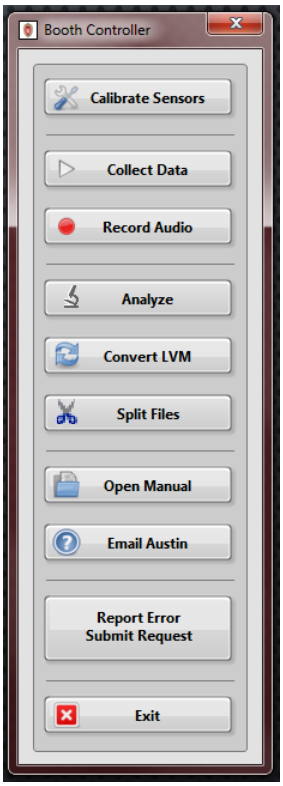
BOOTH PROGRAM

PURPOSE AND USE



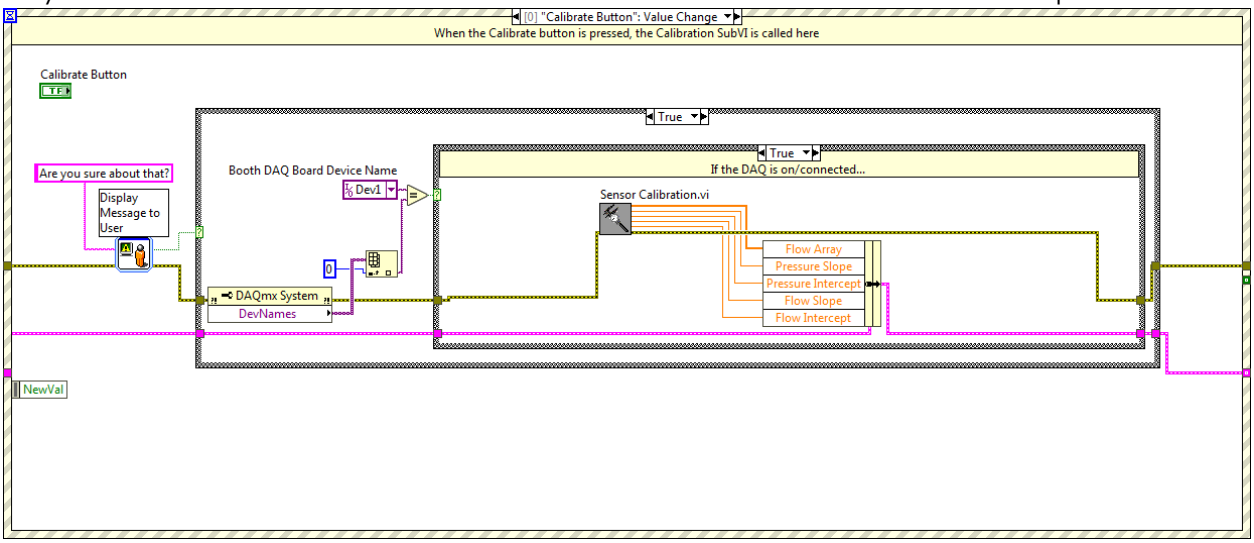
The Booth Program, marked by the neat little UW Crest in the taskbar, is used to collect flow, pressure, EGG, and acoustic data on excised larynges within the booth. Unless changes need to be made to the program, you can and should open it through the shortcut. Currently (as of 3/19/2019), all the program files are saved in <G:\1-Booth Program Updated – 2017>.

Once the main controller is opened, you can choose one of the 10 options listed. If you hover over the button, a short description of its function is available.

If you open the program through the LabVIEW project, you will have access to the block diagram. Note that you will need the developer password before you are able to actually open it.

Before making edits to my program, you should first have a basic understanding of the more advanced methods that are sometimes used in my programs. For example, you should understand what an event structure is and how works…

Hey look! An event structure! Below are the different events that this is setup to handle:

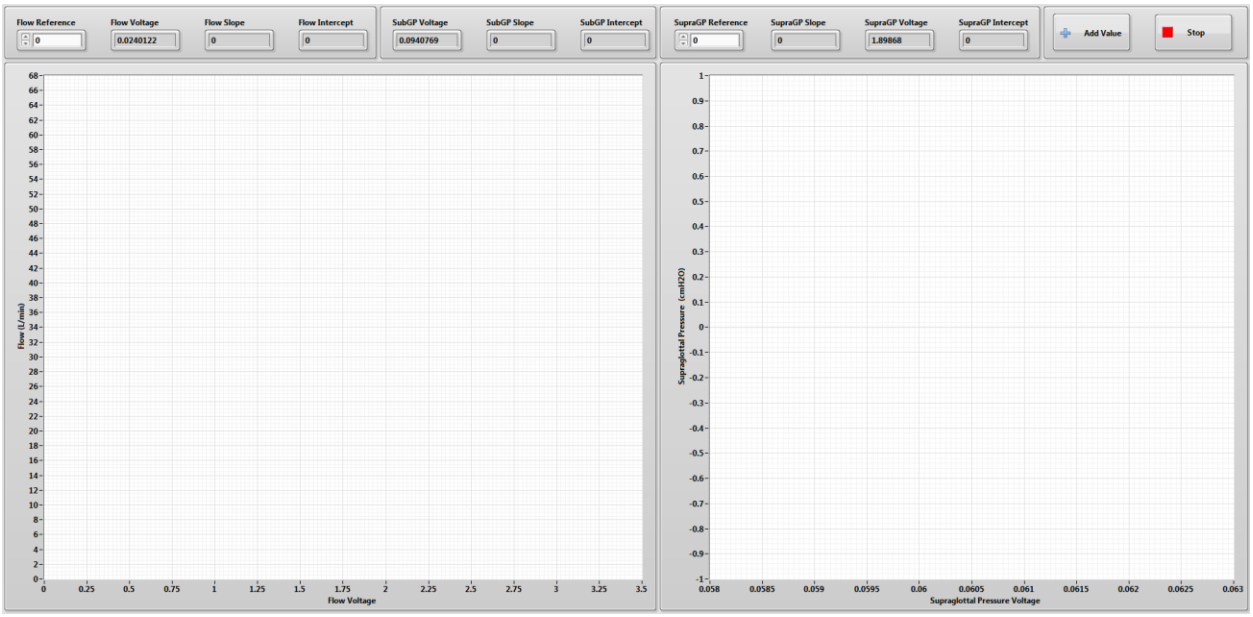


- **Calibrate Button** will open up the uh… calibration SubVI. All the other stuff in this case is to check if the user actually wants to open calibration (this is to avoid accidentally erasing the calibration values) and check if the DAQ board is connected. Once the SubVI closes, it will output the Flow Array, and the new slope and intercept values. They are also written to text files. (See calibration section)

- **Collect Button** will open the data collection SubVI in order to start reading from the sensors. It also reads in the previously collected slopes and intercepts. The extra stuff here checks if the DAQ board is connected and the when last time calibration occurred. The user cannot continue if it has been more than a week.
- **Record Audio Button** will open a data collection SubVI similar to the main collection program that only records acoustic signals. There is also a check like the calibration and collection cases to see if the DAQ is connected.
- **Analyze Button** opens a couple SubVIs to analyze and save calculated values.
- **Convert Button** allows you to choose a folder full of LVM files and convert their recorded acoustic data into WAV files.
- **Split Button** runs a python script written by Kieran Paddock that splits LVM files if they are too long to be run by the Analyze SubVI.
- **Help Button** opens up the Booth Manual (not this document that you are reading but one that Graham Johnson wrote in 2015).
- **Email Austin Button** sends me an email to let me know that there is a problem with the booth. I will respond as soon as I can.
- **Report Error/Request Change Button** opens up a Google Form where someone can report errors or request changes to one of my LabVIEW programs.
- **Exit Button** does what you think it does.
- **TIMEOUT** if nothing happens for 20 minutes, this case will be triggered. The user will be prompted to check if they are still there.

