Water Vapor Dial Labview Software

Brad Schoenrock
Robert Stillwell

NCAR
July 18, 2018

Contents

Chapter 1

Acronyms and Nomenclature

1.1 Acronyms and Nomenclature

- DB HSRL: Diode Based High Spectral Resolution Lidar
- T DIAL: Temperature Differential Absorption Lidar
- WV DIAL: Water Vapor Differential Absorption Lidar
- MCS: Multi-Channel Scaler
- UPS: Universal Power Supply
- HSRL: High Spectral Resolution Lidar

Chapter 2

Design

2.1 Overall Concept

The design goals for the labview architecture are as follows:

1. Improved fault tolerance
2. Improved data quality & collection uptime
3. Algoritm isolation for testing and run-time stability
4. Expandable
5. Flexibility to accommodate hardware configuration changes
6. Intuitive to new users

To accomplish these goals, the following concept is proposed. Small individual programs (children) are written to control conceptually seperate processes which interact with the WV DIAL hardware in various ways. In general no more than one child will interact with one piece of hardware to ensure that hardware faults are isolated. These children can run in parallel with proper synchronization and communication via their queues. If one child needs to talk to another it can open that child's queue and add an element onto it. A main container is used to call collections of children programs to accomplish various tasks, and open the necessary children automatically. These sub-functions are all called by the main function based on configure files that correspond to buttons on the main panel. The operational setups that are called by the main function are:

1. Warm up sub-function that brings all hardware to operational status. This includes things like warming the lasers and warming the etalons, which needs to be done before high quality data can be taken.
2. Main operations sub-function that performs all the mission critical hardware communication during data collection.
3. A template sub function which brings up an empty child with minimal functionality.
4. Switches sub-function which tests our ability to control the switches.
5. Temp. Scan sub-function which sweeps through temperatures to test the lasers.
6. Testing sub-functions for individusal controls to check operational status of hardware pieces such as the wavemeter (laser locking), the MCS operation, or the weather station.

This creates a 3 tiered structure: 1) Main function that opens sub-functions, 2) sub-functions that organize and manage children, and 3) children that actually set and control the state of each individual hardware piece. This creates a natural hierarchy that is predefined. New users unfamiliar with the operation of WV DIAL do not need to know how to control each piece individually but rather are guided through operation. Testing new hardware does not require complete integration with an all in one solution like is currently available, but can be built and tested independantly of the operations of the rest of the DIAL unit. This isolation creates fault tolerance because if one child fails, others are not waiting on that task and data collection for other children can continue unimpeded. Finally, knowledge of the hardware configuration is only needed at the higher levels (the main function and sub-functions) and can be more easily defined than a single all in one program.

2.2 Individual Element Controls

The proposed software update parses the main hardware control function into sub-functions. The sub-functions for individual controls are:

1. MCS
2. Weather Station
3. Laser Locking
4. Housekeeping
5. UPS
6. HSRL Oven
7. Wavemeter
8. Thor 8000
9. Quantum Composer
10. Power Switches
11. NetCDF

2.2.1 MCS

A sub-function that brings up two children. One does the communications via UDP to read the MCS, while the other is a set of controls to change the state of the MCS. These were split into two functions in order to prioritize the UDP communication so photon counting data was always running without interuption, and so that while the child was reading the UDP port the controls would continue to feel responsive. In a previous version of the MCS software putting the UDP communications in the same VI as the controls would lead to delays in the responsivness of the front panel due to the translation from a series of controls into a 32 bit hex word and back that was needed for MCS communications.

2.2.2 Weather Station

A sub-function that brings up the weather station child to monitor surface level temperature, pressure, relative humidity, and absolute humidity.

2.2.3 Laser Locking

A sub-function that brings up the laser locking routines that controls laser wavelengths and the etalons.

2.2.4 Housekeeping

A sub-function that brings up one child whose responsibility is to relay information about the temperature of the container. Thermocouples are placed within the container in various positions (which can be specified for writing into the data in the Configure_WVDIALPythonNetCDFHeader.txt) primarily to help ensure that the climate control for the unit is functioning properly.

