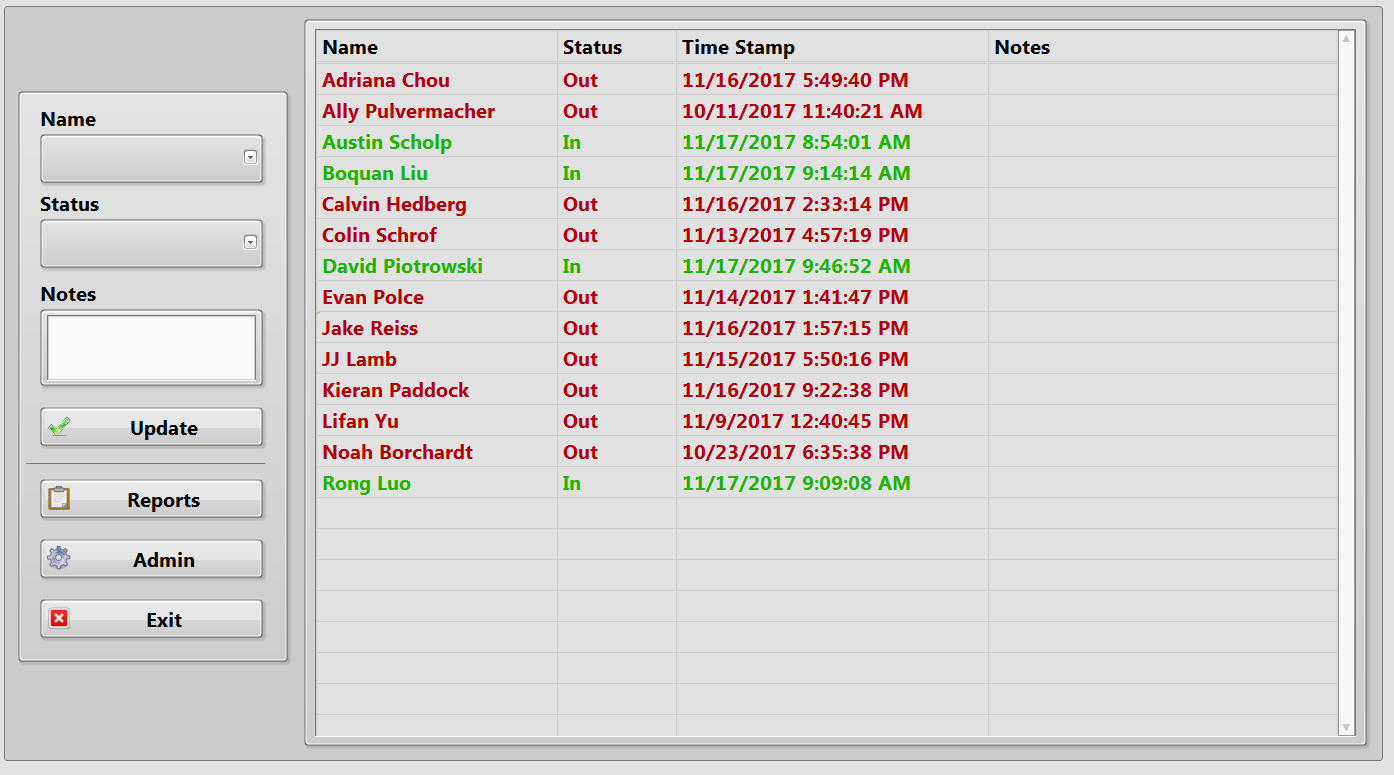
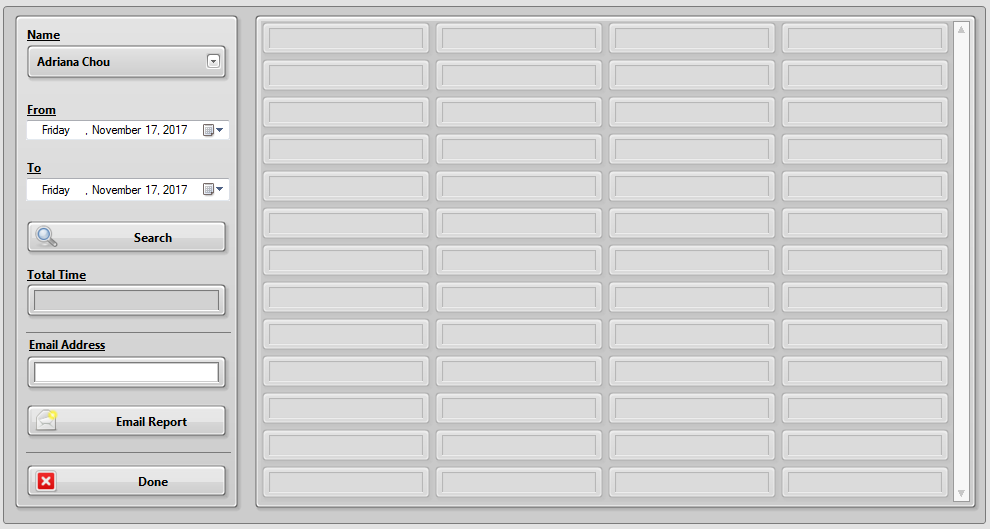
