Aerodynamics Master Program Guide

For collecting and analyzing aerodynamic data.

By: Austin Scholp

Table of Contents

[Master Aero Program 1](#_Toc79406082)

[Front Panel Guide 1](#_Toc79406083)

[Main Menu 1](#_Toc79406084)

[Collect Data 1](#_Toc79406085)

[Calibrate Pneumotach 3](#_Toc79406086)

[Calibrate Helmet 3](#_Toc79406087)

[Randomize Tasks 4](#_Toc79406088)

[Calibrate SPL 4](#_Toc79406089)

[Extract Data from Text Files 4](#_Toc79406090)

[Extract Fundamental Frequency 5](#_Toc79406091)

[Analyze Data 5](#_Toc79406092)

[Data Clipper 6](#_Toc79406093)

[Fix LVMs 6](#_Toc79406094)

[Submit Error/Request 7](#_Toc79406095)

[Block Diagram Guide 7](#_Toc79406096)

[Main Menu 7](#_Toc79406097)

[Aerodyanmics Collection 9](#_Toc79406098)

[Randomize Tasks 13](#_Toc79406099)

[Calibrate Pneumotach 14](#_Toc79406100)

[Calibrate Helmet 15](#_Toc79406101)

[Gain Estimator (SPL Calibration) 16](#_Toc79406102)

[Extract Data 16](#_Toc79406103)

[Extract F0 18](#_Toc79406104)

[Analysis Master 19](#_Toc79406105)

[Clipper 20](#_Toc79406106)

[LVM Fixer 22](#_Toc79406107)

[Timeout Check 24](#_Toc79406108)

[Master Analysis Program 25](#_Toc79406109)

[Front Panel Guide 25](#_Toc79406110)

[Main Menu 25](#_Toc79406111)

[Labial Interruption 25](#_Toc79406112)

[Mechanical Interruption 26](#_Toc79406113)

[Incomplete Interruption 27](#_Toc79406114)

[Airflow Redirector 28](#_Toc79406115)

[Block Diagram 28](#_Toc79406116)

[Main Menu 28](#_Toc79406117)

[Analysis Sub-VIs 31](#_Toc79406118)

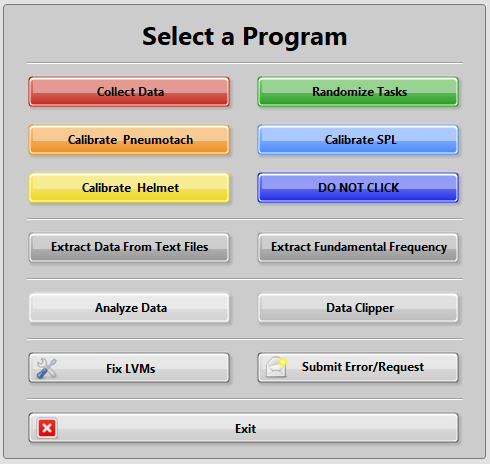
[End Note 35](#_Toc79406119)

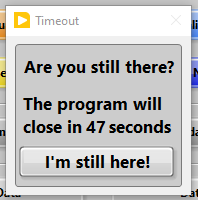
# Master Aero Program

The Master Aero program contains all the collection and calibration programs used with our aerodynamic devices. This includes the complete/mechanical interrupter, the incomplete interrupter, the airflow redirector, and the singing helmet.

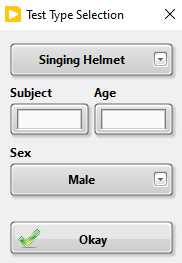
## Front Panel Guide

### Main Menu

The Master Aero program contains all the collection, calibration, and analysis programs for our aerodynamic devices. The Front Panel is simply an array of different buttons. You can hover over each button for a short description of its function.

If the user spends more than two minutes without clicking any buttons, the timeout prompt will trigger. If there is no input after another minute, the program will close.

### Collect Data

The first thing that occurs when this VI is opened, the user will be prompted to select a folder where their data will be saved. This should default to the Saved Data folder on the Aero Drive.

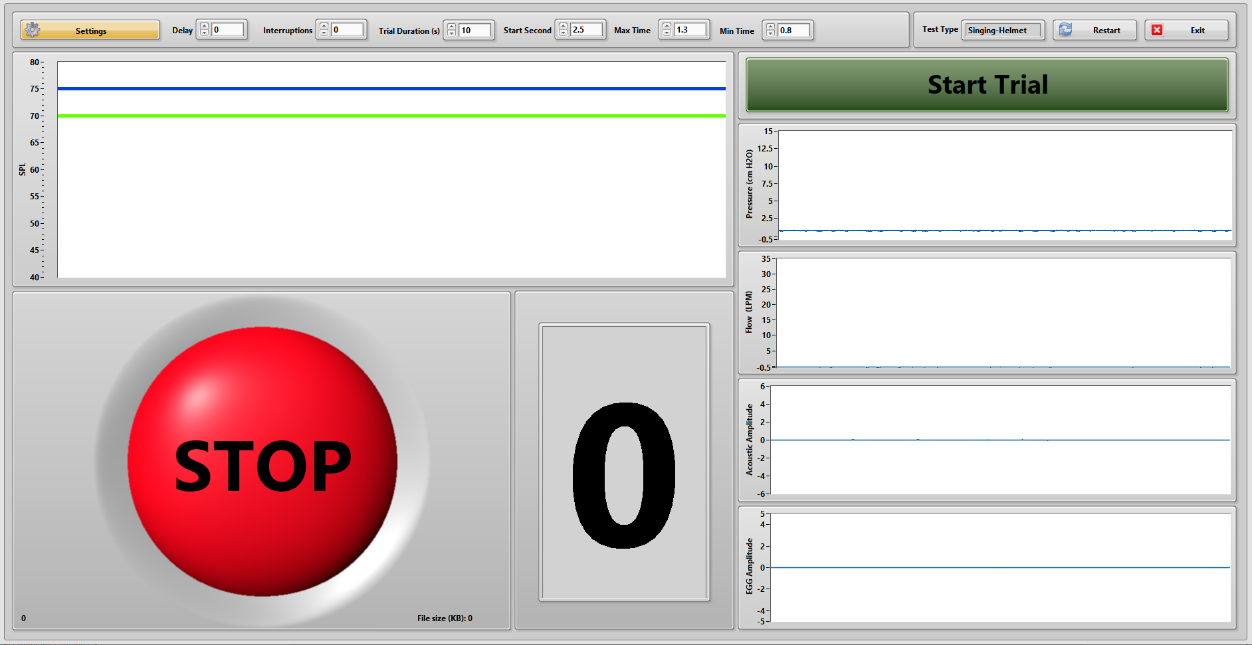
The user will then be prompted with the Test Type Selection VI. Here, they select what type of test you will be conducting. They should also enter the subject code, their age, select their sex, and then click **Okay**.

The settings and calibration data used during collection will depend on the type of test selected at this step.

The user will then be prompted to confirm the save location and file name.

Once the save file location is confirmed, signals will start being read from the NI DAQ board. Signals include:

* Sound Pressure Level (SPL) in dB – Upper Left
* Pressure in cm H2O – Upper Right
* Airflow in LPM – Below Pressure
* Acoustic Amplitude – Below Airflow
* EGG Amplitude – Bottom Right

All these data will be saved whether or not you have appropriate sensors hooked up.

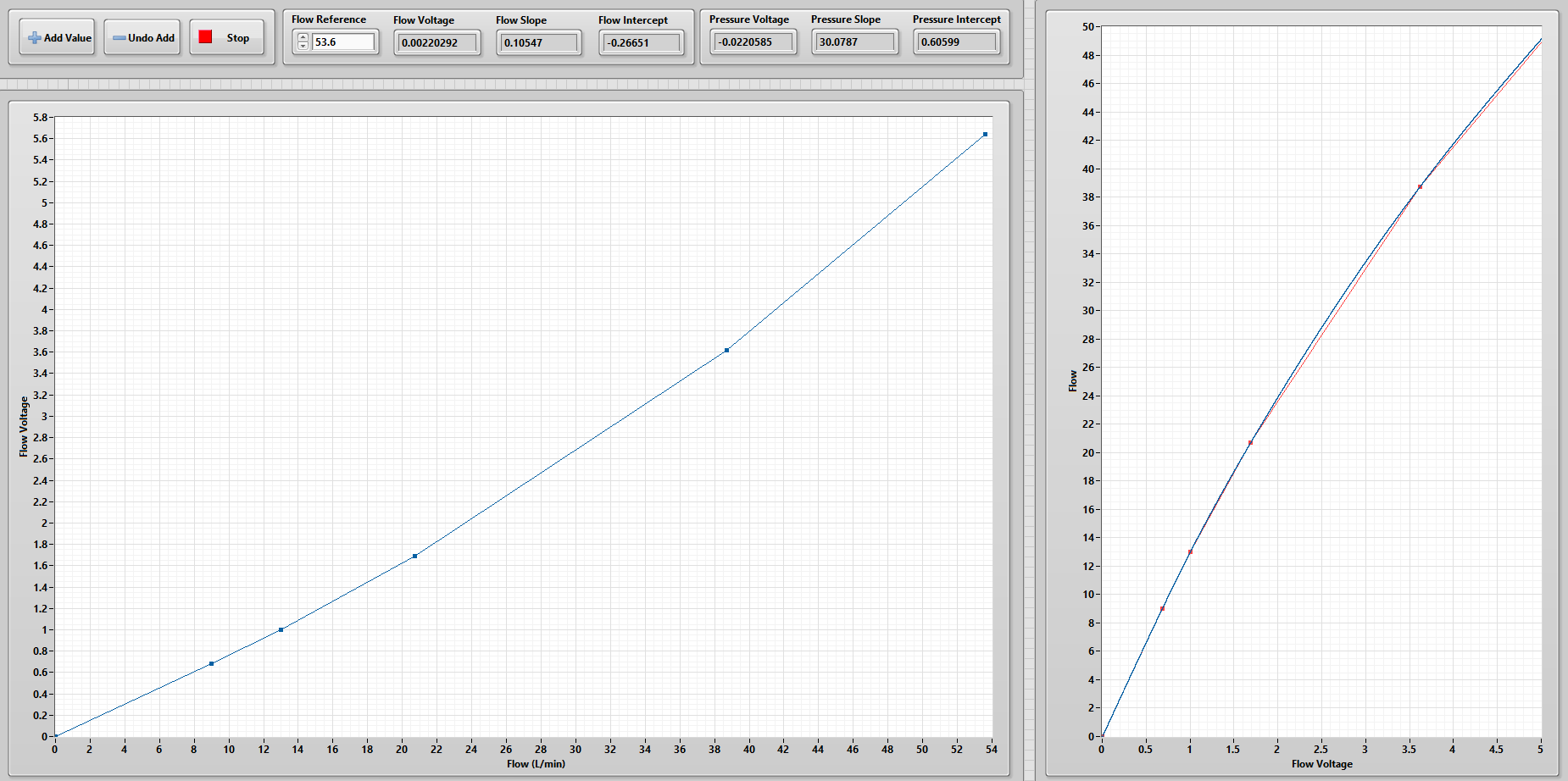
Clicking the **Settings** button will allow the user to alter the following:

* Delay – How long the balloon will be inflated
* Interruptions – How many interruptions there will be during a trial
* Trial Duration – How long until a trial will stop automatically
* Start Second – The earliest second of recording when the balloon can be triggered
* Max Time – The longest time between balloon triggers
* Min Time – The shortest time between balloon triggers

Other notes:

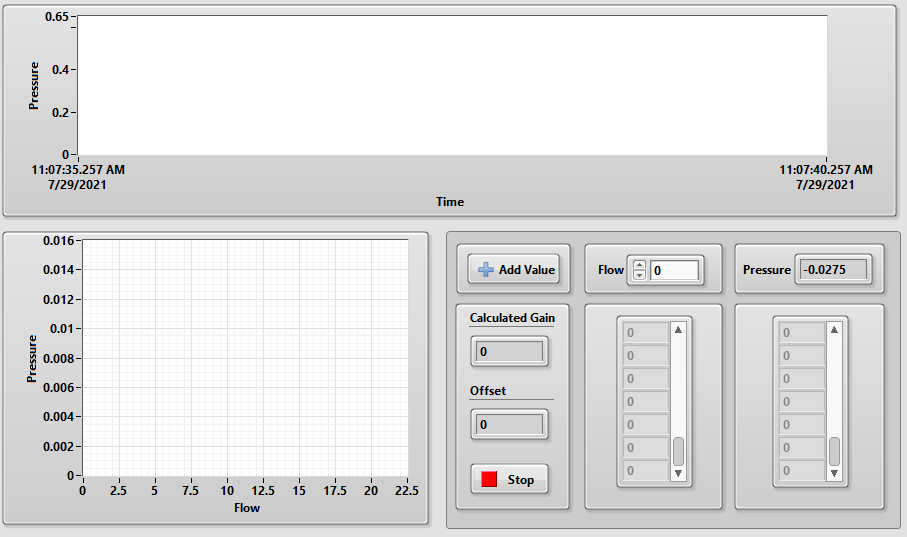
* The **Restart** button allows the user to change the test type without returning to the main menu. Default file names will adjust accordingly.
* The program is only recording when the stoplight shows a green **GO.**
* The small number in the bottom right indicates how many recordings the user has started.
* The size of the recording file is indicated to the bottom right of the stoplight. Use this to make sure that your data is being saved.
* When recording with the singing helmet, the flow data is saved as pressure.