2.2.5 UPS

A sub-function that calls the UPS child to monitor the state of the UPS Battery and power to the unit.

2.2.6 HSRL Oven

A sub-function that ....

2.2.7 Wavemeter

A sub-function that ....

2.2.8 Thor 8000

A sub-function that ....

2.2.9 Quantum Composer

A sub-function that ....

2.2.10 Power Switches

A sub-function that ....

2.2.11 NetCDF

A sub-function that ....

2.3 Back Panel Design

A template was made to standardize all labview control for WV DIAL. The design is to use queues to execute commands. This allows for an intentional bottleneck of execution such that user and automatically generated commands can not be executed out of order and there are a minimum of changes that need to be made to accommodate new hardware. The automatically generated and user generated commands are in the form of a string that is delimitated by an "_". The main loop cases are:

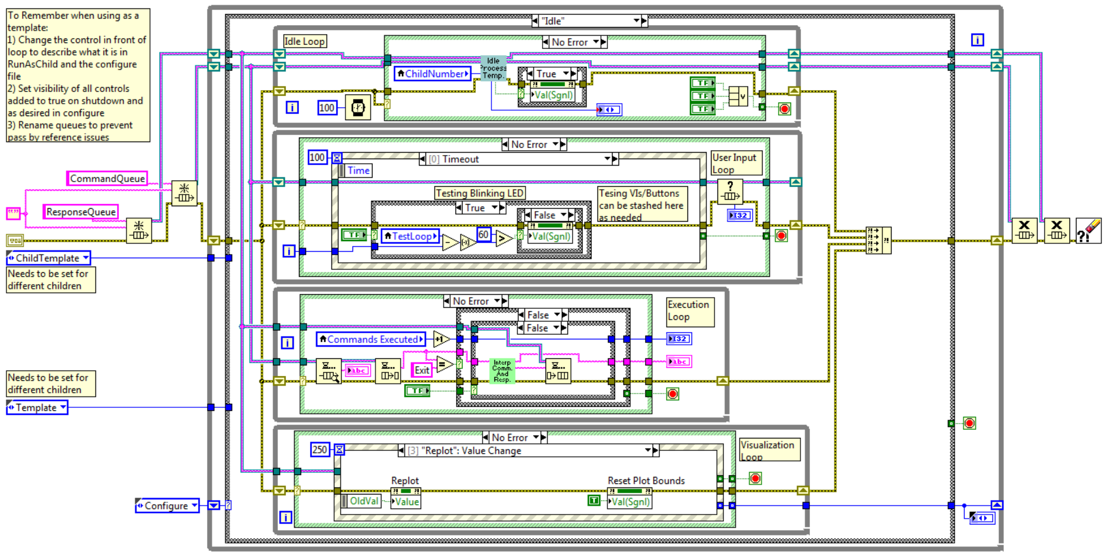
1. Configure
   (a) Reads default state out of the configure file
   (b) Sets the value of all available controls per the configure file
   (c) Sets the function visibility based on control settings like RunAsChild and permissions settings
2. Initialize
   (a) Communicate the initial state of the labview controls to the hardware or vice versa as appropriate
   (b) Create needed file folders for data saving
3. Idle

   (a) Performs baseline commands needed to keep program running
   (b) Waits for user commands to change the state of the system
   (c) Allows for raw data visualization as appropriate

4. Error Handler
   (a) Records all errors in an error log for debugging
   (b) Attempts recovery procedure if the error is recognized
   (c) Alerts the user if error are time sensitive and/or mission critical

5. Commanded Exit
   (a) Performs necessary steps to shutdown hardware that are specific to being told to shut down (only accessible in RunAsChild mode)

6. Exit
   (a) Performs necessary steps to shutdown hardware that are general to RunAsChild or not
   (b) Makes all hidden variables visible again for coding


To Remember when using as a template:
1) Change the control in front of loop to describe what it is in RunAsChild and the configure file
2) Set visibility of all controls added to true on shutdown and as desired in configure
3) Rename queues to prevent pass by reference issues

Needs to be set for different children

Needs to be set for different children

Figure 2.1: A picture of the idle loop stage of the VI template child.