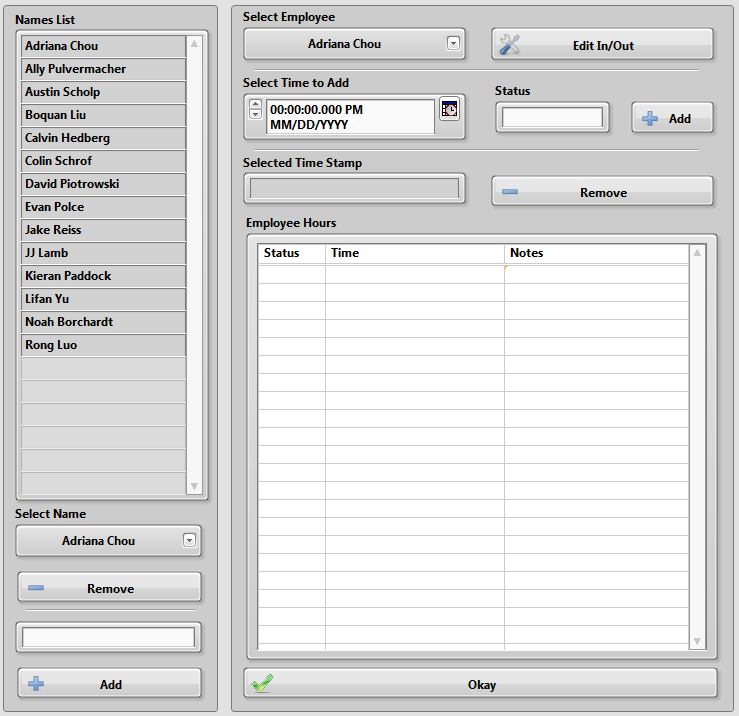
# Login Project

