a '1' or anything else, we will assume the units are reciprocal centimeters. To do this, we will use a case structure. Case structures allow you only execute a certain block of code if a certain condition is met.

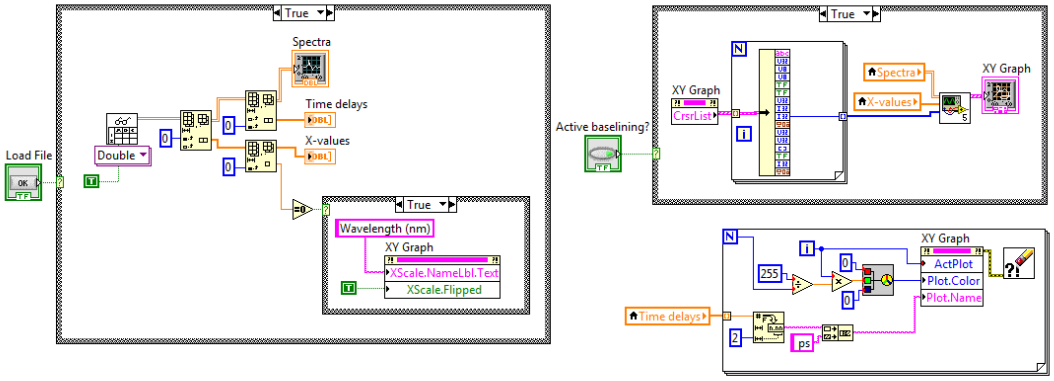


11. Currently, continuously running the program, we are re-reading the file from the hard drive with each iteration of the program. This is very inefficient and will make your program very slow for larger data sets. We now want to make the program run such that the data is only loaded from a file when we press a 'LOAD' button. We also want a switch that can turn the constant baselining on and off, incase we want to save the computer's processing power for something else. This idea will allow us to learn a method of structuring our code that better responds to more complicated front panel interfaces. There are two main methods: by using event structures or case structures. We will use case structures, for now.

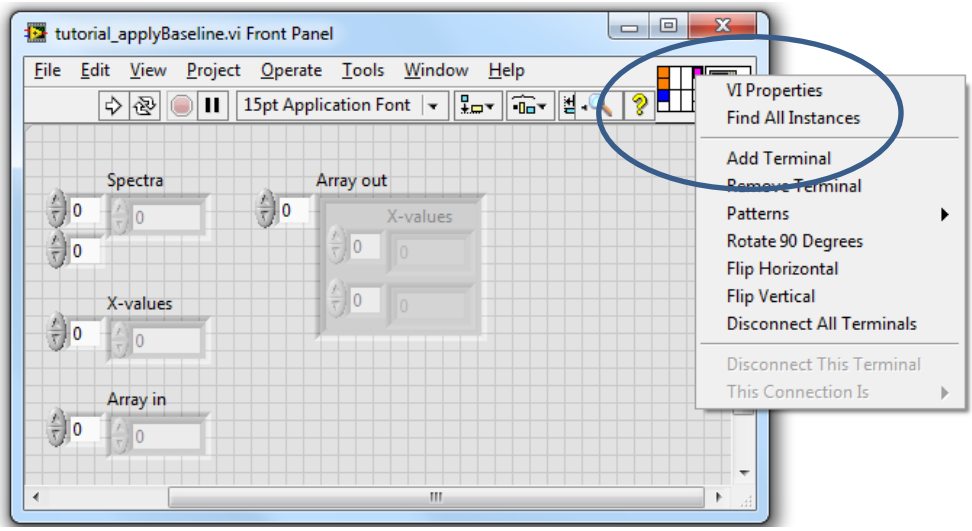


Note that although buttons and switches are both booleans and look identical on the block diagram, they have different default 'latching' properties on the front panel. When you click a switch, it will switch between true and false, and remain that way until you click it again. The default latching of buttons, however, is to change from false to true until the VI reads the true value once, and then it automatically switches back to false.

12. We notice that our program is growing larger, and also that spectral baseling is code that we may want to use again elsewhere in our program as it grows. Rather than copying and pasting such a large code block, which is redundant and makes our program more difficult to read, we will create a new subVI. To do this, simply select the block of code you want to make a subVI, and then go to the menubar and click Edit->Create subVI.

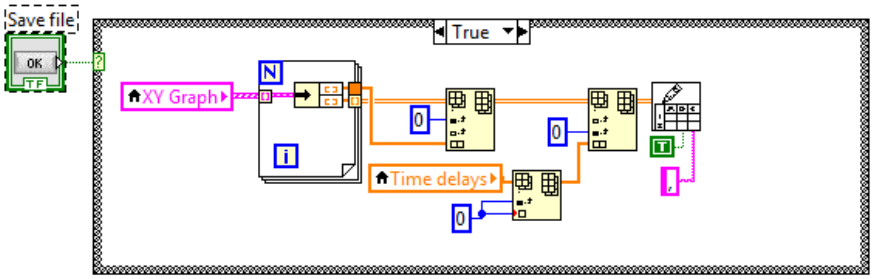


Inputs and outputs of subVI's can be edited by adding and removing terminals on the VI icon in the upper right of the front panel window.