2.3.1 The Command and Response Queues

We have two queues, Command and Response. The command queue is meant to hold user inputs and to process those serially, changing the state of operations. The response queue is meant to hold final data products meant to be written to disk and/or displayed to the screen. The names of these queues is very important. If two queues are running in seperate children with the same name both children will be attempting to put information onto the queues and removing information off of the queues at the same time. When unintended this will cause significant problems in operations. However, this behaviour can be used to facilitate communications between children without the use of global variables and is used to elimiate race conditions that global variables can have. The most prominent example of this usage is with the queue for advanced visualization which is called by different children to communicate the nessicary information to compute and display derived fields.

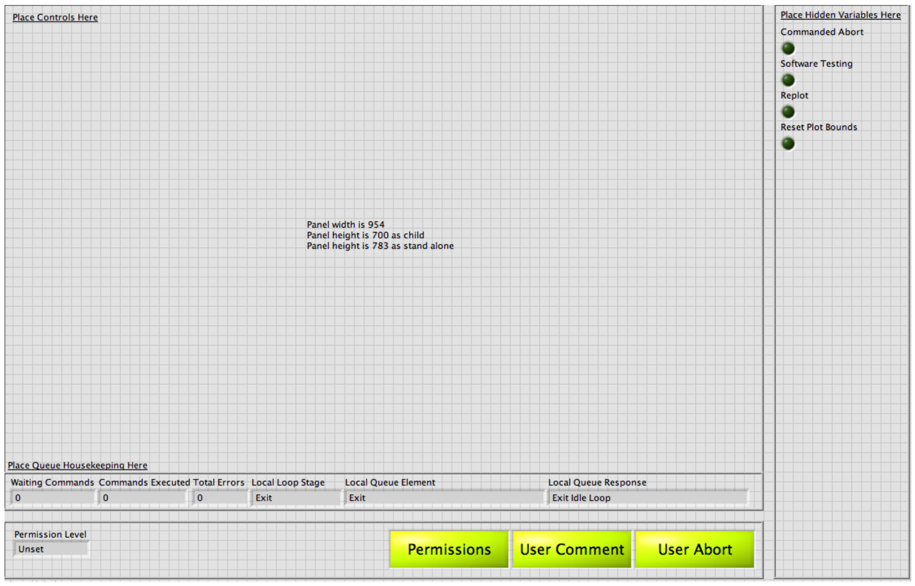
2.3.2 The Idle Loop

In the idle loop shown in Figure 2.1, 4 while loops run. The idle loop is executing commands that are needed to maintain communication with the main function and is used to automatically control the hardware. The user input loop is waiting for user commands and adding them to the command queue. The execution loop is processing elements off of the command queue and taking action within the Interpret Commands and Responses subVI. Critically, this function is called in re-entrant execution mode as it is copied in all functions. Lastly the visualization loop which is used to process hardware responses from the response queue, save the results to disk, and to print the results to the screen.

2.3.3 Run as Stand Alone

The multiple function concept requires one functional distinction. One concept is to run each piece manually, and another is to have a main function run each piece automatically, refered to as running in stand alone mode or running as child, respectively.

A template for the individual control elements is given in Figure 2.2. There are 3 main boxes, the first is in the top left (labeled as "Place Commands Here") for all of the user controls and everything the user needs to see. This will be different for each control. Within that box is a smaller box that would be the same for all controls that help locate the current status of the queue. The second box directly below would appear if the control were running as a stand alone but be hidden if the control were running as child. This really only needs to be in one place and is constant for all functions. The final box to the right is always hidden but is the storage place for all hidden variables that need to be defined but that the user does not need to see.