CALIBRATE SENSORS

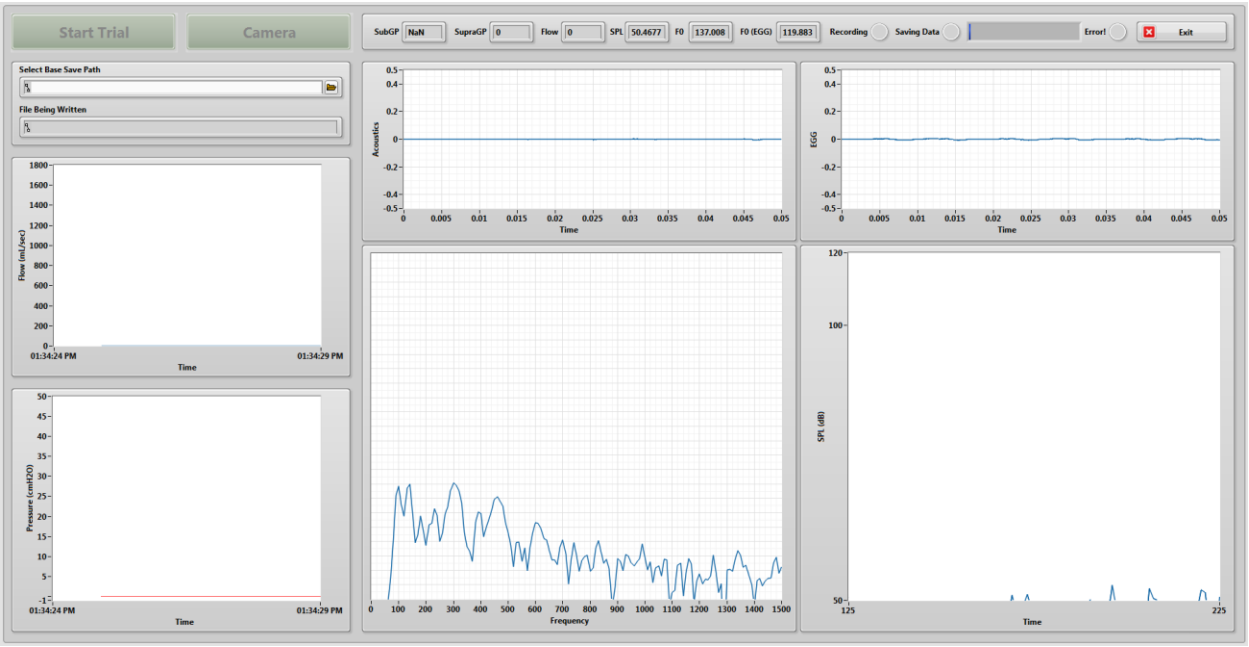
When the calibrate sensors button is pressed, it will open this SubVI. This is basically the same SubVI as all my other calibration programs (See the *Other* section for more details). While the block diagram may be slightly different, the use is the same.



You will need to provide a known pressure and flow to their respective sensors, and set that as the *'Reference'* in the program. Then click 'Add Value' to… add that value to the plot. While you technically only need 2 points to make a line, it is better to do at least 5 to get a good linear regression. As of 3/18/19, you only need to provide reference pressure values to the supraglottal sensor.

COLLECT DATA

Once the collection SubVI is opened, it will start running automatically. You should see data on all the front panel charts. If you turn up the flow, you should see a response in the flow chart (top chart on the left side) as well as the numerical readout on the top bar. You can do the same thing for pressure. If both of them show up as unchanging zeros, you might have overwritten the calibration files and will have to re-run calibration or move previous files into the Calibration Files folder. If only one does not response, there might be a problem with the connection to the DAQ board. On the right ¾ of the front panel, you will see, an acoustic intensity graph, an FFT display where you should see frequency peaks if you play a tone into the microphone, and EGG intensity graph, and an SPL chart which should respond to changes in sound intensity.



You will notice that the 'Start Trial' button is initially disabled. This I because you must select a place to save you data prior to collecting said data. Do this by clicking on the folder icon to the right of "Select Base Save Path".

After a path is selected, you are ready to collect data. Presumably, you have your larynx mounted, tested it to see if it will phonate at all, and have the high-speed camera set up if you would like to record videos. All you have to do to record flow, pressure, and acoustics is press 'Start Trial'. The button should turn red and now say 'Stop Trial'.

Additionally, the Recording light should turn red and the Saving Data light should turn green.

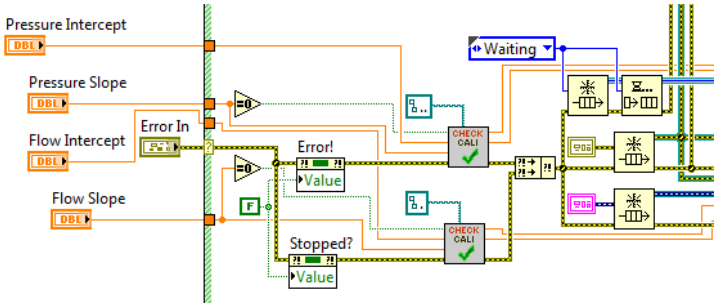
During a trial, you can trigger the camera from the LabVIEW collection program. To have the 'Camera' button trigger collection on the PFV program, you must have the program set to record. You should see 'Trigger In' instead of 'Record' on the live view in the PFV program. Switching between the APX and Multi controls the trigger. Make sure this button matches whichever camera you are using.

Once you have finished recording your trial, clock the 'Stop Trial' button. At this point, there may be data that has not been written to the file. If this happens, the Saving Data light will remain green and the 'Start Trial' button will be disabled. You must wait until the save is complete before closing the program or starting a new trial. This should only take a few moments.

THE BLOCK DIAGRAM

The block diagram for this VI is a little complicated. There are four parallel while loops. One is a state machine/event handler and the other three are set up in a producer/consumer structure with a data queue, a message queue, and an error cluster queue. There are also a variety of SubVIs that are used. I will get into each SubVI after we go through the overarching structures.

OVERARCHING COLLECTION VI



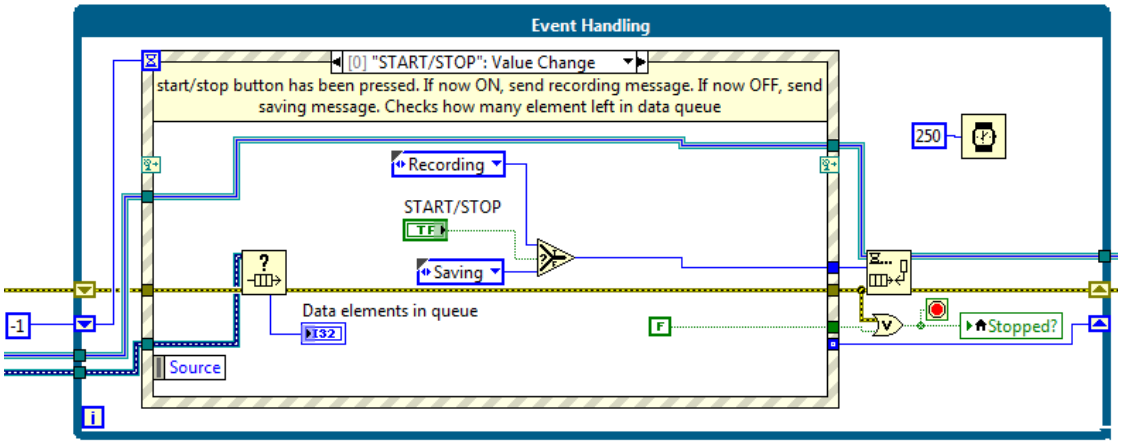
The first thing that happens when the collection program opens is the reading in of the slopes and intercepts from the calibration SubVI. Then, with the property nodes, the 'Error!' and 'Stopped?' booleans are set to be False so that the loops don't automatically stop running.