## Purpose and USe

This program was built to replace an old python-based, open-source, SQL-utilizing check-in program that has not had any type of update since 2008. It keeps track of when employees check in and out of the lab. It also can generate hour reports for each individual as well as a full report that includes hours for everyone.

The user interface is pretty simply. All one has to do it select their name, select their status (In, Out, Break, or Lunch), enter any notes if desired, then click Update.

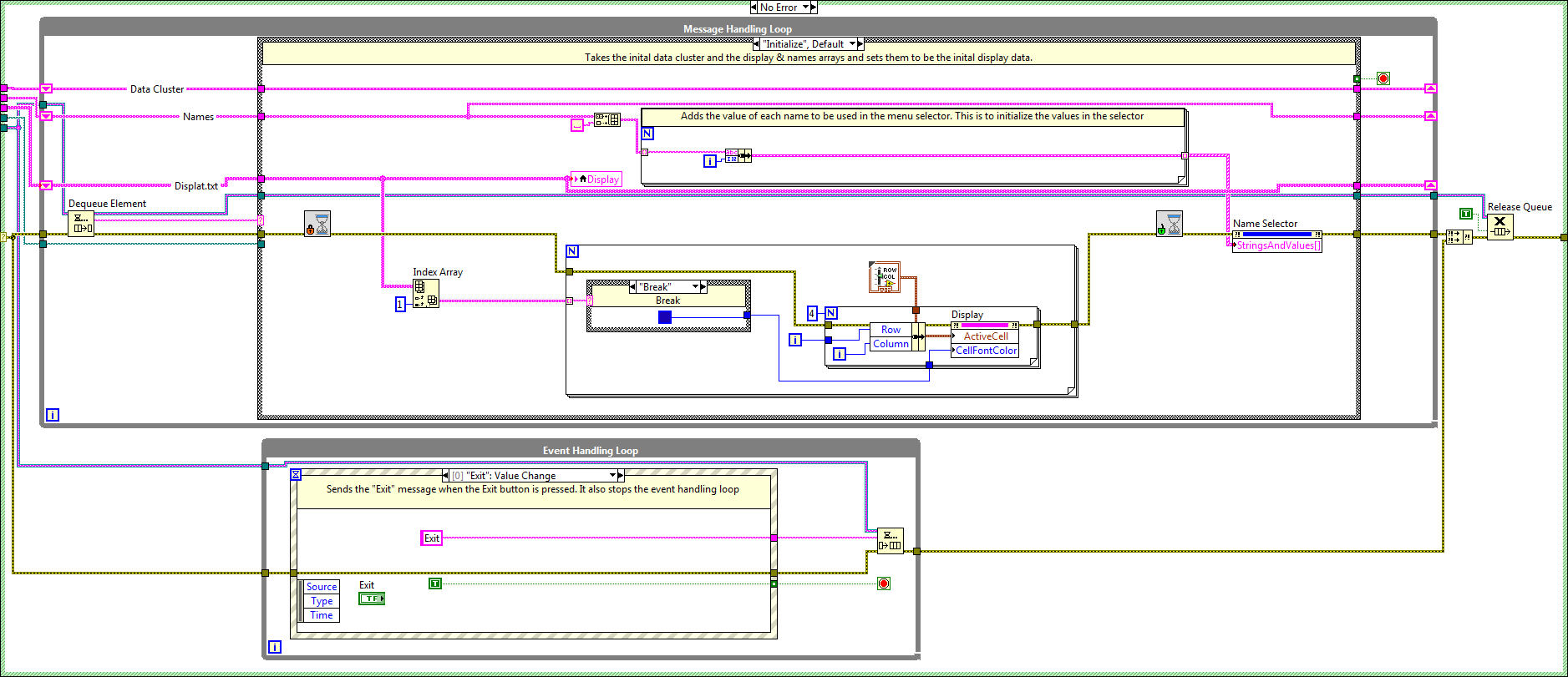
To check on your hours, click reports, select your name and the date range you are interested in, then click search. This report can be emailed as well.