Place Controls Here

Panel width is 954
Panel height is 700 as child
Panel height is 783 as stand alone

Place Queue Housekeeping Here

Waiting Commands	Commands Executed	Total Errors	Local Loop Stage	Local Queue Element	Local Queue Response
0	0	0	Exit	Exit	Exit Idle Loop

Permission Level: Unset

Place Hidden Variables Here
Commanded Abort
Software Testing
Replot
Reset Plot Bounds

Figure 2.2: *The front panel of the main VI template. The software at runtime will hide the unnecessary controls and shrink the front panel window the the appropriate size indicated by the outer boxes. If the VI is run as a child (called from a higher VI) the Global Buttons disappear.*

This standardization is not critical for running all VIs as stand alone, however, the solution for running all the VIs as children in parallel does.

2.3.4 Run as Child

For the case that these individual controls are to be run automatically in parallel, an organizational problem arises. Assuming the code running the individual controls as children knows what functions to open and in what order, it is still possible that a user accidentally closes one function or a mess of VIs appears. To solve this, all front panels of the VIs running are projected into a single container seperated by a tabular control. The container itself has very little code running but is just there to contain all of the sub-VIs. This container is shown in Figure 2.3.

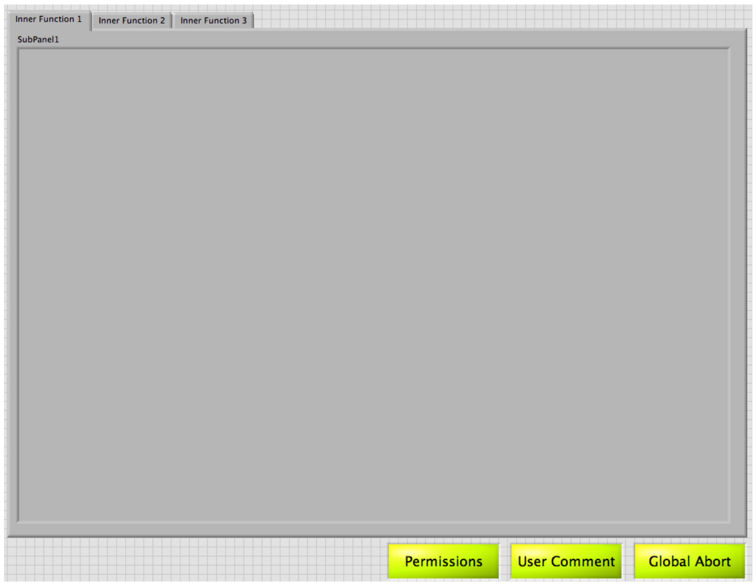


Figure 2.3: *The front panel of the main VI container. If multiple VIs are called and run simultaneously, their front panels are projected into the main container to allow for a simple control for the user but also allows maximum flexibility in VI execution.*

In this case, the controls that are common to all VIs, in the bottom box of Figure 2.2, are collected and included only once. Because the front panels of all VIs are to be projected into a single container, the front panel sizes of all VIs should be of a standard size otherwise the container would not simplify much. The container itself is simply a set of sub-panels that can each hold a single VI. The maximum number of VIs needed can be written if fewer are required, and the main program simply hides unused sub-panels.

2.3.5 Configure Files

Instead of using default variables in Labview, the default configuration for each sub-program will be stored in a configuration file. This ensures that operational hardware configurations are decoupled from any software upgrades or changes. Additionally, it makes it always possible to return the software to a known pre-set operational state and makes it more difficult for users to unknowingly modify the hardware's initial state. The configure files are version controlled via Git to track changes over time. An example of a configure file is given in Figure 2.4.