Following this, there are three copies of the SubVI called "Check for Calibration Values." If the values read in from the controller VI are zero, this SubVI will open up the text files and read the saved calibration values from the path wired in. If they are not zero, they will simply use the values wired in from the controller.

Also before the loops start, three queues are initialized. One is for the collected data (the lower one), one is for messages (the top one), and the other is for errors that might be thrown from the other loops. Notice that the 'Waiting' case is added to the message queue before any loop starts running.

EVENT HANDLING LOOP

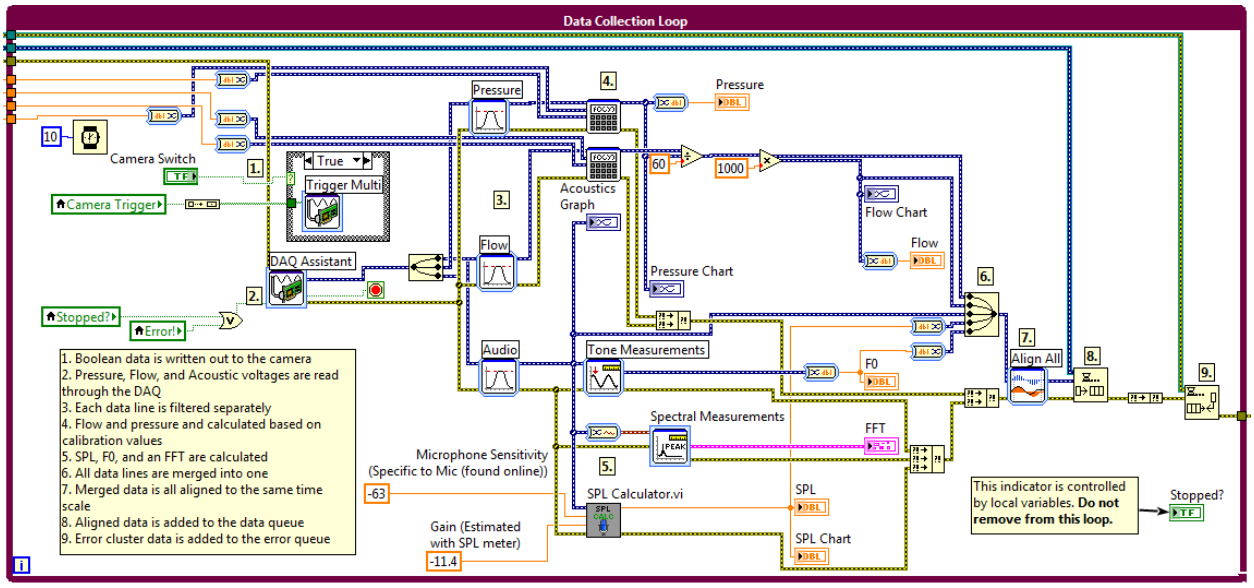
Here is the Event Handling loop. There are seven events handled here. On the next page, I will outline each event and each case for this Loop.



- **START/STOP Button** If the button is changing from False to True (starting trial) the Recording message will be added to the queue. From True to False (stopping trial), the Saving message will be sent. Also, whenever this event occurs, the number of elements in the data queue is checked. This will be used in the data saving loop.
- **Camera Trigger** When the trigger button is pressed, the Recording message is added to queue to make sure that data is being recorded with the camera. It also sets the timeout time to 2500ms by way of shift register. If the trigger is being turned off, it will be set to -1ms, which means that there will be no timeout.
- **Trigger** If no event occurs within the set time, this event will occur. This turns the camera button off again and sets the timeout to -1.
- **Exit** sends the Exit message…duh.
- **Error!** also sends Exit message.
- **Application Instance Close** sends the Exit message when the application is closed.
- **Panel Close?** calls the Exit message when the front panel is closed.

DATA COLLECTION

Here is the Data Collection loop where the voltages are read in from the DAQ and displayed on the Front Panel. The DAQ Assistant Express VI opens the channels to read on the DAQ board and outputs them all in one signal wire. This is then split up and filtered. The pressure and flow signals go through similar low pass filters while the acoustic signal goes through a high pass filter. Also, Boolean data is sent to the camera to trigger it.

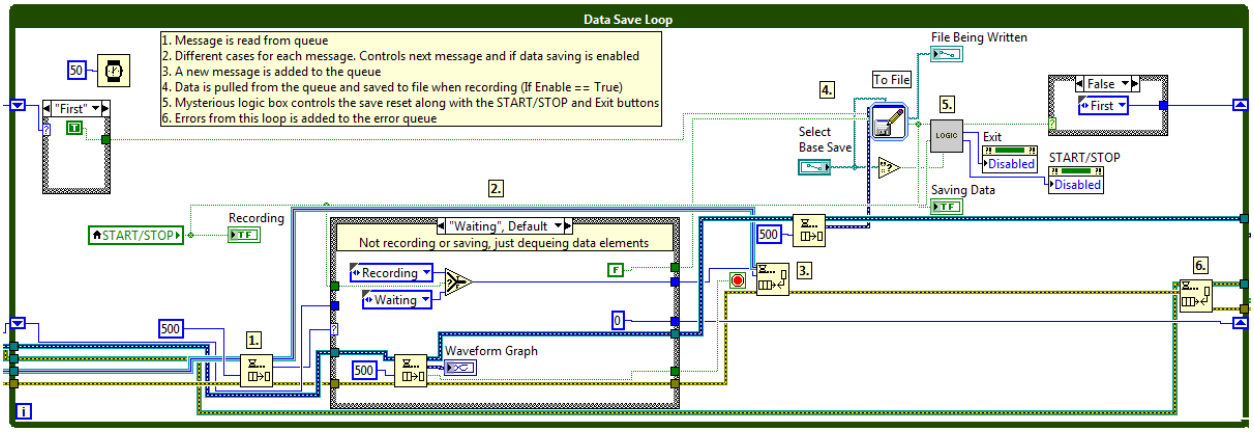


The pressure and flow both need to be translated into their actual values using the slope and intercept found from calibration. This is done through the formula SubVI which just takes the voltage input *(x)* multiplies by the slope *(m)* and add the intercept *(b)* the outputs the actual pressure or flow *(y)*. These new calculated signals are then merged with all the others.

The acoustics go through three different SubVIs, Tone Measurements, Spectral Measurements, and the SPL Calculator. The first outputs the frequency peak of the FFT, also known as F_0. The second just creates the FFT graph for the front panel. The SPL calculator outputs SPL.

All the calculated signals (Flow, Pressure, Acoustics, SPL, and F_0) are then merged together before being aligned together. If recording (start/stop button is set to true), the merged and aligned data will be added to the data queue. If not, nothing happens with the data. The last thing in this loop is the error info is added to the error queue.