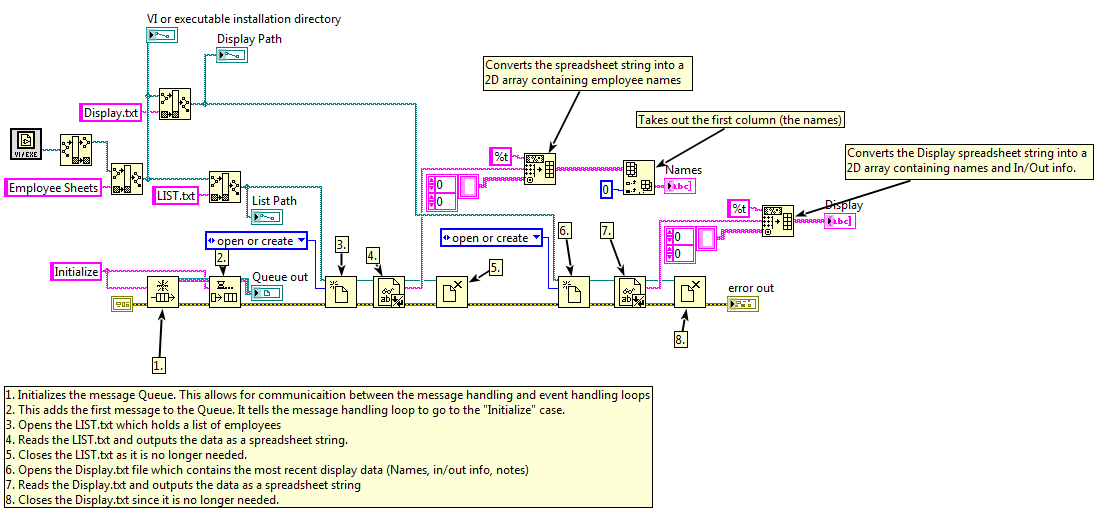
There is also an administrative mode that allows the admin to edit the hours that were input, as well as add or remove employees.

To remove an employee, select their name from the drop-down menu then click remove. To add an employee, type in their name and click add.

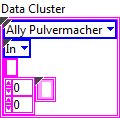
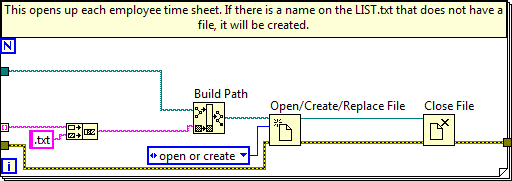
To edit hours, first select an employee, then click edit in/out. Once the hours are displayed, you can select an input by clicking on it in the generated list and click remove to delete the selected time input. To add an input, select a date and time, pick a status, then click add.

## The block diagram

The main part of this program is set up in a producer consumer structure with and event handling/message sending loop and a message handling loop.

There is an event case for each button: Update, Admin Edits, Reports, Exit. Before we get into each case, let us start with the Initialize Data SubVI. The comments I have made on this section should be able to walk you through the basics.

In general, the goals of this SubVI is to create a message queue and read the data from the List and Display text files. LIST.txt contains a list of all the lab members that use the check-in program and Display.txt has the most recent data that is displayed on the main front panel. It contains names, status, update time, and notes. The list of names and display data are output to arrays so they can be used by the program later.

I should also mention now, the data cluster used is made up of and enumerated list of names, an enumerated list of statuses, a string for notes, and a 2D array for display data.

After initializing the arrays, this for loop checks if there is an employee sheet for each name in the list.

Each employee sheet contains a record of all updates for that employee. The name of the employee is the same as the text file.