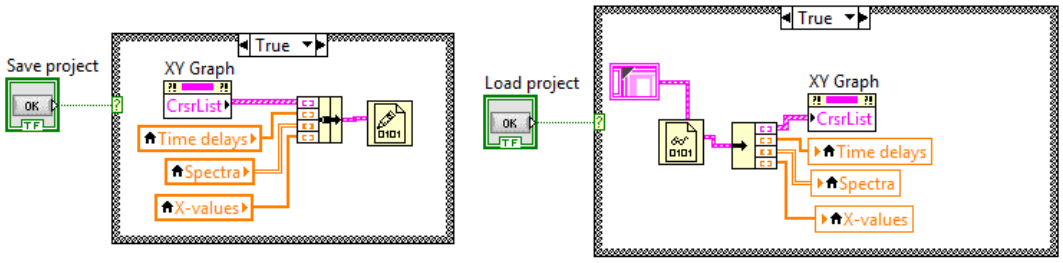
Although subVI's make the front panel of your main VI cleaner, making everything into subVI's can make your program very hard to read, as you have to each subVI in a new window. SubVI's should be used when you have a complicated process that you need to use several different places in the same program, or when the process is general enough that you might wish to use it in other programs as well.

13. We now want to add the ability to save our processed specrum. We can choose any format we like for saving the processed data, but lets assume we want to use the same format as the input file where the columns are time delays and the rows are the wavelength values. Just for something new, lets suppose you have another program that only reads in *.csv files (comma delimited instead of tab delimited). This can be written as follows:



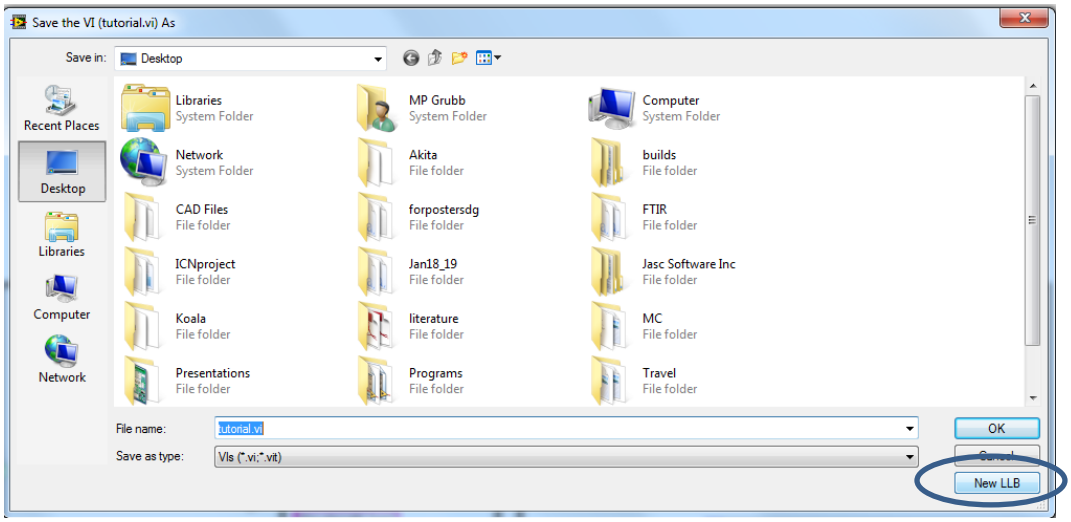
14. Finally, we want to add the ability to save our processing parameters such as what points we used to baseline and the pre-processed data. We also want the ability to reload these parameters into our program so that we can do more processing later. To do this, we will save a binary file. Binary files have the advantage of being able to save any combination of data types in any format in a very simple manner. The disadvantage is that as an encoded binary file, you will only be able to read and interpret this file using LABview.

    To save a binary file, simply bundle the data you want to save in a big cluster:



To load a binary file, you must input a cluster of the same format to the 'Read Binary File' VI so that it knows how to interpret the data it is reading. The easiest way to do this is simply to create a constant from the wire going

15. Now that we have a subVI, our program is composed of two separate files. This can become difficult to keep track of very quickly. There are two separate ways to organize LABview Vis: Projects and LABview Library files (LLBs). Projects are strictly organizational; you can 'point' to the location of the VI files you are using on your computer. Make a new project of the Vis you currently have open, and their dependant Vis, by clicking File -> New Project in the menubar. LLBs package your main VI and all the dependant subVI's into one library file on your computer. They can be particularly if you need to send the program to someone else, and only want to email them one file without worrying about forgetting include a necessary subVI. They can be created through File -> Save As and then clicking the 'New LLB' button.



16. Our example program is now complete. Here are some suggested additions to the program you can try to make on your own:

-Allow baseline regions to be selected instead of single baseline points
-Include the ability fit a polynomial baseline instead of a linear one
-Add a kinetic trace plot that graphs the integrated absorption intensity of a defined spectral region vs the time delay.
-ADVANCED: Some of these data files include multiple spectra at the same time delay from different measurement cycles. Make your program identify these spectra and average them together.