### Calibrate Pneumotach

To calibrate the pneumotach, the user just needs to provide a reference flow value (i.e., tell the program the flow readout from the Omega flow meter) and add values. After adding at least 5 data points, you should end up with a plot similar to the one below. Do not worry, it is not supposed to be perfectly linear.

Since only the flow portion of the pneumotach needs to be calibrated, **the P+ port should be disconnected from the interruption device during this process**.

### Calibrate Helmet

This program is used to define the relationship between the voltage read from a pressure transducer and the flow through the singing helmet.

The user just needs to provide the flow input while the transducer is connected to the pressure port on the helmet.

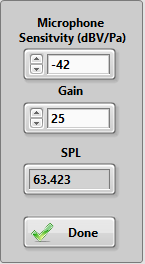
A *very* sensitive pressure transducer is needed to get a good calibration curve.

Note: when collecting data, the flow data is saved as pressure.

### Randomize Tasks

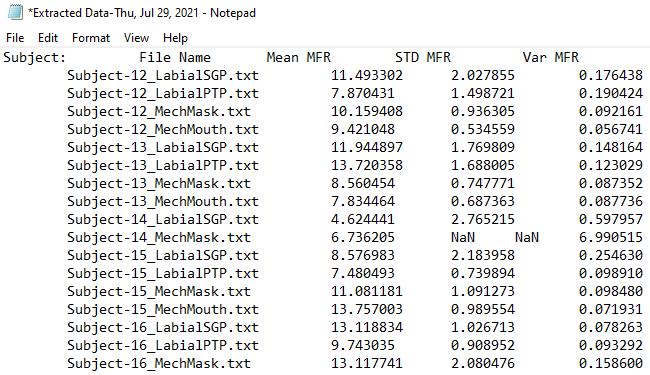
This simply displays a randomized order of tasks to be carried out while testing the complete interrupter. This is typically only used when you are comparing labial interruption to mechanical or complete interruptions.

### Calibrate SPL

To calibrate the SPL read from a microphone, you will need the microphone sensitivity in decibel-volts per Pascal (dBV/Pa). For the microphone currently in the complete interrupter, this value is -42. You will also need a calibrated SPL meter and a source of sound.

Set the sound source equidistant to the opening of the interrupter and the SPL meter. Then, adjust the gain on the program until the SPL readout matches the one seen on the SPL meter. This is not especially precise, but it gets the job done. Come up with a better way if you want.

### Extract Data from Text Files

**This program is used to get the means, standard deviations, and coefficients of variation from all the analysis text files in a folder. Note that the front panel for this program will not appear to the user.

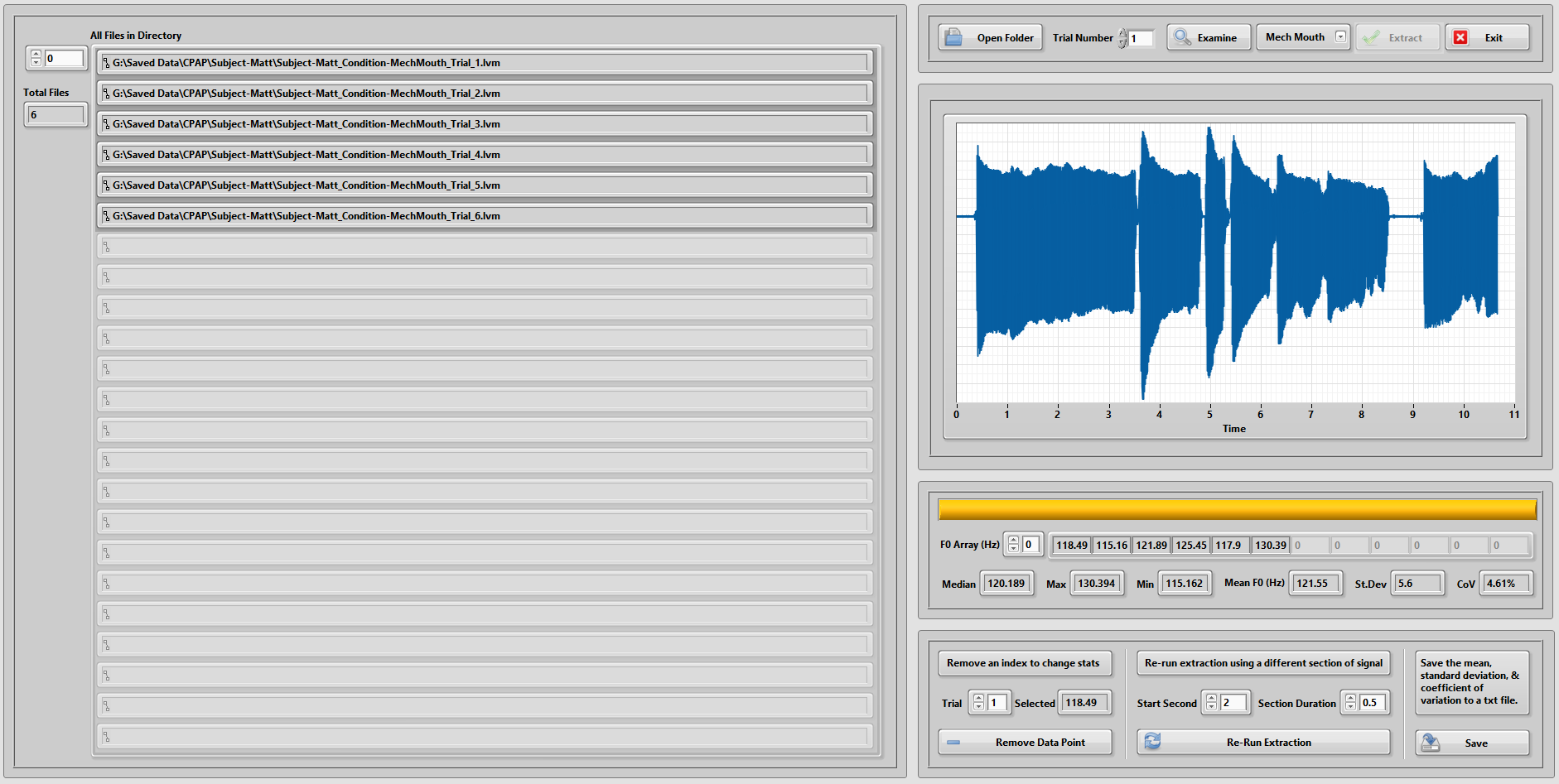
When opened from the main menu, the user will be prompted to select a folder of data. The program will then find all of the analysis results files in that folder and its subfolders and runs some basic statistics. The results of this are output in new text file with the default name of *Extracted Data-Day, Month, DD, YYYY.txt* For example: *Extracted Data-Thu, Jul 29, 2021*

Note: If and *NaN* appears in the extracted data, that means the program could not find any trials with that data. If the *NaN* is just in the standard deviation and variance columns, that means only one trial was found but stats could not be computed.

### Extract Fundamental Frequency

You should not need to use this program unless you are reanalyzing data collected before 2018. It is used to calculate the fundamental frequency on data files where F0 was not recorded, but acoustic data was recorded. Steps to complete this are below.

1. Click **Open Folder** and select a folder of LVM files.
2. Select the type of trial (e.g, mechanical mouthpiece) you need to extract data from using the dropdown menu.
3. View each trial by setting the **Trial Number** and clicking **Examine.**
4. In the bottom right, set the **Start Second** and **Section Duration** to be where there is phonation in all of the trials (2 second start and 0.5 second duration usually work).
5. Click **Extract** and wait for F0 data to be calculated for each trial.
6. If one of the trials shows an impossibly low or high F0, you will want to remove it. Do this by selecting that trial and clicking **Remove Data Point** in the section labeled “Remove an index to change stats”.
7. There is also an option to re-run the extraction.
8. Once you are satisfied with the results displayed in the **F0 Array**, click **Save.**