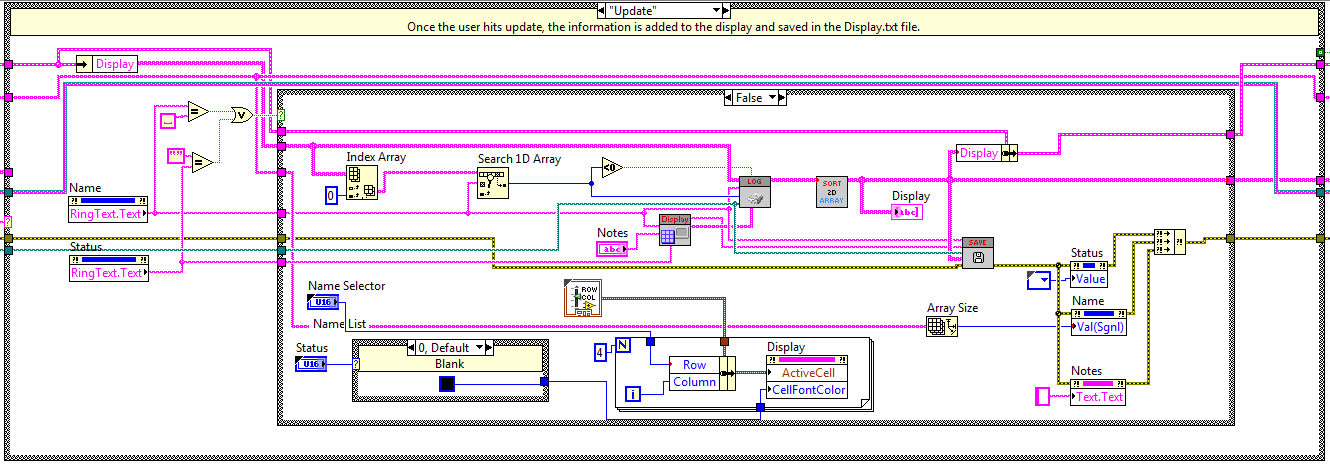
### Initialize

This is the first case to be called when the program starts. The bottom nested for loops search through the statuses of each employee on the list and changes all the text in their row to the appropriate color. The other for loop assigns numerical values to each name in the list.

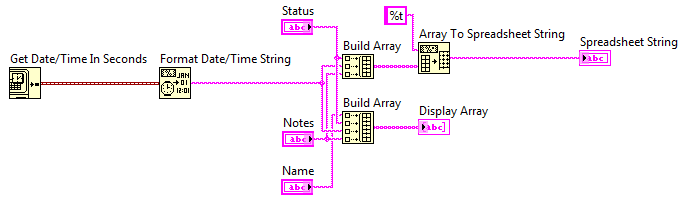
### Update

When the Update button is clicked, the update case is called. The first thing to occur in this case is a check to see if the Name or Status drop-down menus are blank. If either one of them are, then nothing will happen.

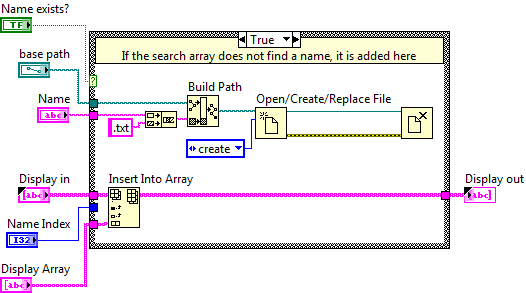
If a name and status is selected, the it will move on to the inner case structure. Wired to this structure (from top to bottom) is the Data cluster, 2D Display array, 1D Names List array, the selected Name string, the Base File Path, and the Status string.

Here, the Display and employee sheet text files are updated. Additionally, the color of the text for the selected employee changes based on the status. The SubVIs used here are “Build Display Array”, “Log Hours to Array”, “2D Array Sorter”, and “Save Display Files.”

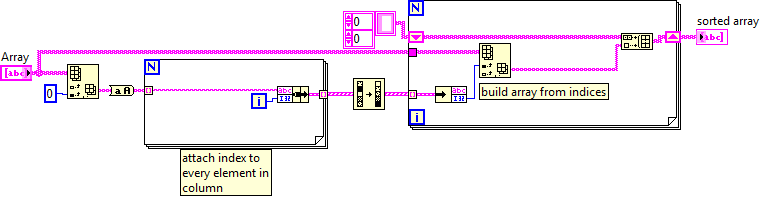
#### Build Display Array

This SubVI finds the current time then formats the time into a string. It then builds an array of the Status, Timestamp, and Notes. This is then converted into a spreadsheet string (to be added to the employee sheet). It also builds an array of Name, Status, Timestamp, and Notes (for the display).