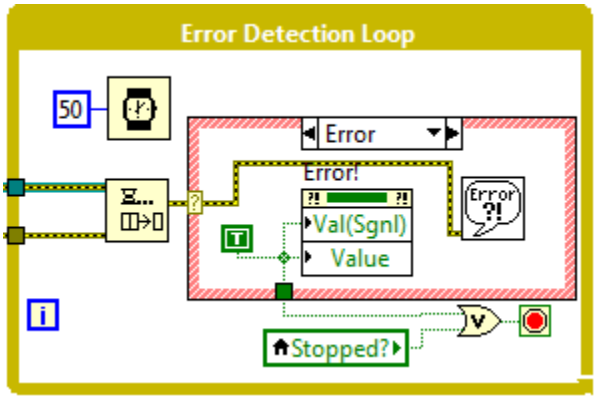
DATA SAVE



This loop contains a state machine that is controlled by the enumerated messages sent from the event handling loop. There are four cases that can be called that dictate what happens in other parts of the loop. There is also a Logic SubVI that controls the disabled status of the exit and Start/Stop buttons, as well as the First/After case structure system.

- **Waiting** is the default case. It is initially called before any loop starts. It will continuously send new 'Waiting' messages while the Start/Stop button is false. It also de-queues elements from the data queue so that there isn't data in the queue when it comes to start recording/saving data.
- **Recording** sends a true value out to enable data saving to file. While the Start/Stop button is true, it will keep sending 'Recording' messages until it is turned off. At this point, the saving case will be called. Finally, this case updates the number of data elements in the queue and stores this value in a sift register.
- **Saving** will continue to send itself 'Saving' messages until there are no more data elements in the queue. This ensures that all the trial data is written to file. After there are no more elements left, the 'Waiting' message will be sent again.
- **Exit** stops the loop.

ERROR DETECTION

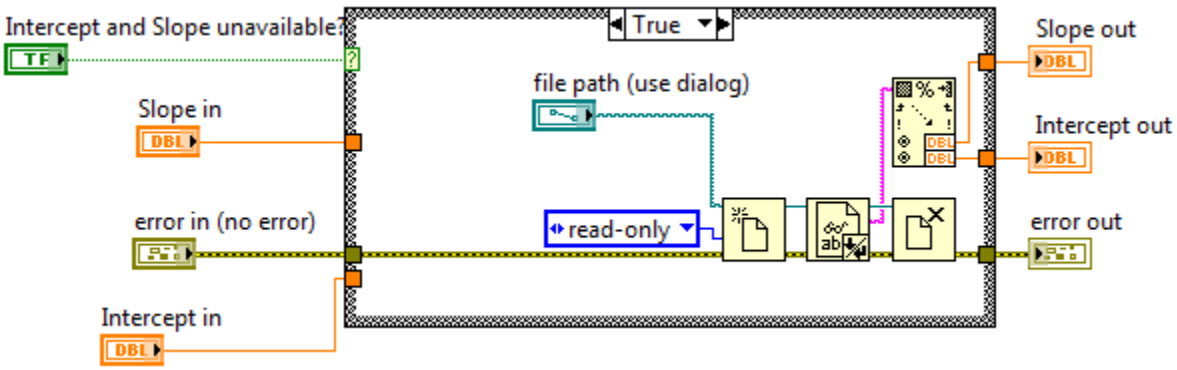


The Error Detection Loop pulls the error cluster data from the Data Save and Data Collection loops. If there is an error read from either loop, the Error! light is set to True and the loop is stopped. With the change, the other three loops will receive signals tell them to stop.

An error message will then pop up through the Simple Error Handler SubVI.

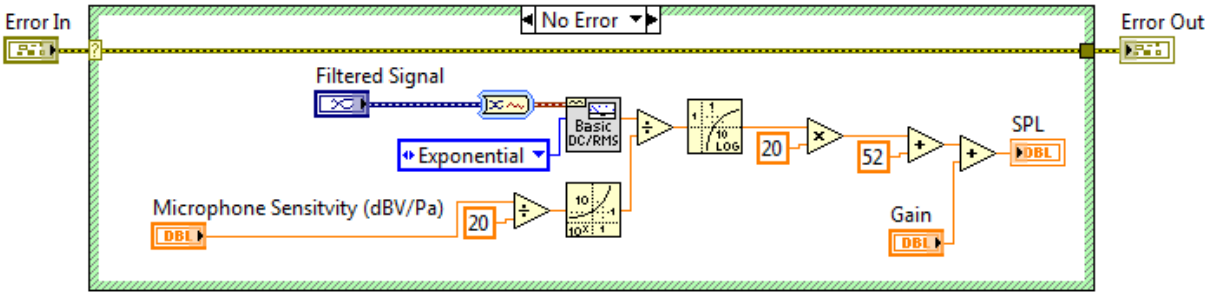
Finally, once the Exit button is pressed, or an error occurs, all the loops will be stopped. Then the three queues will be flushed and released and the Collection SubVI closes.

CHECK FOR CALIBRATION VALUES SUBVI



If there is no value read in from the Controller VI, calibration values will be read from a pre-saved TXT file. The file path is read in from a constant and opened with the Open/Create/Replace File function, the string is read using the Read from Text File function. The file is then closed and the string is converted to two numeric (DBL) values.

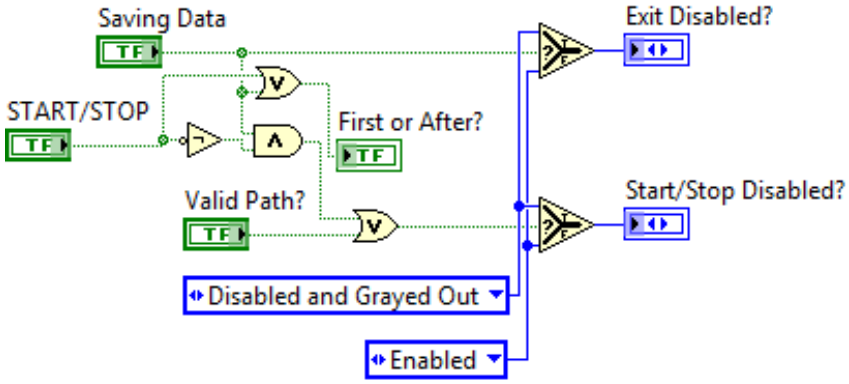
SPL CALCULATOR SUBVI



$$SPL = 20log\left(\frac{vRMS}{RMS_{ref}}\right) + 52 + gain \qquad RMS_{ref} = 10^{(sensitivity/20)}$$

LOGIC SUBVI

This is the logic controlling when the Exit and Start/Stop buttons are disabled.



If Saving Data is set to true, then the exit button will be disabled.