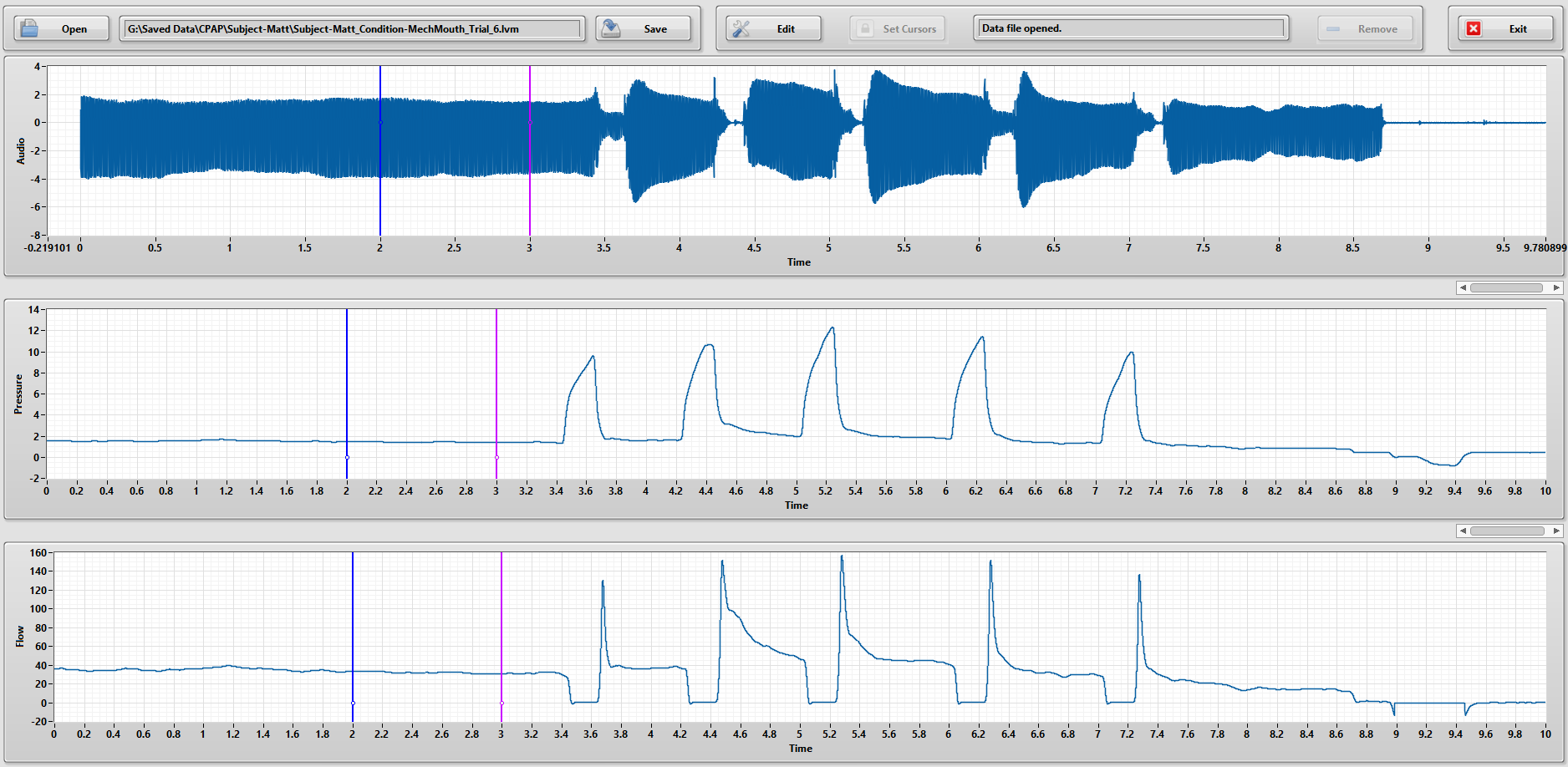


### Analyze Data

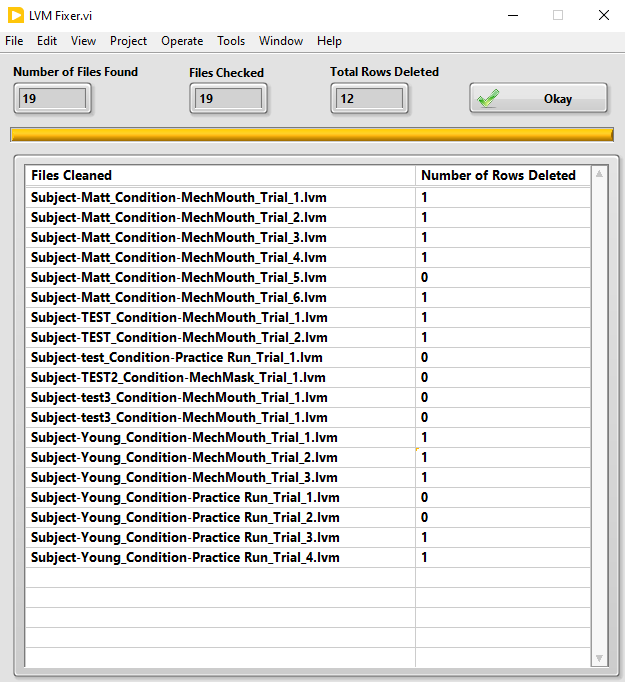
More information on the data analysis portion of this program can be found in a later section of this user guide.

### Data Clipper

This program is used to remove sections of data from an LVM file. This is primarily used for the singing helmet project.

1. Click **Open** and select an LVM file you would like to edit.
2. Click **Edit** to unlock the cursors.
3. Move the cursors to a section of data you want to remove.
4. Click **Set Cursors** to lock them in place.
5. Click **Remove** to delete the select section of data.
6. Click **Save** to save the edited data file.

### Fix LVMs

*NaNs*can appear for a myriad of reasons, but they usually are not a problem. However, they can cause problems if there are a lot of them. If an LVM data file has a lot rows that show *NaN*, use this program to delete those rows.

The user will be prompted to select a folder of data files. The program will then search through all of the LVM files in that folder and delete the rows of data that contain one or more *NaNs.*

Once completed, it will display the files that were found in the selected folder and how many rows of data were deleted from each.

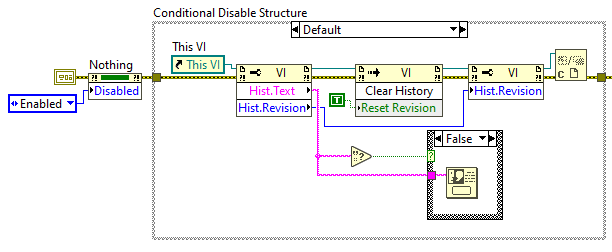
### Submit Error/Request

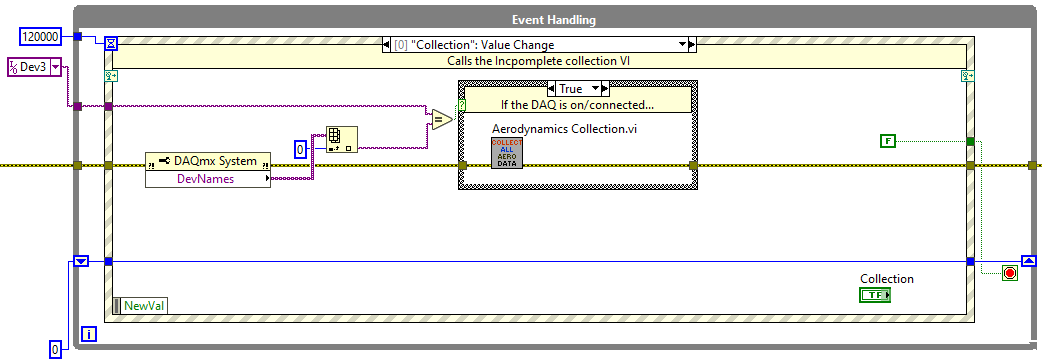
Clicking this button will open up a Google Form in your default internet browser. From there, you can report errors that you run into, request edits be made on a program, or request a completely new program. All submission on this form will notif y Austin and he will respond whenever he can.

## Block Diagram Guide

In the following sections, we will go over the block diagrams for all of the important VIs and SubVIs found in the Master Aero Program. Details for the Master Analysis Program will be covered later in this user guide.

### Main Menu

The first thing that runs is Conditional Disable Structure. This only runs when the program is running from the LabVIEW project folder. The code in this structure displays the most recent revision notes and updates the revision number.

The primary part of the main menu code is an event structure. If you do not know what that is, you should review the first two sections of the LabVIEW binder. On the next pages, you will find descriptions of all the cases in this event structure.

#### Collection

* Checks to see if the DAQ board is connected by checking the device names found by the computer and comparing the primary name to what the default is set to (Currently Dev3).
  + **If you try running this on a different computer or with a different DAQ board, you will need to change the default device.**
* If the DAQ board is found, the Aerodynamics Collection VI will be called. If not, a warning message will be displayed to the user.

#### Randomize

* This only opens the Random Task Order VI.

#### Calibration

* First, this case checks if the user *really* wants to calibrate. This is to avoid accidentally running the calibration VI and setting everything to zero.
* Similar to the Collection case, this checks to see if the DAQ board is connected.
* If the DAQ board is connected, the Calibrate Pneumotach VI is called.

#### Calibrate Helmet

* There is the same check about *really* wanting to calibrate.
* Again, there is a check for the DAQ board connection.
* If the DAQ board is connected, the Calibrate Helmet VI is called.

#### SPL

* Opens the Gain Estimator VI if the user *really* wants to run calibration.

#### Extract Button

* The user is prompted to select a folder of data analysis files.
* Assuming file selection is not cancelled, the Extract Data VI is called.

#### Extract F0

* This just calls the Extract F0 VI.

#### Analysis

* This just calls the Analysis Master VI.

#### Clipper

* This just calls the Clipper VI.

#### Fix LVM Button

* The user is prompted to choose a folder of LVM files.
* Assuming file selection is not cancelled, the LVM Fixer VI is called.

#### Report Error or Request Change

* The Google form for reporting errors or requesting changes is opened in the default browser. This uses the Open URL in Default Browser VI. Very convenient.

#### Timeout

* This case is triggered when no user input is detected for the length of time wired to the event timeout node.
* This is currently set to two minutes (120,000 ms).

#### Application Instance Close

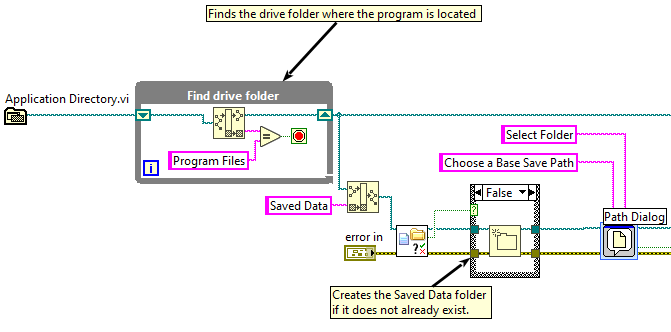
* This stops the loop when the user hits the X button and the VI is running as an executable or installed program.

#### Panel Close?

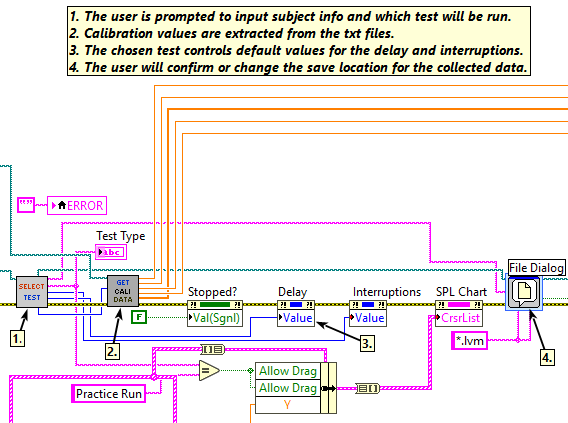
* This stops the loop if the user hits the X button and the VI is running from the LabVIEW project folder.

The final part of the main menu code is another Conditional Disable Structure. Here, LabVIEW is closed, but only when the program is running as an executable or installed program (i.e., when RUN\_TIME\_ENGINE==True).

### Aerodyanmics Collection

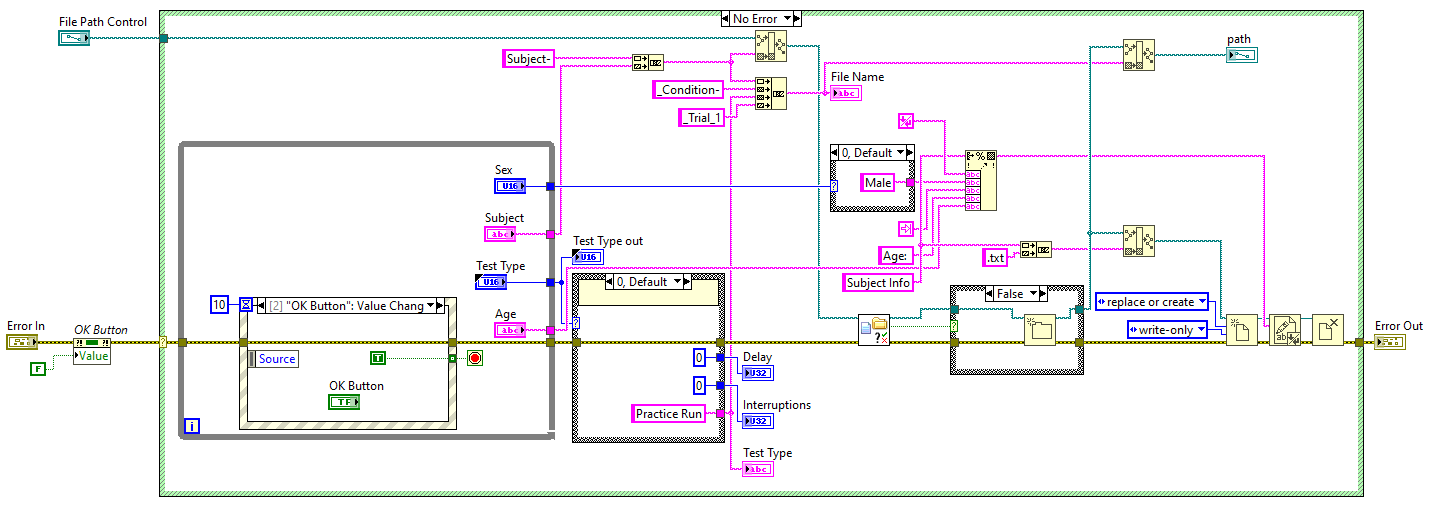
First, the program finds the drive folder where the program is located. **It must be *somewhere* in a folder named “Program Files”. If it is not, this will fail.** You could also change the folder name within the Find drive folder loop.

Next, a folder named “Saved Data” is created if it does not already exist and the user is prompted to confirm the save path using the File Dialog express VI. If “Cancel” is clicked in the file dialog, nothing else will happen and the user will be sent back to the main menu. Once the save path is confirmed, the program enters the Master Control Loop.

Once in the loop, the Test Type Selection SubVI is called. Here, the user selections determine what the default delay and number of interruptions. It also controls which pressure calibration data is output by the next SubVI, Get Calibration Data. More details on what happens in each of these SubVIs can be found in the following sections.

Again, the user is prompted to confirm the save file name and location. If this is not cancelled, the program moves on to the parallel loops portion (described after the details of the Test Type Selection and Get Calibration Data SubVIs).