#### Log Hours to Array

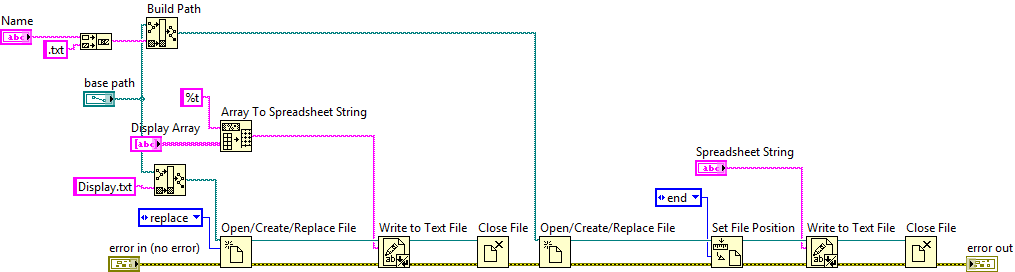
If in the main update case, the name selected does not match any in the list (search array index < 0), a new employee sheet is created for them. If there is a match, the updated 1D display array from “Build Display Array” is inserted into the 2D Display Array at the index of the matching name.

#### 2D Array Sorter

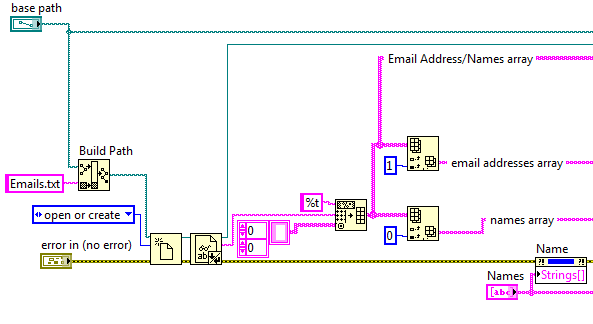


This SubVI takes in the 2D Display Array and sorts it by the index of each name. This is why we needed to assign numerical values to each name in the initialization of the VI.

#### Save Display Files

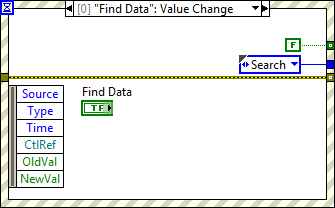
Here, the Display.txt file is updated with the new 2D Display Array (after it is converted to a spreadsheet string). The Status, Time, Notes array from “Build Display Array” is added to the end of the employee sheet of the selected name.

### Report

The report case (opened after the user click the Reports button) simply opens up the “File Hour Report” SubVI. There is another text file (Emails.txt) that contains each employee name and their email address. The first part of this SubVI opens this file and separates out the names and addresses into two 1D arrays.

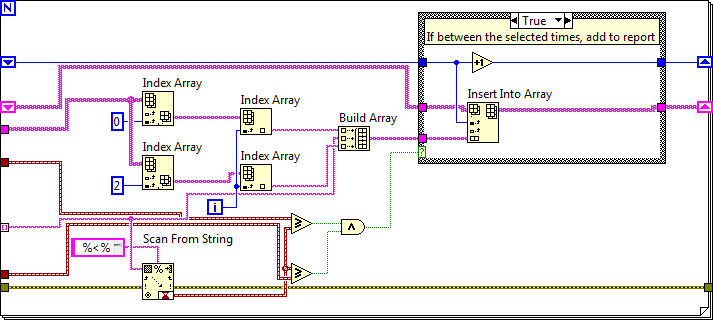
The rest of the program is set up in a state machine layout. The cases that it works through are “Wait”, “Search”, “Email”, “Exit”, and “Full Report.”