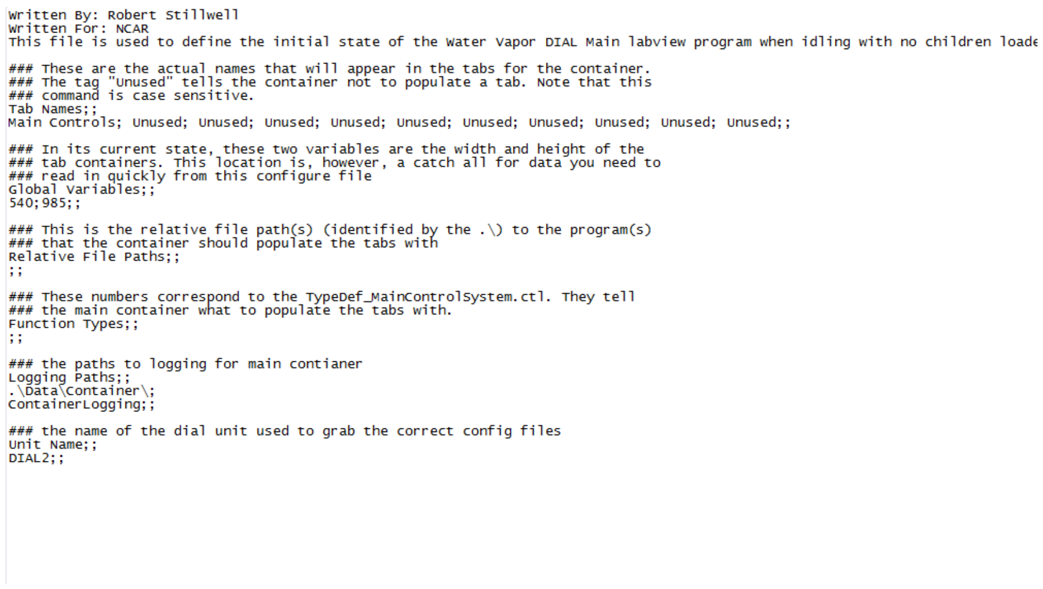
2.3.6 Log Files

Log files are to be kept to help track the status of WV DIAL and to help debug any anomalous states. The logs to be kept are:

1. Operations Logs: Describes when the system starts and stops and any user defined changes
2. Warning logs: These indicate failures in operations which does not inhibit mission critical functionality.
3. Error logs: These indicate failures in mission ciritical functionality and must be addressed imediatly.

2.3.7 Type Definitions

One of the unfortunate features of labview occurs when trying to modify the contents of a complicated structure. For example, if you initially make a cluster to define the state of a single laser and wish to add a piece to that structure, labview will show each connected wire as broken throughout the entire program. To



```
Written By: Robert Stillwell
Written For: NCAR
This file is used to define the initial state of the Water Vapor DIAL Main labview program when idling with no children loade

### These are the actual names that will appear in the tabs for the container.
### The tag "Unused" tells the container not to populate a tab. Note that this
### command is case sensitive.
Tab Names;;
Main Controls; Unused; Unused; Unused; Unused; Unused; Unused; Unused; Unused; Unused; Unused;;

### In its current state, these two variables are the width and height of the
### tab containers. This location is, however, a catch all for data you need to
### read in quickly from this configure file
Global Variables;;
540;985;;

### This is the relative file path(s) (identified by the .\) to the program(s)
### that the container should populate the tabs with
Relative File Paths;;
;;

### These numbers correspond to the TypeDef_MainControlSystem.ctl. They tell
### the main container what to populate the tabs with.
Function Types;;
;;

### the paths to logging for main contianer
Logging Paths;;
.\Data\Container\;
ContainerLogging;;

### the name of the dial unit used to grab the correct config files
Unit Name;;
DIAL2;;
```

Figure 2.4: *An example of the structure of a configure file. Variables are identified by a single name followed by two semi-colons. The next line has the required values delimitated by a single semi-colon with the end of line denoted by a double semi-colon.*

solve this, labview allows the user to define structure types and to link the controls on all panels directly to that definition. This allows the user to make a change in a single place and have all variables referenced to that place change at the same time.

This type definition is widely used for controls. It requires an extra step to make the definition then load it into the program much like defining a global variable but it makes updating and maintaining the code simpler and cleaner.