#### Test Type Selection

First the error and file path are wired in from the calling VI and the OK button is set to False. Assuming there are no errors, the program starts the while loop. Using the while loop with an event structure, this SubVI waits for the user to click the OK button. Once that happens, whatever the user has set in the front panel will be used to determine the default file name and default number of interruption and interruption length (delay).

Default file names are built as outlined below. Items in parenthesis indicate user input.

*Subject-(Subject)\_Condition-(Test Type from case structure)\_Trial\_1.txt*

A folder for the subject is created if one does not already exist. Subject Info text files are also saved here. They include Age and Sex of the subject.

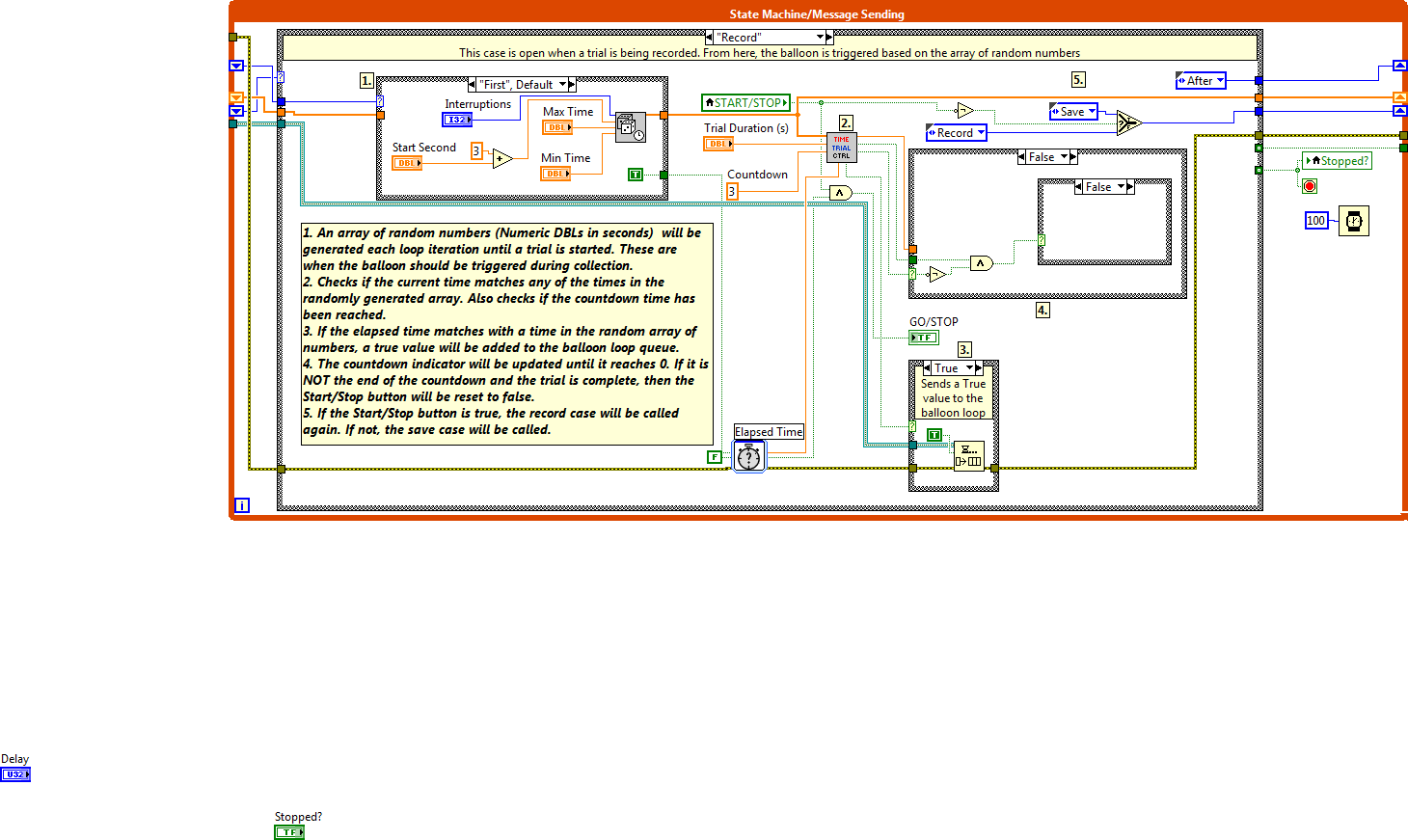
#### Get Calibration Data

This SubVI opens all the calibration text files using the sequence of functions shown to the right. This opens a file reference for a given file path, reads the text file as a single string, then closes the reference.

#### Parallel Loops Portion

This program is organized into three parallel loops within the Master Control Loop. The design pattern used here is called a queued message handler. One loop contains a state machine that sends instructions to the other two loops based on user input.

##### State Machine/Message Senging Loop

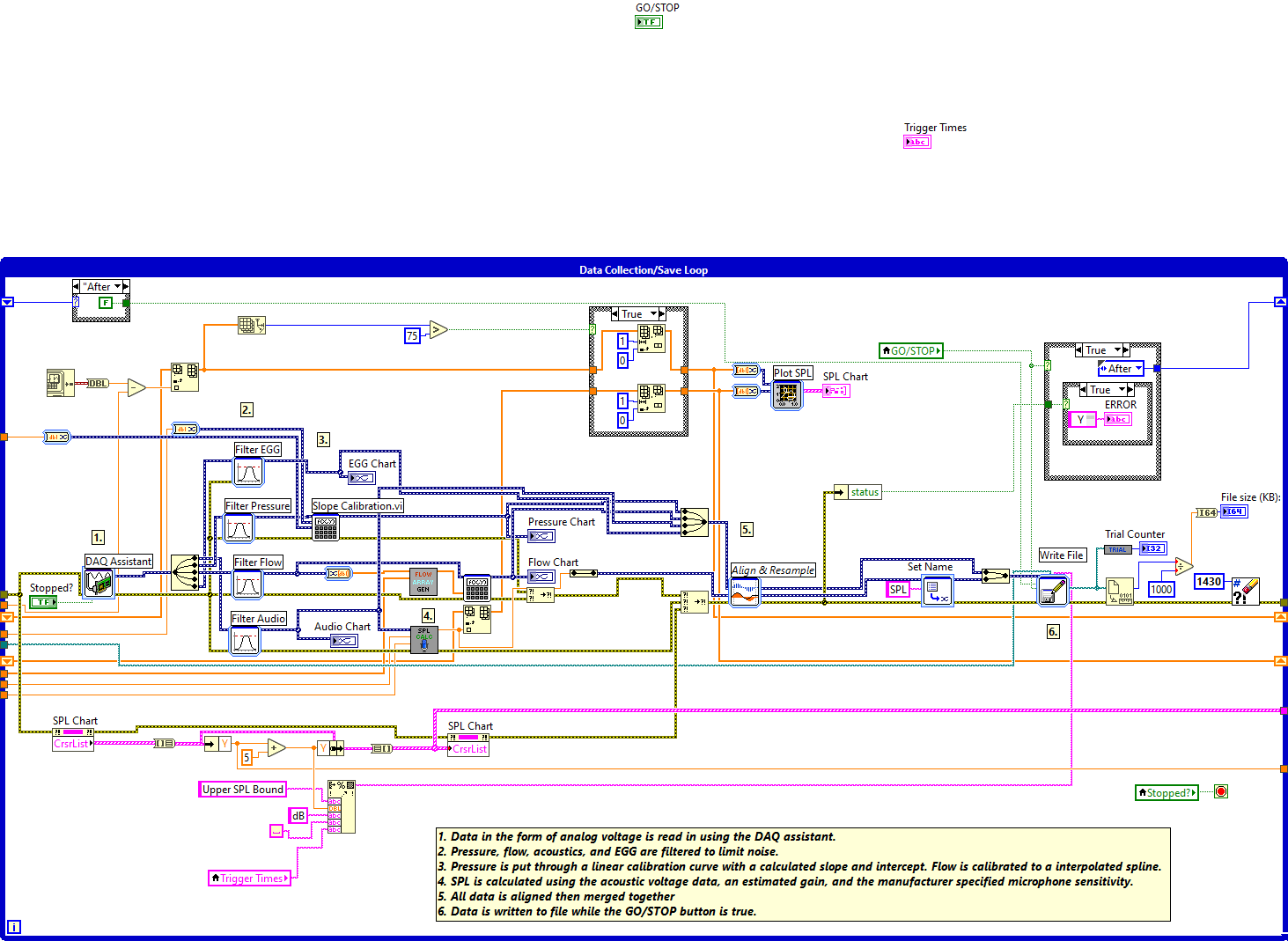
This loop is an event-controlled state machine. This controls the GO/STOP light, as well as when the balloon interruptions are triggered. A short description of each case is below.

* **Initialize** Sets the GO/STOP button to false (Red/Stop) and then calls the **wait** state.
* **Wait** Contains the event structure. The next state called depends on the event.
* **Record** Has the controls for the random array of numbers that will be used as interruptions. It also controls when to stop the trial and sends triggers to the Balloon Control Loop. Finally, it controls the stoplight and the START/STOP button.
* **Save** A previous version of the program utilized this state, but it is no longer called.
* **Exit** Stops the State Machine loop, which in turn stops the other two loops. This will also stop the master control loop.
* **Reset** Stops the State Machine loop, stopping the other two internal loops, but does not stop the master control loop, which then goes through another iteration of the loop, going back to the Test Type Selection SubVI.

I mentioned an event structure. Here are the events:

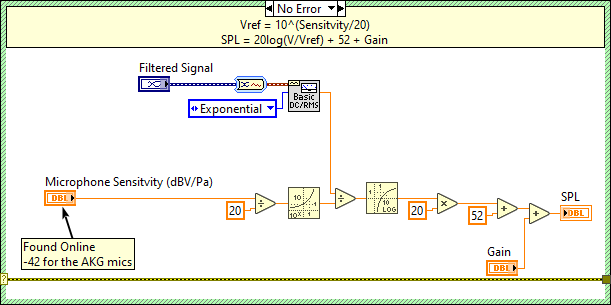
* **START/STOP** Whenever this button is pressed, the record case is called. Whether or not data is recorded is controlled through the **Record** state.
* **Exit Button** calls the **Exit** case when the exit button is pressed.
* **Restart Button** calls the **Restart** case
* **Settings** Depending on if the settings button is true or false, the trial settings will become visible.

##### Data Collection/Save Loop

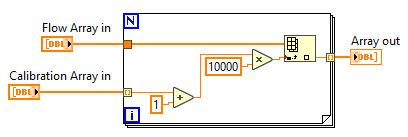
Here is the data collection/save loop. It reads from four channels on the DAQ board using the DAQ Assistant. It takes in acoustic, pressure, EGG, and flow data from their respective devices. Data is then filtered, and run through calibration curves to obtain actual values.

Other things to note are the pair of “First-After” case structures. These set the reset value on the Write Measurement to true only on the first iteration of the loop so a new data file is created whenever a new trial is started. Also, the x-axis of the SPL graph is controlled by building an array of values that is limited to 80 points.

###### Calibrate SPL SubVI

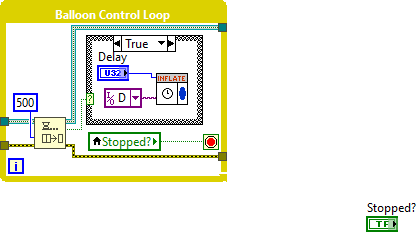
This is where SPL is calculated from the filtered acoustic signal. The equations used in this calculation are below. The gain is taken from the *SPL Gain.txt* file.

###### Flow Array Compare SubVI

When the pneumotach is calibrated, an array of flow values is created instead of using a slope-intercept formula. This is because the voltage-flow relationship for the sensor is not completely linear.

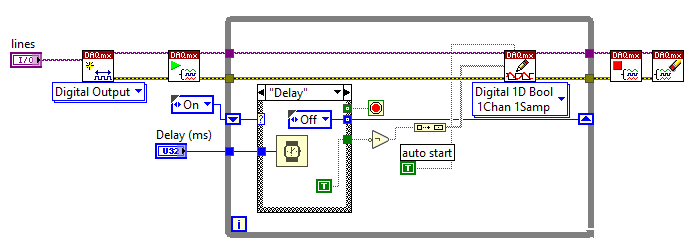
This SubVI takes in the flow voltage collected through the pneumotach applies it to the calibrated array to get calibrated flow data.