#### Wait

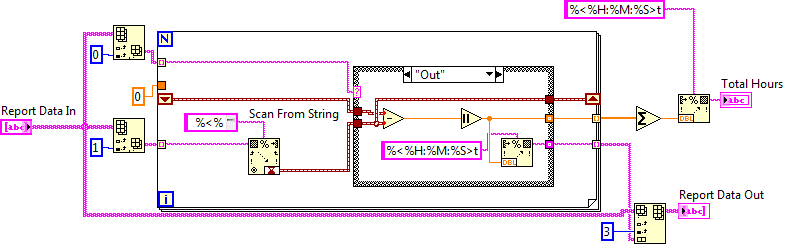
This default case contains an event structure that handles button presses for each of the buttons on the front panel (Search, Email Report, Done) as well as a case for if F9 is pressed.

F9 calls the Full Report case. This is so it is not readily accessible to anyone who does not know of it. All the other buttons are self-explanatory.

#### Search

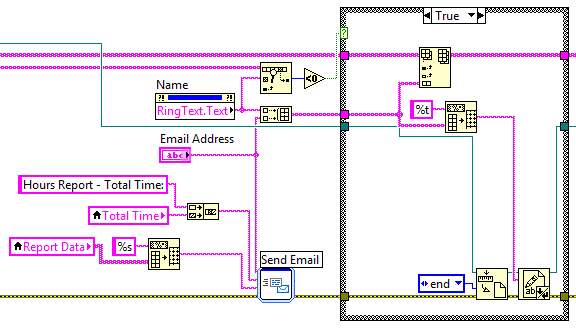
 The main feature of this SubVI is this for loop that iterates through the times in the selected employee sheet and checks if the dates are in between the Start/End dates selected by the user. If they are, it adds the Time, Status, and Notes to the report.

After the report is made, it runs through the “Hour Differences” SubVI. Here we have a for loop that iterates through each status on the report and for “Out”, “Break”, or “Lunch” status, find the time difference from the previous index in the report (this assumes that the previous status will be an “In” status). Once it works through the report, it sums to total differences and adds each difference to a new column in the report array. This is the final report array that is then displayed to the user.

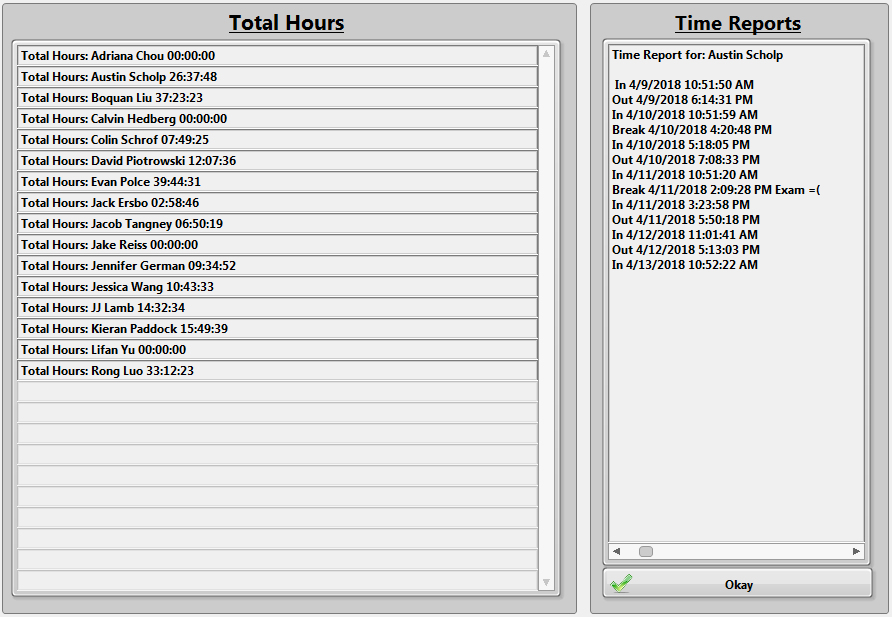


#### Email

This case takes the hour report, converts it into a spreadsheet string, and uses the Send Email ExpressVI to… send the report via email. The subject of the email is “Hours Report – Total Time -<HH:MM:SS>”

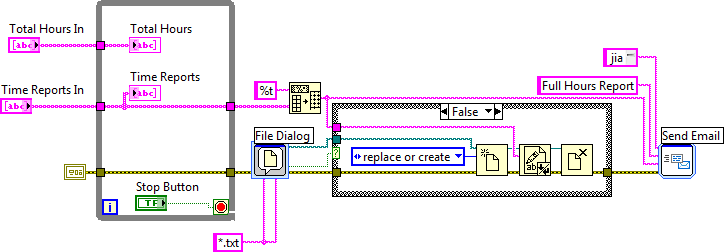
Also, if the selected name is not on the email address list, it is added through the use of the case structure shown.