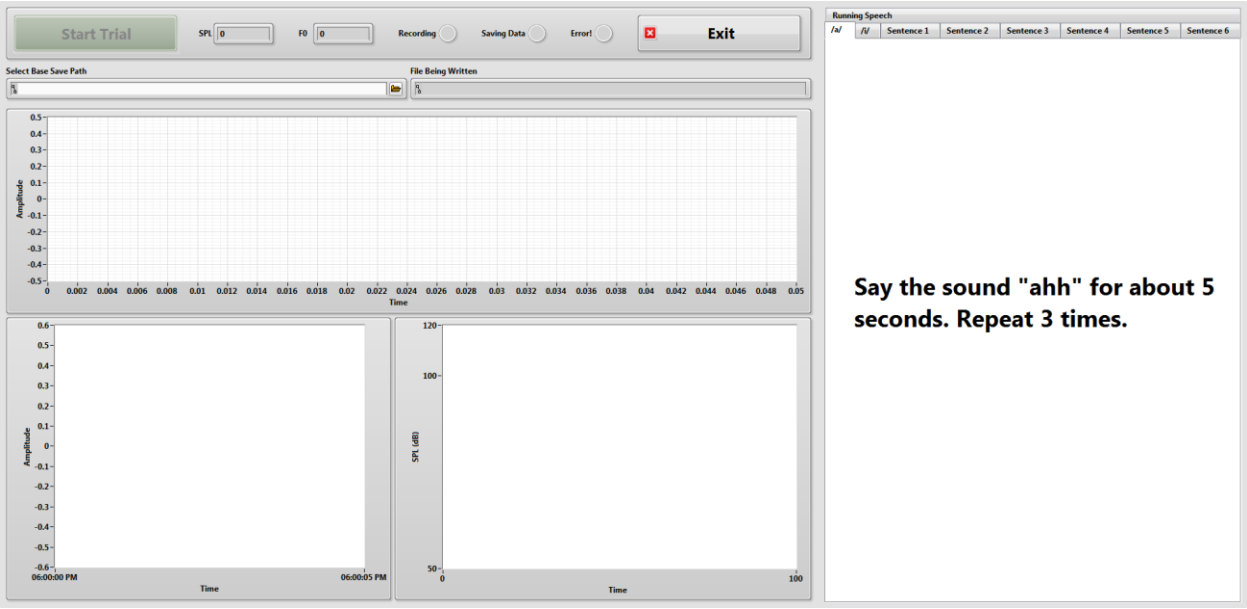
If the Start/Stop button is set to true, or data is being saved, then false will be sent to the First/After case structure.

If data is being saved and the Start/Stop button is ***not*** true, or the Base Path control does not contain a valid path, then the Start/Stop button will be disabled.

RECORD AUDIO

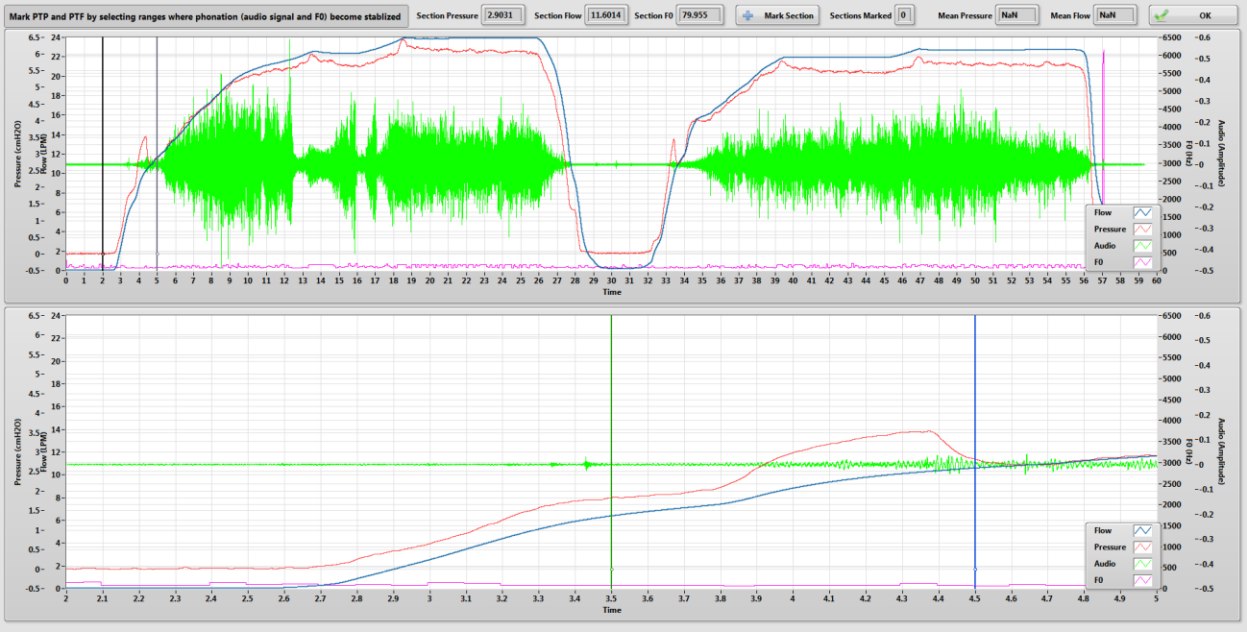
Clicking the Record Audio button brings up an interface similar to the main collection program. The difference is that only acoustic signals are read from this program. F_0 and SPL are still calculated during this collection; however, the SPL is not currently calibrated for this purpose so will only have relative values. You will need to run the Gain Estimator VI to get a new gain estimation and change that constant in the block diagram of the program.

Also, you will notice the large text tabs on the right. These are vocal tasks that are performed as part of CAPE-V data collection and analysis. You can change the text by clicking on different tabs on at the top.



ANALYZE DATA

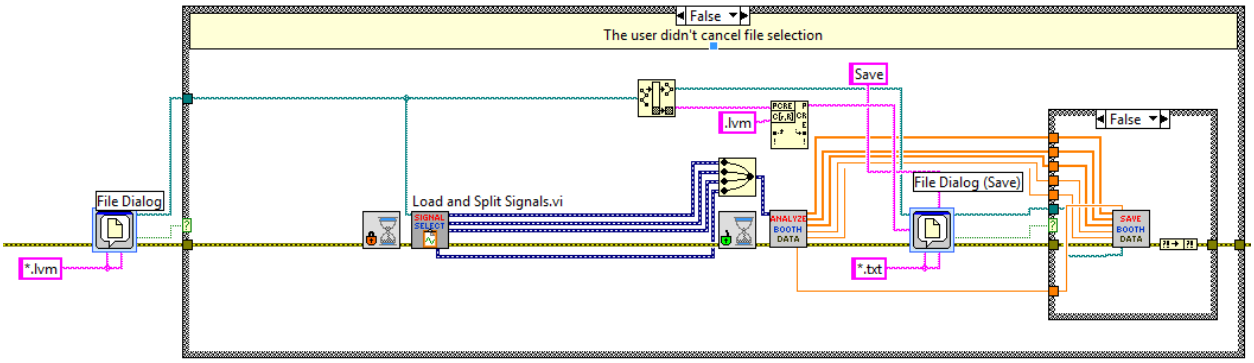
When you click the Analyze Data button, you will be prompted to select a file to analyze. The file will then be loaded and displayed on the two graphs. It may take a while to load the data depending on the size of the file.