##### Balloon Control Loop

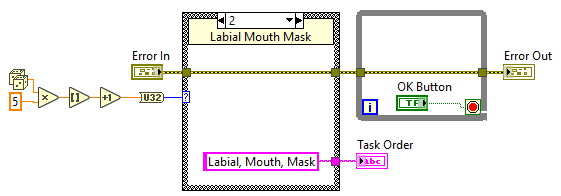
Within the **Record** state of the trial control loop, if the current trial time matches one of the times in the randomly generated array of numbers, a true value will be added to the Boolean queue. When this happens, the case structure within the balloon control loop changes to true and the balloon is inflated for however many milliseconds the delay control is set to.

###### Inflate SubVI

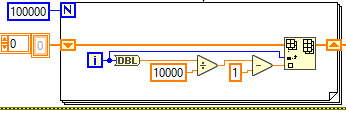
The balloon interruption length is controlled by delaying the loop iteration by the input number of milliseconds. This previously used the DAQ assistant. It now uses the formal DAQmx control structure, which is what you are supposed to use if you are not lazy.

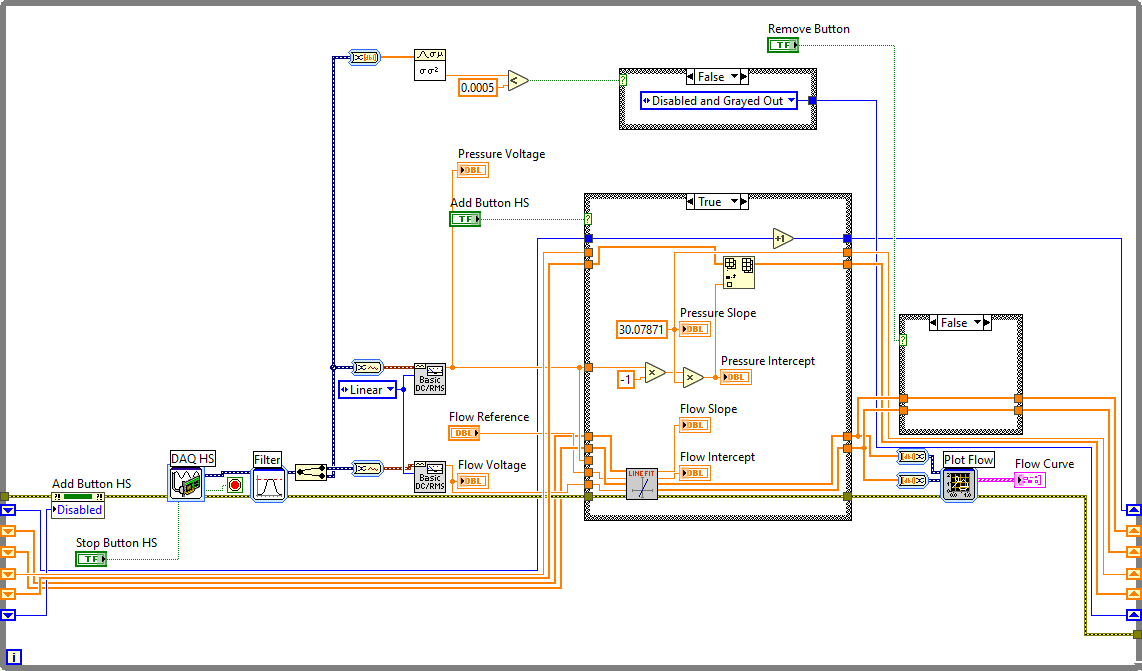
First the “On” case is called which sets the DAQ digital output to True, which tells the balloon control box to inflate the balloon. In the “Delay” case, the loop waits for a given number of milliseconds before stopping the loop and deflating the balloon.

### Randomize Tasks

Each of the six possible options is preloaded in the case structure. A random number from 1 to 6 is generated to pick a case and display it to the user. The while loop waits until the user clicks OK then exits the SubVI.

### Calibrate Pneumotach

This for loop creates a 100,000 index array and fills it with values from 0 to 10,000. This will be used later to create an array of flow vs. voltage data.

The main loop of this SubVI can be found below. Here, voltage data is read in using the DAQ assistant, then filtered and the voltages are displayed to the user. There is also a small piece of code (outlined in red) that controls whether the add button is enabled. It will only be enabled when the signal has leveled out (i.e., when variance < 0.0005).

When the user clicks **Add Value**, the main case structure (blue box) is set to true. Here, the pressure intercept is updated (zeroed) and the flow slope and intercept are calculated. See the [Fit Curves VI](#_Fit_Curves_SubVI) subsection for more detail on this calculation.

The next case structure is set to true whenever the **Undo Add** button is pressed. This removes the most recently added data point from the pressure and flow arrays. Once the user is done calibrating, the flow array is run through a spline curve interpolation VI and the pressure intercepts are averaged. Next, the data needs to be saved into the appropriate text files. To do this, the base folder needs to be located (the same thing occurs in the first step of the Aerodynamics Collection VI). From here, a path is built to the *Calibration .txt files* folder, then the *Calibration Values.txt* file.

1. A picture containing diagram

   Description automatically generatedThe File/Directory Info VI finds the old calibration file and outputs the timestamp of the last time it was modified (this should be the same as when it was created).
2. The Get Time/Date String VI converts the timestamp to an abbreviated date string.
3. With the Concatenate String function, a new file name is created for the old calibration file called *Calibration Values\_Day, Month DD, YYYY.txt.*
4. A new path is built for the old calibration file. It is now in the *Old Calibrations* folder, within the *Calibration ,txt files* folder.
5. The Move VI moves the old calibration file to its new path.
6. The Open/Create/Replace File VI creates a new reference for the new *Calibration* *Values.txt* file.
7. The Write to Text File VI Takes in the string of calibration data and writes it to the new text file.
8. Finally, the Close File VI closes the reference for the new calibration file.

This process is done for both the flow array values (saved in *Flow Array.*txt) and the pressure values (saved in *Calibration Values.txt*).

#### Fit Curves SubVI

Diagram

Description automatically generated with medium confidenceThis SubVI takes in a reference value (what the user wants to assign to the signals read from the DAQ board) and a voltage value. These are added to their respective arrays, which are then wired to the Linear Fit VI. Using the input arrays, a slope and intercept are calculated. These values are output from this SubVI and later saved.

### Calibrate Helmet

This VI is similar to the one used to [calibrate the pneumotach](#_Calibrate_Pneumotach). The main difference is that this VI assigns voltages read from a pressure transducer to known flow values. The same [Fit Curves VI](#_Fit_Curves_SubVI) is used to calculate the slope and intercept (gain and offset) to be saved and used during collection. New calibration values are saved to *SH Cali.txt* and old values are not saved as they are for the pneumotach.

### Gain Estimator (SPL Calibration)

Diagram

Description automatically generated with medium confidenceUsing the [same calculation](#_Calibrate_SPL_SubVI) seen in the Aerodynamic Collection VI, the user alters the gain until the SPL readout matches a known SPL value.

See the front panel description for more details on how to use this VI.

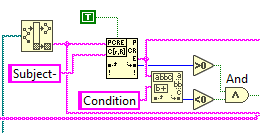
Note: the filter used on the audio signal is a high-pass filter with a cutoff frequency of 70 Hz.

### Extract Data

The block diagram for this VI is poorly commented and I apologize for that. Hopefully, no one will ever need to edit this program.

Anyway, using the folder path input from the calling VI, the Recursive File List VI (left) locates all the text files in the given folder. The first for loop will iterate through all the files found here.

#### Outer For Loop

For each file path in the input array, the file name is stripped from the path (only the file name needs to be checked, not the entire path) and scanned for the string “Subject-“ and the string “Condition”. If “Subject-“ is found, but “Condition” is not, a true value is wired to the case structure and the next section of code is run.

Note: The reason these specific strings are used is because they will only appear in the default names of analysis text files. If a user saves their analysis files as something other than the default, this program will not be able to find them.

A picture containing shape

Description automatically generatedA string of data is pulled from the matching text file and converted into a 2D array. The first column of this array is wired into the inner for loop.

#### Inner For Loop

Diagram

Description automatically generatedThe first column of each analysis text file should contain labels for what each row of data contains. As the for loop iterates through the rows, it scans them to see if any of the string match the types of data that are to be extracted (SPL, MFR, F0 Values, SGP, PTP, Pressure). All of the data types will be run through the Find Match Values VI described in a later section.

In short, the Find Match Values will add to an array containing data of a specific type (Pressure, SPL, etc.). Each time a match is found in a row, the matching data will be added to the array. Note that Pressure and SGP are the same. Older files may have different organizations than newer analysis files.

Diagram

Description automatically generatedMFR is slightly different because previous version of analysis code saved MFR values differently, so the program checks for both. It will either add the value found after the colon or (if the scan from string throws an error), it will be added using the Find Match Values VI.

Diagram, schematic

Description automatically generatedOnce all the data has been found in the given text file, resistance is calculated (Pressure or SGP divided by MFR). Means and standard deviations are then calculated for each data type. The coefficients of variation are calculated from theses. All the computed data (means, StDevs, CoV) are added to an array. This array is then converted to a spreadsheet string, which is then added to an array of strings, which exits the case structure.

The outer for loop will run through all of the analysis files. When complete, all the data is then formatted into a new file named *Extracted Data* plus a timestamp.

##### Find Match Values SubVI

This SubVI is called every iteration of the inner for loop described in a previous section. The code inside the case structure will only execute if a match was found.

Graphical user interface, diagram

Description automatically generatedThe first part of the code within the case structure finds the matching row in the full text string then removed the first index (the label where the match was found).

The for loop then iterates through all the columns (indices) in the matching row. If an index is not blank, the code in the next case structure will be run.

Within the case structure, the Scan From String function will attempt to convert the string into a double. If it is successful, it will add that number to an array. Once all the indices of the row array have been checked, they will be averaged and, if the mean is confirmed to be a number, the mean is added to the Array of Values and output from the SubVI.

### Extract F0

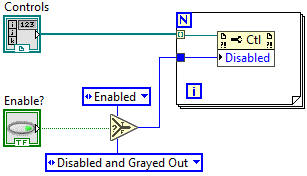
This VI is set up in a state machine design pattern. Here are short descriptions of each case found in the state machine. More details can be found in the comments for this block diagram.

* **Initialize** – The front panel objects are set to their default values here. All data that needs to be accessed or updated in other cases is added to a cluster. The update case is always called after initialization is complete.
* **Update** – In this case, data is unbundled from the cluster and used to update the front panel controls. The wait case is always called once update is complete.
* **Wait** – This case contains and event structure. The events handled here are described a little later in this section.
* **Open Folder** – Through the file dialog VI, the user selects a folder of LVM files. An array of file paths for these LVM files is created using the Recursive File List VI. The **Test Type** and **Examine** buttons are enabled while the **Extract** button is disabled. The update case is called when this is complete.
* **Examine** – The first for loop searches through the array of file names made in the open case looking for those that match the selected test type (labial, mechanical, etc.). If matching files are found, the audio data from the selected trial will be displayed to the user. This case makes use of the Split Signals VI, which is described in more detail in the Master Analysis section of this guide. The update case is called once this is complete.
* **Extract** – This case takes all the files with the matching test type and calculates the fundamental frequency of a section of the data. This section is set by the Start Second and Section Duration controls. Statistics for the frequency data are calculated and displayed to the user. The update case is called once this is complete.
* **Rerun** – This case is similar to the examine case. However, instead of opening a trial and displaying the audio data, it simply reloads the subset of file names that match the selected test type and calls the extract case.
* **Remove** – Here, a selected index is removed from the data array and the statistics are ran again. The update case is called once this is complete.
* **Save** – The fundamental frequency data is saved into text files.
* **Exit** – The program exits.

Here are the events handled by the event structure.