#### Full Report

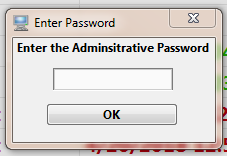
The process for this case is the same as the regular hour report except that it is inside a for loop that iterates through all the names on the list instead of making a report for a single employee.

The total hours for each employee are displayed on the left, and you can scroll through individual reports on the right.

It also creates a text file of the report and sends it to [jiang\_voicelab@yahoo.com](mailto:jiang_voicelab@yahoo.com). All of this is done through the “Full Report” SubVI. It displays all of the report data and waits until the user clicks the Okay button to stop the while loop.



### Edit

When the Admin button is pressed, the Edit case is called. It first asks for a password. Right now, the password is “password” because everyone else will guess “voicelab1.” If someone enters the wrong password, they will be presented with some Donald trump inspired responses. This was for no particular reason. I just felt like it.

If the password is correct, the Admin SubVI will open. Once again, here is a state machine program structure. The states are as follows: “Set Display”, “Wait”, “Edit In/Out”, “Add Name”, “Remove Name”, “Add Input”, “Remove Input”, and “Exit.” What each of these does is pretty obvious, but we will go into some details on how each works.

#### set display

The first case called sets the display seen earlier in this section of the binder. It is also called after some other cases when updates occur. While I did name it “Set Display” it really just opens the Display.txt file and takes that data and converts it to a 2D Array.

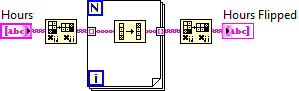
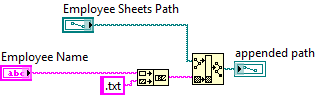
#### Wait

Here we find an event structure. It handles the clicks for all of the buttons. It also has the Mouse Up case (shown) that updates the ActiveCell property. This is important for removing inputs and the Time Select Loop.

When the user selects a cell on the spreasheet found on the front panel, the “Employee Hours” case is triggered and the ActiveCell proptery is updated. This loop is constantly updating Selected Time Stamp string. This is just the second column (LabVIEW is 0 indexed) of the Active Cell’s row.

The wait case also opens up the LIST.txt file, removes the empty name, sorts them and adds them to the Names List indicator as well as the Select Employee drop-down menu. The Employee Hours spreadsheet is also updated in this case.

#### Edit In/Out

When the Edit In/Out button is pressed, the employee sheet is found, opened, and added to the 2D Hours Array.

This array is flipped before the front panel indicator is updated so that it shows the most recent times first.

#### Add Name

After a name is typed into the string control and the Add button is pressed, this case will add that name to the LIST.txt and Display.txt files. This is done by adding a new array for that employee to the Display and List arrays, then re-sorting them to be in alphabetical order. A new employee sheet is also created.

#### Remove Name

When an employee leaves, use this case to remove them from the LIST.txt and Display.txt files. Using Search 1D Array functions, we can find the index of the selected name and remove the entire row from each 2D array.

#### Add Input

If an employee forgets to check in or out, the admin can add a time and status to their employee sheet. The 2D Hours Array is flipped so new additions are added to the end of the sheet. I currently do not have a sorter for the hours sheet but it is not needed since all of the hours are searched through when making reports.

#### Remove Input

Using the ActiveCell row and value, the row of the selected timestamp is removed from the array. The employee sheet is updated with this new data.