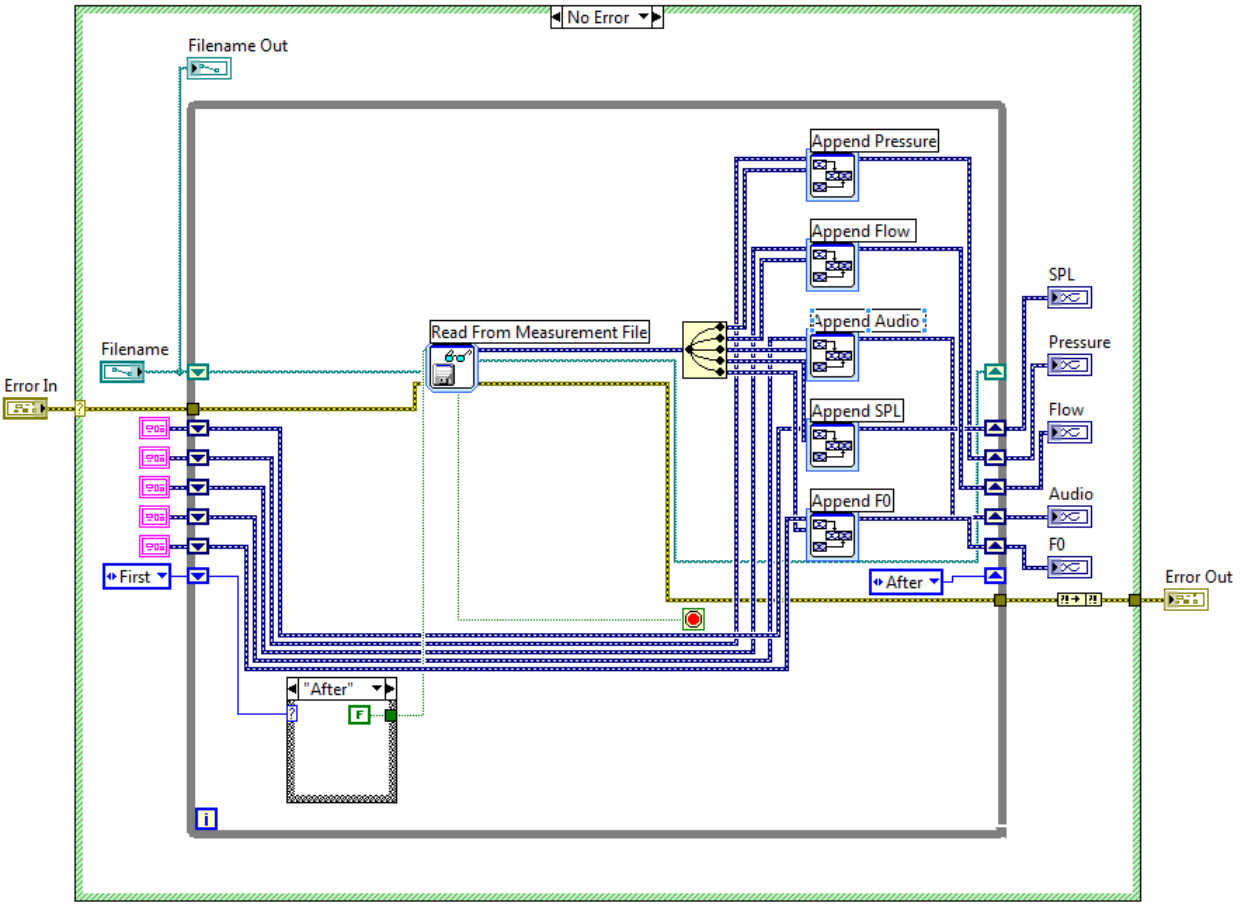
Once the data is loaded, the user can select sections of data to analyze. This is done through the use of cursors. On the upper graph, you can move the cursors left and right to control the bounds of the bottom graph. The cursors on the bottom graph can be moved to select the section of data that you would like averaged. When you are ready, you can click the 'Mark Selection' button and the averages will be stored. When you have marked all the sections, click 'OK' and you will be prompted for a save location.

THE BLOCK DIAGRAM

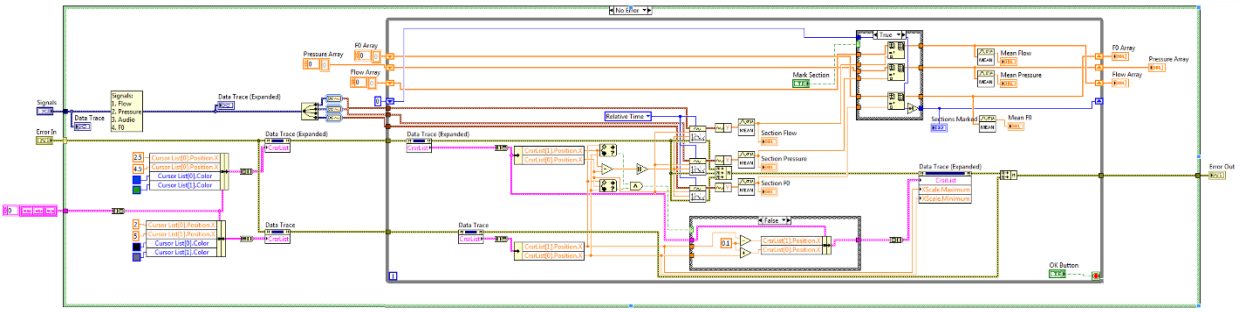
This process is comprised of three SubVIs placed within the 'Analyze Button' event case. The first loads and splits the signals read from a select LVM file. The second opens up a user interface where you select the sections you would like to average. The third saves the data the user selected into a TXT file.



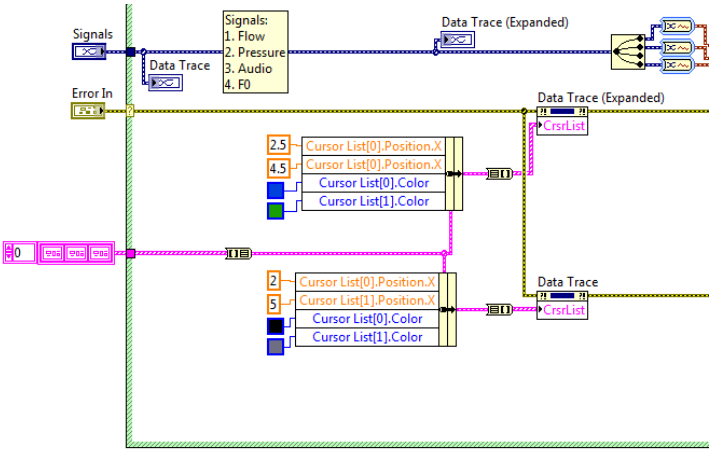
LOAD AND SPLIT SIGNALS SUBVI



Since the data is saved in segments, it must be read again is segments. What this VI does is read a LVM file one segment at a time until the end of the file is reached. Each segment is appended to the previous, then output from the loop once finished. In retrospect, I could have appended everything before splitting the signals, but I didn't… oh well.



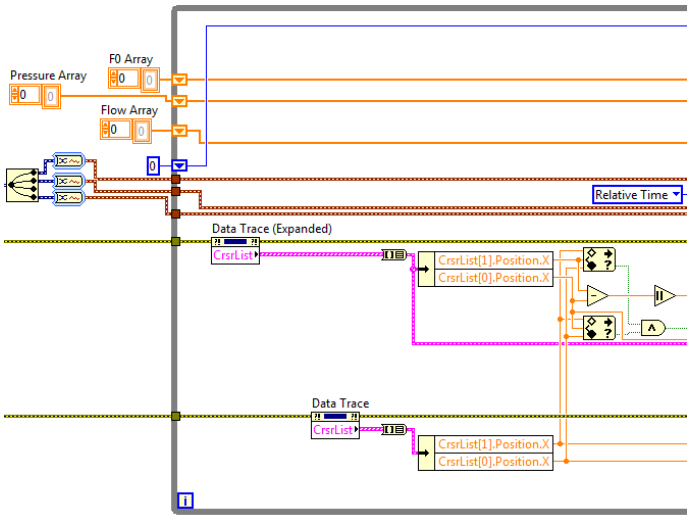
After all the signals are loaded, they are displayed through this SubVI. This is where the Front Panel controls appear (shown previously).



First, the signals and error wires are read into the SubVI. Then cursors for the upper and lower graphs are set then bundled and written into the Cursor List property nodes for the two graphs.

Since only flow, pressure, and fundamental frequency are needed for this analysis, the 1st, 2nd and 4th signals are split off the main.

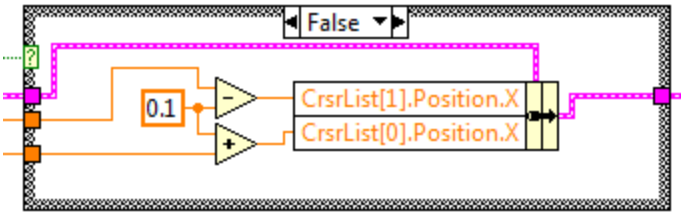
The dynamic data for the signals is converted to waveforms so it can be read later in the while loop. Arrays are also initialized here. These will store values to be saved later.