2.4 Sub-Functions and Calling

All of the individual controls are collected into a single container to run more complicated tasks. For example, the main operation of WV DIAL at full capability would require simultaneous control of the MCS, weather station, laser locking, etc.... The labview code would simply run the 6 individual controls responsible for each of these tasks and store them in a single container.

If a single VI fails, the other sub-functions are not affected by it. Its data would be missing from the global data however. For example, if the weather station communication failed, that data would be missing but it is not completely critical to have at all times. When its service could be restored, its data stream would be restored without interuption of other services. Details of how that data appears is outlined in Chapter 3.1.

Chapter 3

Data

3.1 Raw Data Output

3.1.1 Raw File Naming Convention

A naming convention for the files written is proposed as:

1. MCS Data: NCARWVDIAL###_MCS_YYYYMMDD_HHMMSS.cdf where ### is the number of the instrument, YYYYMMDD is the year, month and day of the file creation, and HHMMSS is the hour, minute, and second of file creation.

2. Weather Station Data: NCARWVDIAL###_WeatherStation_YYYYMMDD_HHMMSS.cdf where ### is the number of the instrument, YYYYMMDD is the year, month and day of the file creation, and HHMMSS is the hour, minute, and second of file creation.

3. Transmitter Housekeeping Data: NCARWVDIAL###_HousekeepingTx_YYYYMMDD_HHMMSS.cdf where ### is the number of the instrument, YYYYMMDD is the year, month and day of the file creation, and HHMMSS is the hour, minute, and second of file creation.

4. Receiver Housekeeping Data: NCARWVDIAL###_HousekeepingRx_YYYYMMDD_HHMMSS.cdf where ### is the number of the instrument, YYYYMMDD is the year, month and day of the file creation, and HHMMSS is the hour, minute, and second of file creation.

3.1.2 Main Data Product Changes

The main changes from previous architecture and how those changes help accomplish design goals are listed below.

1. NetCDF file types instead of binary files.
   (a) Pros:
      i. Expandable and customizable to different hardware configurations or improvements.
      ii. Can be easily converted to have outputs to mirror the GV HSRL or in CFRadial format.
      iii. Can be easily read and understood by non-NCAR personnel lacking knowledge of binary format.

   (b) Cons:
      i. Requires a change to the post processing ingest code.
      ii. As far as I can tell, only allows for a single unlimited variable making it not possible to stream data at different cadences to the same file efficiently.

2. Multiple files containing data instead of a single file per hour. The plan is to split the hourly files into a file containing: all photon counts and power measurements (measurements from the MCS), weather station data, laser housekeeping and locking data, and receiver housekeeping data. Additionally, on restart, a new netCDF file will be created instead of reopening the current hour's file.

    (a) Pros:

    i. Data for WV DIAL can be taken at different cadences. Though the current architecture can accommodate all data being taken at the same cadence, it is not necessary. There is no physical reason to take weather station data at 2 second resolution for example. By allowing multiple files, data can be taken at physically meaningful resolution and recombined in post processing.

    ii. Shrinks data files: Removing the requirement to take data at the same resolution removes redundant measurements.

    iii. Data transfer flexibility: Over limited data transfer links, such as a cell modem or at bandwidth limited sites, the most critical data can be identified and transferred while still saving all data for post processing.

    iv. Hardware changes or major changes requiring Labview shutdown can be easily tracked by file header information.

    (b) Cons:

    i. Data requires recombination to a single time grid in post processing.

    ii. The proposed split may require multiple VIs talk to the same hardware. This will require an access check before writing commands to make sure that two commands are not sent simultaneously that confuses the hardware.

3. Writing photon counting data at native resolution regardless of the state of the transmitter or receiver. This requires status bits being recorded instead of only recording data when all status bits indicate good data.