* When the **Open Folder** button is pressed, the open folder case is called.
* When the **Examine** button is pressed, the examine case is called.
* When the user changes the trial number control in the “Remove an index” section, this will load the array index to the cluster so it can be removed when the user clicks the **Remove Data Point** button. The update case is called after this.
* Whenever the user changes the test type in the drop-down menu, this case loads the corresponding string to the cluster and disables the extract button. The update case is then called.
* When the **Extract** button is pressed, the extract case is called.
  + This is the same for the **Rerun**, **Remove**, and **Exit** cases.
* When the user exits the program, the application instance close or panel close cases are called, depending on whether the program is running through the run-time environment. These stop the loop.

#### Set Enable State on Mutiple Controls SubVI

This SubVI is used to enable or disable more than one front panel control at a time. To do this, an array of control references is wired into the for loop. Based on the Boolean value wired to the select function, each control in the array will be enabled or disabled through the use of an unlinked property node.

### Analysis Master

Details about the Block Diagram of the Analysis Master VI will be found in a later section.

### Clipper

I originally made this VI using the Queued Message Handler template provided by LabVIEW. That is why there is some extra stuff in the block diagram that you will not find in my other programs that use this design pattern. There are also some pre-made SubVIs here that I do not fully understand.

* First up is the Create User Event – Stop VI. This helps to make sure that all the loops stop when the user exits the program.
* Next is the Create All Message Queues VI. This sets up the message queue for the user interface (UI) objects.
* The last pre-made icon is actually for a control, not a SubVI. This sets up the data cluster for information that needs to be shared between cases in the message handling loop.

#### Event Handling Loop

Graphical user interface, text

Description automatically generatedThis is simply an event structure. I hope you are familiar with these by now. All of the events handled by this structure (button presses and application closing) add their respective messages to the message queue. The error handling VI found here is pre-made.

#### Message Handling Loop

* **Initialize** – Establishes the control references for the front panel objects and bundles them into the data cluster. The ‘Initialize Data’ and ‘Initialize Panel’ messages are added to the message queue.
* **Initialize Data** – Sets the default values for front panel objects and bundles them into the data cluster.
* **Initialize Panel** – Unbundles the data cluster to set the initial states for front panel objects then adds the ‘Update Display’ message to the message queue.
  + Sets cursor position for all the graphs, clears the data from the graphs, enables the **Open** and **Exit** buttons. Disables all other buttons.
* **Open File** – Enables the **Save** and **Edit** buttons while disabling the **Set Cursors** and **Remove** buttons. The user is then prompted, via the file dialog, to select an LVM file. The data will then be extracted from the LVM using the Split Signals VI (describe in analysis section). The LVM data bundled into the data cluster for later use.
* **Save File** – The clipped data is saved to an LVM file. The default name is the same as the original. The **Remove**, **Set Cursors,** and **Edit** buttons are disabled. The **Open** button is enabled.
* **Edit File** – Moving the cursors is enabled. The **Set Cursors** button is enabled while the others are disabled.
* **Set Cursors** – The **Edit** and **Remove** buttons are enabled and cursor movement is disabled. The left and right cursor positions are bundled into the data array.
* **Remove** – Using the Remove Between Cursors VI, the data between the cursors is removed from the buffered data. The **Remove** and **Set Cursors** buttons are disabled. The **Edit** and **Save** buttons are enabled.
* **Update Display** – Simply unbundles the data cluster to update the graphical displays, the file name indicator, and the message on the UI.
* **Error** – If an error is thrown, it will be displayed to the user and the program will exit.
* **Confirm Quit** – Adds the ‘Exit’ message to the queue.
* **Exit** – Stops all the loops.

##### Remove Between Cursors SubVI

Timeline

Description automatically generatedThis SubVI uses express VIs to extract data from the whole signal. It takes the data left of the left cursor and appends it to the data to the right of the right cursor. Clipped data is wired out of the SubVI to be saved elsewhere.

#### Cursor Loop

Graphical user interface

Description automatically generated with low confidenceThis loop only controls the cursor positions on the bottom two graphs (pressure and flow). These cursors are dependent on where the user moved the cursors in the audio graph.

1. The program reads the Cursor List property node for the audio graph.
2. This array is converted to a cluster and unbundled into the x positions for the two cursors (CrsrList[0] and CrsrList[1]).
3. The x positions from the audio graph are bundled again.
4. The cluster is converted to an array.
5. The array is wired to the Cursor List property nodes of the pressure and flow graphs.

### LVM Fixer

A picture containing diagram

Description automatically generatedUsing the file path wired in from the main menu, the Recurse List File VI creates an array of file paths for all the LVM files in the folder. The number of files is set to be the maximum value on the progress bar. The outer for loop iterates through all the files in the array.

Qr code

Description automatically generatedThe data for each file is opened using the Open LVM Data VI. The data output from there is converted into a 2D array of doubles, then into a spreadsheet string, then into a 2D array of strings. Do not ask why, it just works.

Also in the outer for loop, the LVM file whose data is now opened is saved in a new folder named *Dirty Files* with the file name *OLD-(PreviousName).lvm.* At the end of the for loop, the edited data is saved to the previous file name and location.

The inner for loop iterates through the rows of the previously mentioned 2D array of strings. Using the Search 1D Array, the program looks for an NaN in a given row. If it is found, the row at that index is deleted from the 2D array of strings. I do not remember why this part is within a while loop, but I will not be changing what already works.

Diagram

Description automatically generated Once the for loop is done searching through rows and deleting ones with NaNs, the 2D array of strings needs to be reformatted into dynamic data (dark blue wires). To do this The 2D string array is first converted into a spreadsheet string. The spreadsheet string is converted into a 2D array of doubles. Each row of this array is converted into a waveform, which is then converted to dynamic data.

More information (the name of the signal data, the timestamp, the X Dimension, and time mode) are assign to the signal using the Set Attributes express VI. Most of these attributes are wired directly from the Get Dynamic Data Attributes VI found imediately after the Open LVM Data VI.

Once the data is prepared, all signals are merged and written to an LVM file. When complete, each file edited and the number of rows deleted from each will be displayed to the user. A while loop keeps the VI open until the user hits the stop button.

#### Open LVM Data SubVI

This VI is similar to the Split Signals VI that will be described later. First, the Read SPL Comment VI is run. This used to have other purposes, but here it is just used to get the names of the signals saved in the LVM. After that, the program checks if EGG is one of those signals. This is because not all saved files will have it and that needs to be accounted for in the following while loop.

A picture containing diagram

Description automatically generatedSince the data is saved in chunks during collection (one chunk per each iteration of the loop), it needs to be read out as such.

Each iteration of this loop opens a chunk of data and adds it to the previously opened chunk until the end of the file is reached.

The data is then merged back into a signle signal. If the original data file did not have any EGG data, a blank column is added to the signal.

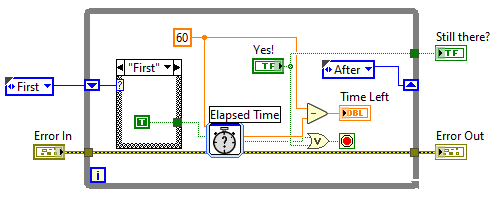
##### Read SPL Comment SubVI

It looks like I have already added comments to this SubVI, so I will just let you read those instead of rewriting them here.

Diagram

Description automatically generated

### Timeout Check

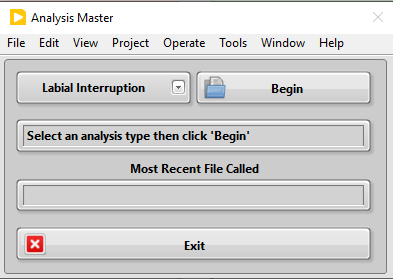
This pop-up uses a while loop and the Elapsed Time express VI to prompt the user with a countdown. In the first iteration, a True value is wired to the Elapsed Time VI reset node. This resets the amount of elapsed time. After the first loop iteration, a False value is wired to the reset node.

The displayed countdown is 60 minus the amount of time that has elapsed since the VI was reset. Once reached, a False value will be output to the calling VI through “Still There?” Boolean indicator.

# Master Analysis Program

## Front Panel Guide

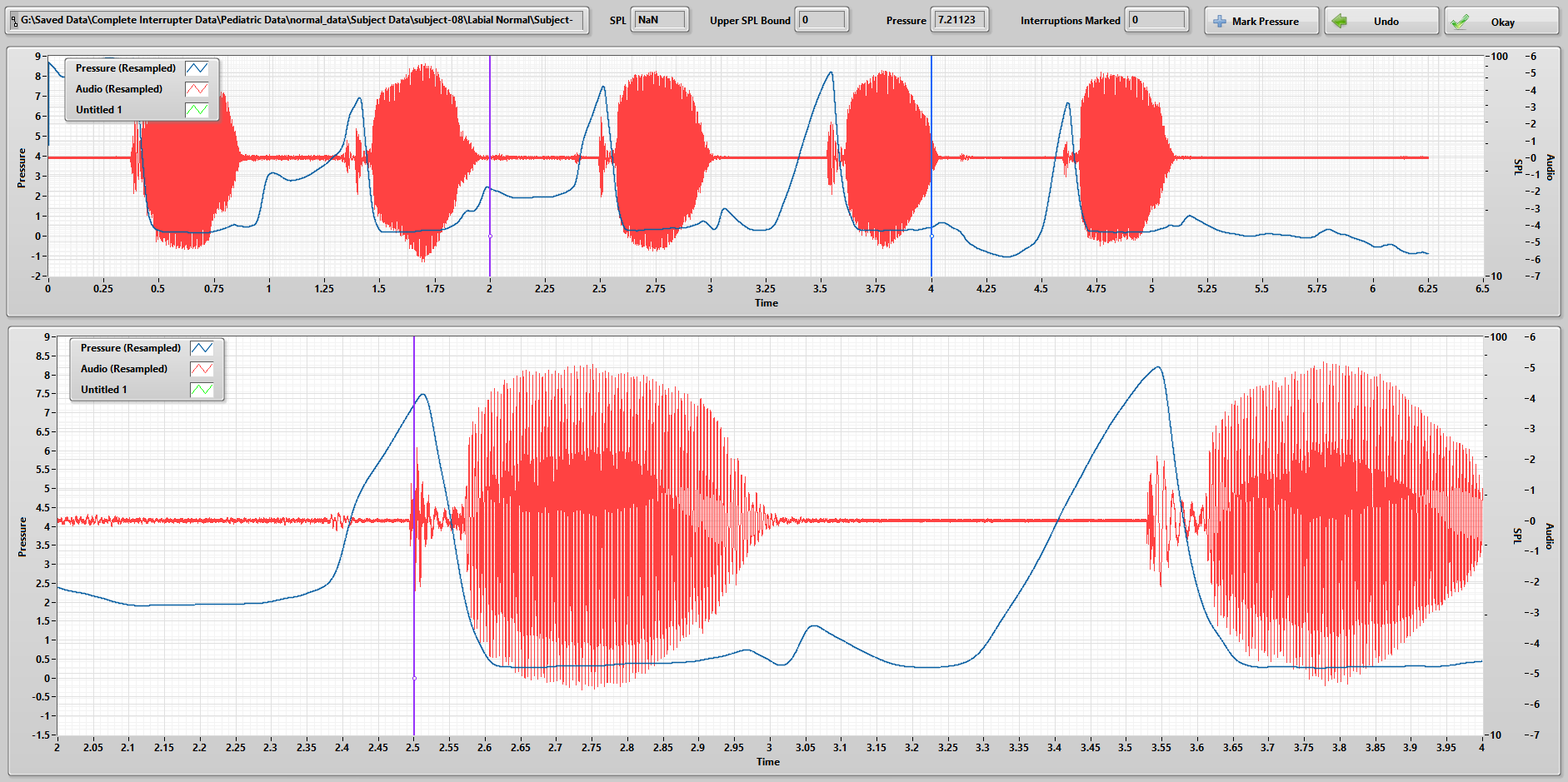
### Main Menu

The main menu is simple. The user just needs to select the type of data they want to analyze and click **Begin**. No matter which test type is selected, the user will always be prompted to select an LVM file before anything else occurs.