Inside the loop, the positions of the cursors are read from both graph's property nodes. Since the upper graph's cursors determine the lower graph's bounds, the lower cursors (Data Trace Expanded) must be between the upper cursors.

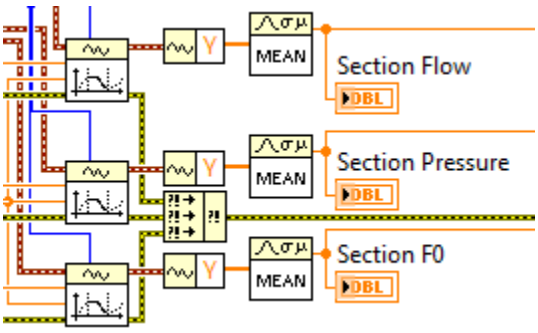
This is checked with the "In Range and Coerce function". The upper and lower limits are the right and left cursors of the upper graph (Data Trace).

Also, a range of data to average will be needed. This is the absolute value of the difference of the positions of the two cursors on the lower graph.

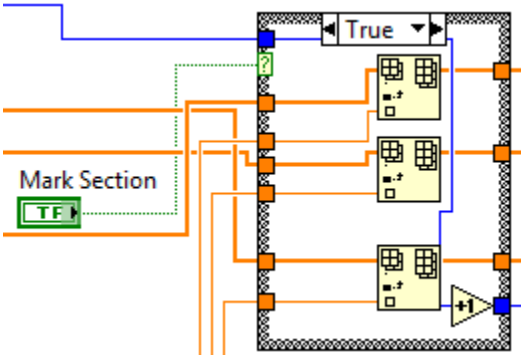


If the bottom graph's cursors end up outside the range, they will be moved to 0.1 seconds within the upper graph's cursor values.

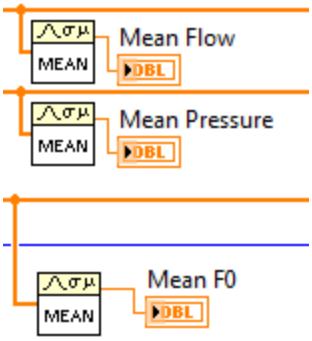
Using the Get Waveform Subset SubVI, and the range and left cursor value of the bottom graph, a section of data is selected. The mean of those sections are calculated and displayed.



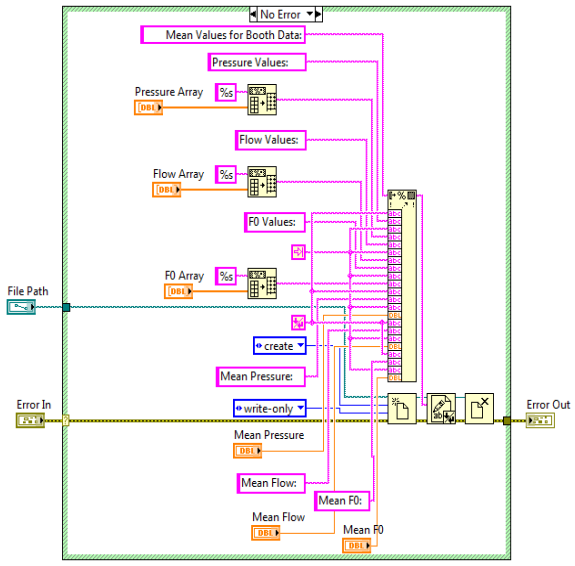
When the 'Mark Section' button is clicked, the selected data sections are added to their respective arrays. The Sections Marked value is also incremented here.



As section means are added the array, the overall mean will also update. These means, along with the arrays are written out back to the controlling VI to be saved in another SubVI



SAVE BOOTH DATA SUBVI



The main mess here is the many, many wires going into the Format Into String function.

Each array first needs to be converted into a spreadsheet string using the Array to Spreadsheet Sting function.

Once everything is either a string or a numeric (DBL) it can be formatted into a string. The extra inputs here are titles, tabs, and new line constants.

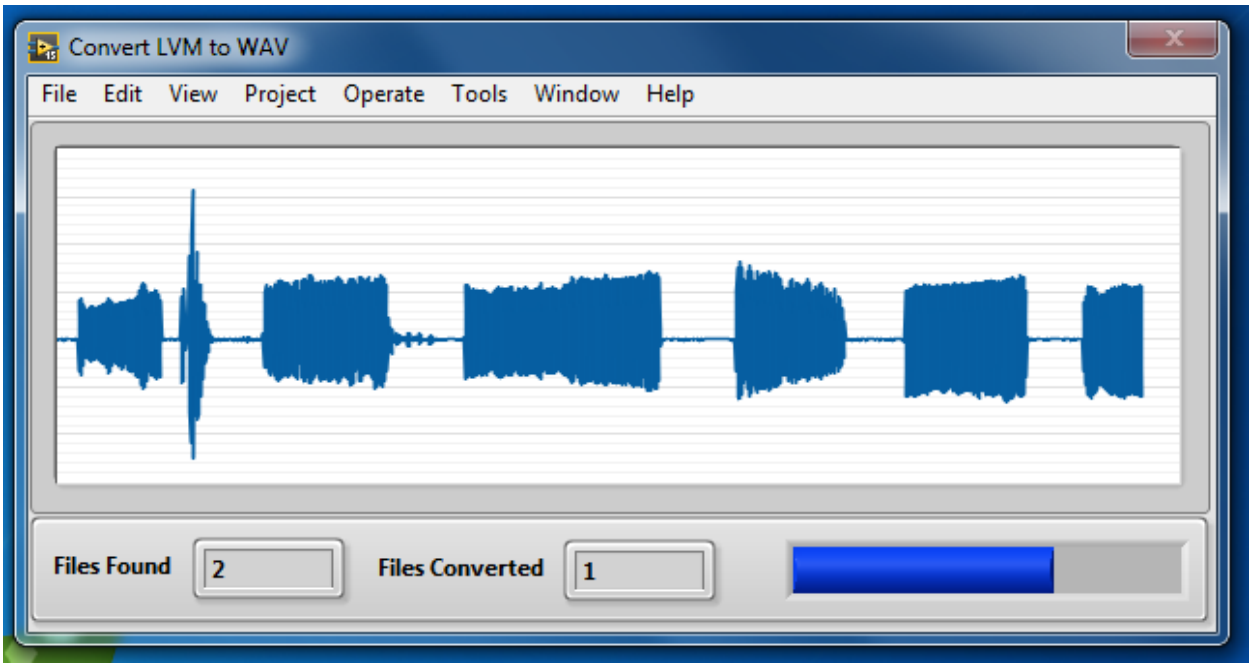
After everything is formatted into a string, it now can be written to a text file.

CONVERT LVM

In order to analyze the acoustic trace that you have previously collected, you will need to convert the collected acoustic data currently in an LVM (LabVIEW Measurement File) into a WAV file so that it can be run in our other analysis programs.

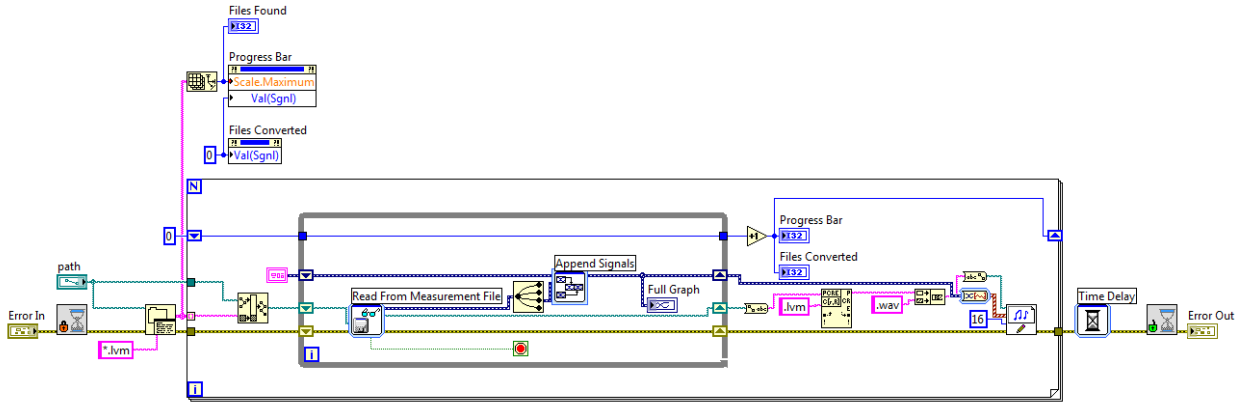
To do this, click the 'Convert LVM' button on the main controller. This will open up the file explorer and allow you to select the folder where your data is saved. Select the folder, and the program will run through all the LVM files in that folder and create WAV files that are the same name as the LVMs. These are then saved in the same folder.

When the files are being converted, they will be displayed on screen along with a progress bar for how many of the found files have been converted.



THE BLOCK DIAGRAM

This SubVI is not set up in a special structure. It is simply and while loop within a for loop.



To start, the SubVI reads in any errors and the selected path from the main controller VI. It then sets the cursor to *busy* while the program is running. Then, using the List Folder function, all the LVM file names within the selected folder are output into an array of strings. The size of this array (i.e. the number of files found in the folder) determines how many times the for loop will run. Additionally, through the property nodes above the for loop, the Progress Bar and Files Converted indicators are set to zero while the scale maximum for the Progress Bar is set to equal the number of files found.

The for loop will run through each of the file names. Using the Build Path VI, it will take the file name from the array and the folder path and output that into the while loop. Inside the while loop the Read From Measurement File VI will read the LVM and output all the data. It is then split (the third signal in saved data is the acoustics. This is just how it is saved in the collection program) and appended to any previous data from the loop, through the use of shift registers, and displayed on the graph.

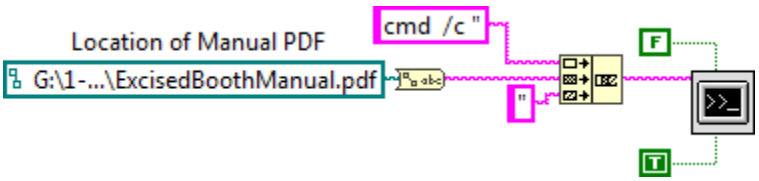
You might be asking "Hey, why is this done through a for loop, can't I just use the Read From Measurement VI to output all the data at once?" And my response will be, "Do you know exactly how many segments of data there are? Yeah, me neither." The data is written in chunks, so it must be read in chunks.

Anyway, after the data is all added together, it is taken out of the loop, converted to a 1D array of waveforms, and then put into the Sound File Write VI. The path used for this is the same as the one that is read by the Read Measurement File VI but the '.lvm' at the end is replaced with a '.wav'.

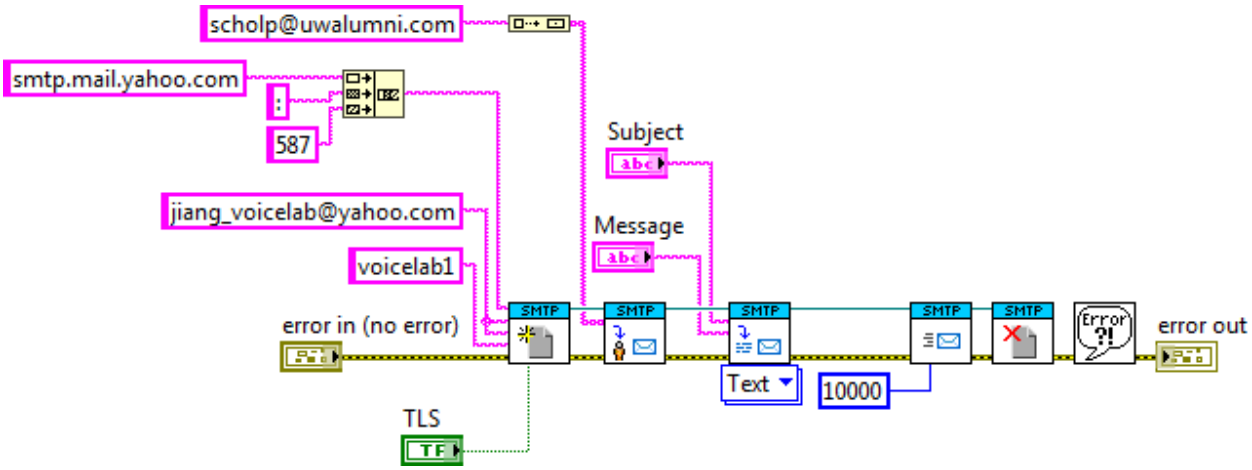
After all the file names in the array are run, the cursor is unset from busy.

OTHER EVENT CASES

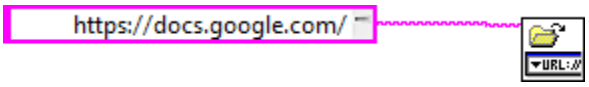
Help Button – Uses the System Exec.vi to open a PDF



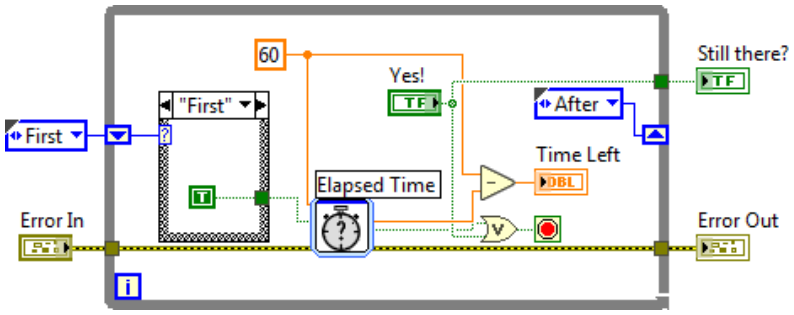
Email Austin – Uses SMTP protocols to send me a preset message. This will probably fail if the password is changed on the yahoo lab email account.

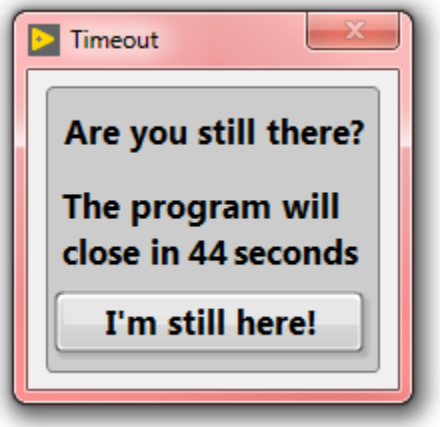


Error/Request – Uses a premade SubVI to open a URL in the default browser



Timeout – If 20 minutes go by without an event occurring, the user will be prompted. It is simply a while loop using the Elapsed Time Express VI.





If you cannot figure this one out, I do not think you should be messing with LabVIEW.