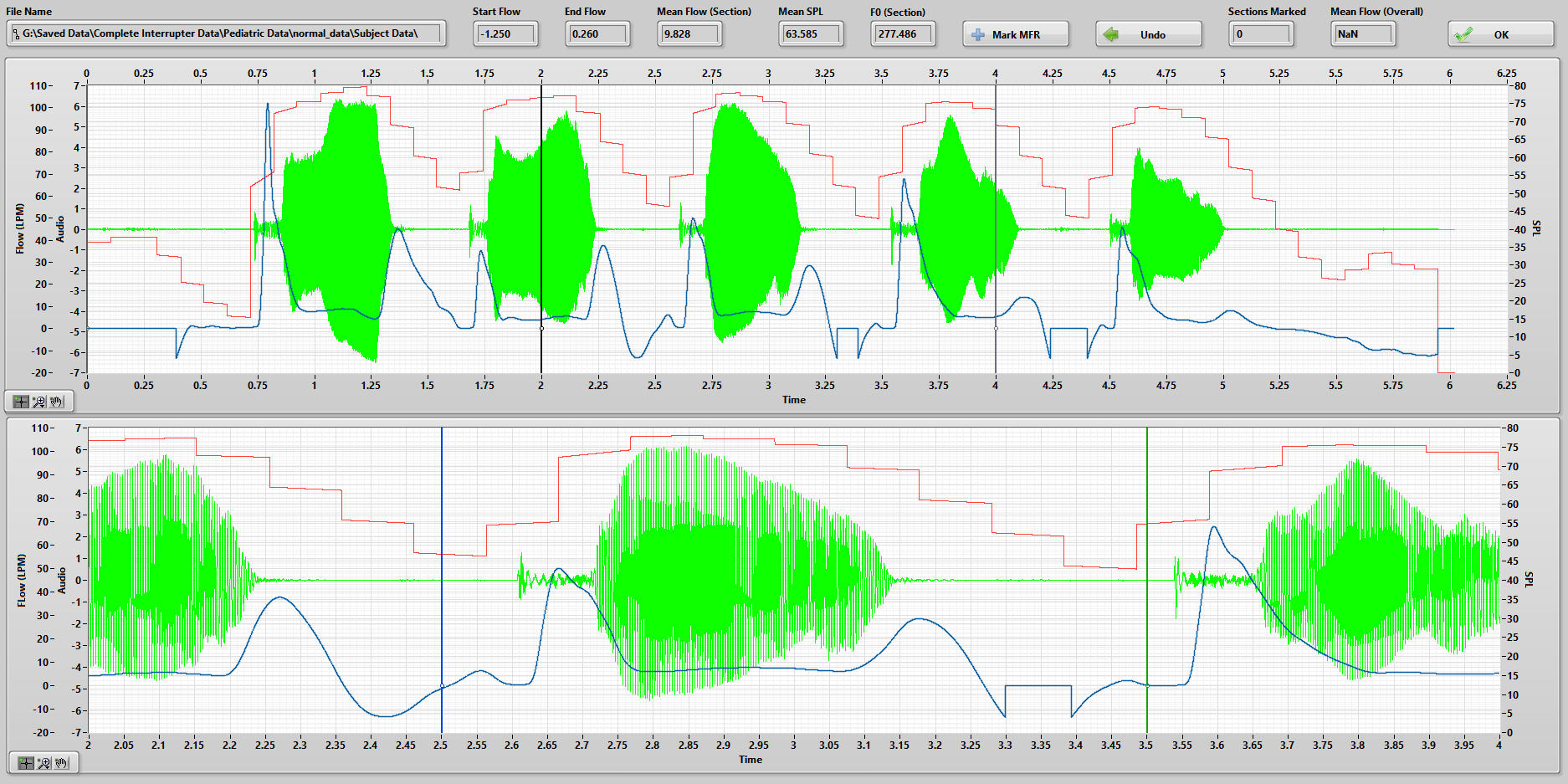
The display for most recent file called is to help the user remember where they are in analysis and avoid reanalyzing the same file multiple times.

Note: there is no analysis option for the signing helmet data. Analysis for this was done with a MATLAB program.

### Labial Interruption

The first interface that appears when analyzing labial interruptions is call Labial Pressure Marker. Here, the user moves the cursors on the upper graph to zoom in on each interruption, seen in the lower graph.

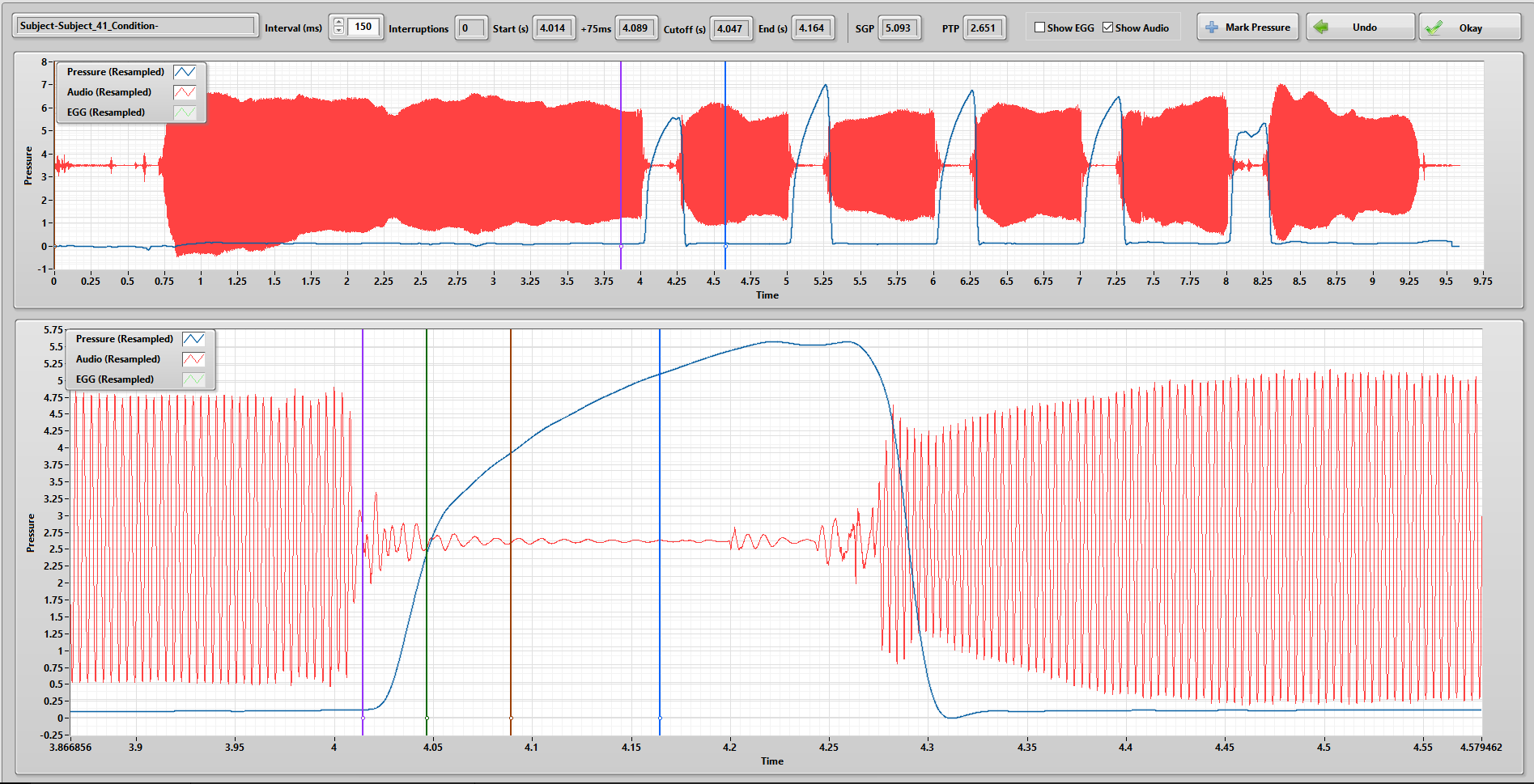
1. Move the cursor in the lower graph to the peak pressure that occurs immediately before phonation.
2. Once the peak is located, click **Mark Pressure.**
3. If you accidentally mark the wrong spot, you can click **Undo.**
4. Repeat steps 1 and 2 for all the peaks found in the data file.
5. Click **Okay.**

The next interface is simply called MFR. Similar to the Labial Pressure Marker, the user moves the cursors on the upper graph to control zoom in on each interruption, seen in the lower graph.

1. Move the cursors in the lower graph to a section where there is phonation.
2. Click **Mark MFR.**
3. If you accidentally mark the wrong spot, you can click **Undo**.
4. Repeat steps 1 and 2 for at least 3 phonation segments.
5. Click **OK**.

After MFR closes, the user will be prompted to save these data (MFR, SGP, SPL, and F0) to a new or existing text file.

### Mechanical Interruption

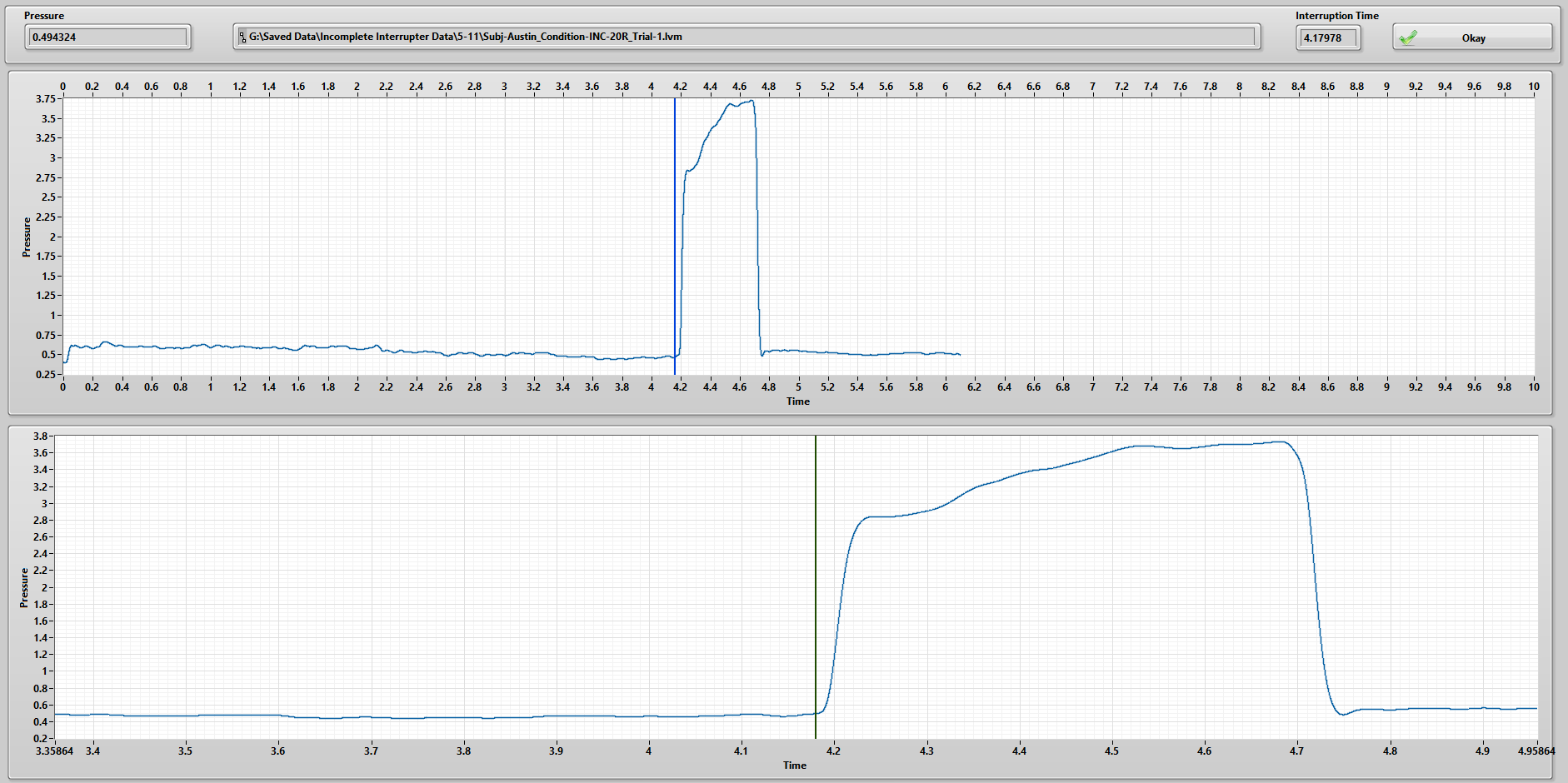
The first interface for analyzing mechanical interruptions is called Find SGP and PTP.

1. Move the cursors on the upper graph to zoom in on an interruption.
2. In the bottom graph, move the farthest left cursor (purple) to the spot right before where the pressure trace starts increasing
   1. The brown and blue cursors move with the purple one.
   2. The brown cursor marks 75 ms after the purple one.
   3. The blue cursor is after the purple at whatever the interval is set to (this is usually 150 ms).
3. Use the check boxes in the top bar to show acoustic and/or EGG data.
4. Using either the acoustic or EGG data, move the green cursor to where phonation is cut off (this can be tricky, refer to JJ’s analysis guide in the Aero Protocols binder)
5. Click **Mark Pressure**.
6. Repeat steps 2 through 4 for all the interruptions in the data file.

The next interface is the MFR analysis program. This is the same one used when analyzing labial interruptions. Please refer to that section.

After MFR closes, the user will be prompted to save these data (MFR, SGP, PTP, SPL, and F0) to a new or existing text file.

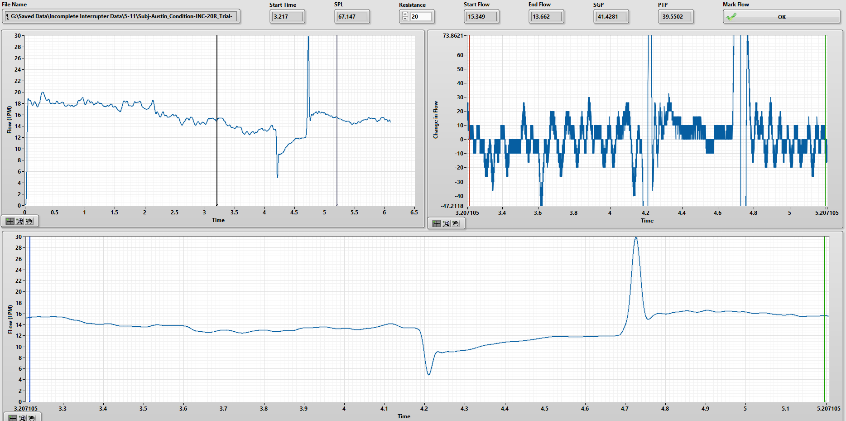
### Incomplete Interruption

This program was originally written back in 2016 and has not seen many updates since. Its purpose is to analyze a single long (500 ms) interruption. During the interruption, the subject will be phonating through a set resistance instead of completely blocking airflow. The user will be prompted to choose PTP or Not PTP.

If Not PTP is chosen, the above will appear. Further instructions can be found on the following page.

1. Move the cursor on the upper graph to change the bounds on the lower graph.
2. Use the lower cursor to mark where the interruption begins.
3. Click **Okay** when ready to move on.

In the following interface,

1. Set the resistance to be whatever was used during collection.
2. Use the cursors in the upper left graph to zoom in on the interruption.
3. Use the cursors in the upper right graph to mark a section of minimal changes in airflow.
4. Use the blue cursor in the bottom graph to mark a spot of steady flow just before the interruption and click **OK.**

If PTP is chosen at the beginning, the first interface will be similar to marking cutoffs for mechanical interruptions. The difference is that only one cutoff needs to be marked.

### Airflow Redirector

In this program, the user simply needs to move the cursors to mark where the max pressure of the trial is. This will be where the pressure in the tank equilibrates with subglottal pressure (read the redirector paper for more information).

After the pressure is marked, the MFR program will come up again. See the labial interruption analysis section for more details.

## Block Diagram Guide

In the following sections, we will go over the block diagrams for all the important VIs and SubVIs found in the Master Analysis Program.

### Main Menu

The main menu is organized into a state machine design pattern. The states are as follows: Initialize, Wait to Start, Analyze, Save, and Exit. The following subsections contain more details on each of these states (except Exit, since I hope that I can assume you know what happens there).

#### Initialize

This will always be the first state called when the main menu starts. Here, the startup message is added displayed to the user and the Wait to Start state is called next through a shift register.

#### Wait to Start

Here we find an event structure. There are only three events handled by this structure.

1. **Begin** – When the B**egin** button is pressed, the Analyze state is called.
2. **Exit** – The Exit state is called when the user presses the **Exit** button.
3. **Timeout** – The Timeout Check VI is called if no input is detected for two minutes.

#### Analyze

Graphical user interface

Description automatically generatedWhile this state may look complicated, it is just a bunch of nested case structures. The first case structure is controlled by whether the file dialog is canceled. If it is, nothing happens. If it is not, the next case structure is entered.

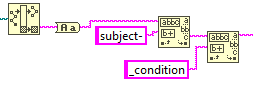
This case structure checks if the selected file exists. If it does not, nothing will happen, and a message is displayed to the user. If it does, the next case structure is entered.

The case that is used depends on which type of analysis is selected in the dropdown menu. Each type of analysis has its own SubVI, which will be covered in a later section. If any error is detected in the selected analysis SubVI, the next case structure adds an error message to the main menu display using a local variable.

Detection of an error also determines which state is called next. If there is an error, the initialize state is called. If there are no errors, the save state is called.

#### Save

A picture containing qr code

Description automatically generatedThe section of code to the left takes in the name of the analyzed file, makes it all lower case, then finds the string after ‘subject-‘ and before ‘\_condition’. This will output the subject number, assuming the default names were used to save collected data. The section of code to the right finds the trial number between ‘trial\_’ and ‘.lvm’. These are used later.

A picture containing schematic

Description automatically generatedThe case structure that follows has cases for each of the test types. They all function in the same way.

1. Format timestamp into a string to be the top of a new text file.
2. Add ‘Collection Method: *(method)*’ to the string for the new text file.
3. Creates the default name for the file ‘Subject-*#*\_*Test-‘*
4. Convert data into spreadsheet strings.
5. Combine the strings from steps 1 and 2 to create a single header string.
6. Combine the data strings and various labels to create one labeled data string.

The three strings compiled in this case structure are sent to the Save Data SubVI, where they are finally written to a text file. Additionally, a three-button dialog prompts the user to select if they want to save the data as a new text file, add it to an existing one, or cancel saving.

##### Check for Empty Data SubVI

Qr code

Description automatically generatedThis SubVI takes in an array of doubles and checks if it is empty. If it is, an empty string array is output. If it is not, the array is formatted into a spreadsheet string.

##### Save Data SubVI

The output from the three-button dialog is wired into this SubVI and it determines which of these following four cases is run case is run: Left Button, Center Button, Right Button, and Window Close.

The next page has more details on these cases.

###### Left Button

Diagram

Description automatically generatedThe left button is labeled ‘New’. Thus, this case saves the analysis data into a new text file. First, the user is prompted to enter their name. This labels the analysis file with who performed the analysis for later reference. The user is then prompted to confirm the save file and location. The default name for the file is created during step 3 of the save state. A new text file is then created that contains a header (step 5) and the analysis results (step 6).

###### Center Button

Diagram

Description automatically generatedThe center button is labeled ‘Existing’. Thus, this case appends the analysis data to the bottom of an existing text file. The user is prompted to locate the file they wish to add to.

Once selected, the file will be opened. The Set File Position function lets the function that follows (Write to Text File) that the incoming string should be added to the end of the file.

###### Right Button

The right button is labeled ‘Cancel’. I will not explain further.

###### Window Close

This button this case corresponds to is the little ‘X’ in the upper right of the dialog window.

### Analysis Sub-VIs

As mentioned above, the inner case structure for the main menu block diagram contains different SubVIs for each type of test. This section details each of those SubVIs.



Note: Some of the SubVIs found within these SubVIs are used more than once in different analysis types.

#### Labial Interruption

Diagram

Description automatically generatedThe labial analysis VI, like all the other analysis VIs, is just a couple of different SubVIs wired together. The Split Signals VI is found in all the other analysis VIs as well. Later subsections will go into more detail about all the SubVIs seen here.

Each SubVI only uses some of the collected data. The Pressure, Audio, and SPL data are merged and wired to the Labial Pressure Marker VI. Flow, SPL, and Audio are wired to the MFR VI.

##### Split Signals

I now realize that the Split Signals VI is almost identical to the [Open LVM Data VI](#_Open_LVM_Data), so just refer to that section for more details. In the future, I may replace one with the other.

##### Labial Pressure Marker

Diagram

Description automatically generatedDiagram, schematic

Description automatically generatedThis VI makes extensive use of property nodes and Cursor Lists, so you should be familiar with those. These first few nodes set the default positions for all the cursors that appear on the front panel.

Inside the while loop, cursor positions are read from the upper graph to control the bounds of the lower graph. This setup is common to most of my analysis programs, so you should be familiar with how it works.

The **Add** and **Undo** buttons trigger case structures that add and remove indices from the pressure array. The mechanical action for these is set to ‘Latch when Released’. This way, they revert to their default state (off/false) after the true value is read by the case structure.

##### MFR

This SubVI works in a similar fashion to the Labial Pressure Marker. It uses property nodes and cursor lists to read the cursor positions on the upper graph to control the bounds of the lower graph. Instead of taking the data at a single position, it averages the flow and SPL between the two cursors in the bottom graph. It also calculates the F0 for that region using the acoustic signal.

#### Mechanical Interruption

Only one SubVI here differs from those found in the Labial Interruption VI. The Labial Pressure Marker VI is replaced with the Find SGP and PTP VI.

##### Find SGP and PTP

Diagram

Description automatically generatedThis SubVI is very similar to the Labial Pressure Marker, so I will just go over the main differences. The first difference is shown below. Each iteration of the while loop, this code alternates between the setting the EGG and Acoustic traces as the active plot (depending on if the loop iteration number is even or odd). This is needed so the EGG and Acoustic signals can be shown or hidden individually.

Diagram

Description automatically generated with low confidenceThis SubVI uses 6 different cursors. The two in the upper graph control the bounds of the lower graph, similar to most of the other analysis VIs. Cursor [0] (marker for interruption start) and Cursor [3] (marker for phonation cutoff) are the only cursors the user can move in the lower graph. Cursor [1] and Cursor [2] are set relative to the position of Cursor [0].

Cursor [1] is set to be a variable distance after Cursor [0]. This distance is set by Interval variable. This is a type-def numeric control that as preset settings for different intervals.

Cursor [2] is always set to be 75 ms after Cursor [0]. This is just a visual reference.

#### Incomplete Interruption

Only the split signals SubVI is reused here. All the others are unique to incomplete analysis.

##### Incomplete Analysis Type Selection

Using an event structure, a ‘Yes’ string or ‘No’ string is output from this SubVI.

##### Pressure

This SubVI only has two cursors – one in the upper graph, one in the lower. The pressure at the position of the lower cursor is used to mark where the interruption starts.

Note: This SubVI is only called if the user selects ‘or No PTP’ when presented with options in the incomplete analysis type selection VI.

##### Mark Phonation Cutoff

This SubVI actually works the exact same way as the Pressure VI above. I do not remember why I have two separate VIs here. This may change in future updates.

Note: This SubVI is only called if the user selects ‘PTP’ when presented with options in the incomplete analysis type selection VI.

##### Flow

The main difference for this SubVI is that it uses resistance and subject of flow data to calculate SGP and PTP (PTP is only accurate if a cutoff pressure is entered). The SGP calculation occurs in the aptly named SGP calculator. PTP is simply the SGP minus the cutoff pressure, as it is with mechanical/complete interruptions.

###### SGP Calculator

Diagram

Description automatically generated This SubVI is the LabVIEW version of the equation below.

The value is divided by 60 because resistance values are in cmH2O/L/s, while flow values are in L/min

Note, this SubVI also assume that the second resistance (*Z­2*) is zero.

#### Airflow Redirector

The only SubVI for redirector analysis is the Max Pressure Find VI.

##### Max Pressure Find

This SubVI is very similar to those the that mark a single pressure (e.g., Pressure and Mark Phonation Cutoff). The main difference is that the cursor in the lower graph also shows the vertical position by way of a horizonal bar at the intersection of the cursor and the pressure trace. This is used to help the user find where the maximum pressure peak is among many different peaks.

# End Note

This concludes the written guide of the LabVIEW programs written for data collection and analysis. Hopefully, I have not missed anything. Descriptions of the MATLAB programs that JJ wrote can be found on his flash drive and on the Aero Drive found in the collection computer.