    (a) Pros:

    i. Data is not missed for one laser while other lasers are locking.

    ii. Data collection decoupled from all other data makes sure you are not loosing data for a reason other than MCS failure. For example, if the HSRL laser turns off, the system will be stuck trying to lock a laser it can't find. Meanwhile, WV data is lost that is not necessarily bad.

    (b) Cons:

    i. Data file must now include status flags that were not needed before.

3.2 Raw Data Ingest and Mating with Python Scripts

Steps of unpacking a single day of data:

1. Loop over hourly folders (I prefer to dump all data into a single daily directory but that is a convenience not a necessity)
    (a) Loop over hourly file types (more that one of each type can be written if the labview is started more than once in a single hour or restarted)
        i. Check if particular file contains data (by checking for zero-valued dimensions)
        ii. Loading raw photon count data
        iii. Loading laser housekeeping data
        iv. Loading weather station data
        v. Loading the operational parameters of the system and time stamps for each file type
    (b) Interpolate slower data to highest cadence data (likely housekeeping and weather station data to photon count time grid)
    (c) Collect all data into a single daily file with a single time dimension
    (d) Fill in missing data in time (if any time gaps exist) with nan values
    (e) Combine laser locking, receiver temperature, and all other data flags and create a single quality control flag: (Use Data, Data Warning, or Don't Use)

Chapter 4

Other

4.1 Other Labview Details

4.1.1 Front Panel

4.1.2 Back Panel

It is easy to forget the Labview standard back panel color coding scheme so It is included here for reference in Table 4.1.

Table 4.1: *Labview Standard back panel color coding scheme.*

Color	Type	Picture
Blue		
	Integrers	I32
	VI or Queue Reference	
Green		
	Boolean	TF
	File Path	
	Error Cluster	
	Robert's functions (might like to change this)	Time Stamp
Orange		
	Double	DBL
Pink		
	Cluster	
	String	abc
Purple	Hardware Resource	I/O

4.1.3 Priority Settings

Because the main structure of the software is based around queues, it is possible to remove elements from a queue before they are executed, put things on top of the queue in front of other commands, or put things at the bottom. Additionally, for performing commands that occur in steps, the queue is an ideal solution for organizing commands. Each of these possibilities is governed by a priority setting. For normal operation, the string commands are added to the bottom of the queue. If that queue is full or has several items in it, a standard command will not take precedent over other commands. The user abort command, for instance, has first priority. This command flushes the queue of current commands and inserts the set of commands to shut the system down.

4.1.4 Password Settings

NOTE: This functionality has not been built upon for Relampago, but the back end is in place for it to be added when desired. All commands are currently set up as accessable to all levels of user.

There are three password levels. They allow various things to happen such as bypassing the hardware warmup cycle or changing laser settings or turning off key components. Each level requires a password then the level permissions are hardwired into the code. When developed a list of password protected actions will be given here where the level is specified as either 1,2, or 3. Permissions on startup will begin at 0, which is recognized as unset. To prevent the user with a lower access level to attempt to use a sensitive command the main programs will set the visibility of certain controls such that they disappear with lower access.

The levels are complete access (or developer access), access to all but the most sensitive features (called administrative access), and general access. The passwords for each level are:

1. developer: %1543Lidar
2. administrative: !532Lid@r
3. general: SoundAndFury

Once a password level is recognized, all the privileges of the lower levels comes with the highest. That said, there are two passwords to downgrade access from a higher to a lower level. Note that if one is signed in with developer access and tries to downgrade to general access with the above given password, the password function will maintain the current access level. The two downgrading passwords are for completely unsetting the permissions and going back to general access. The passwords are respectively listed as:

1. Clear
2. Revert

All passwords are case sensitive.

There is one way to beat the password protection provided by the visibility and password functions which is possible but not likely. As the main function executes most of the user desired functions by creating and executing string commands, the main way to defeat the password options is to open and directly use the string execution commands. The commands are defaulted to do nothing if not entered exactly correctly but will execute a command if the user types the correct string commands. If for example, the string command is incomplete or the case sensitive commands do not match exactly what the execution function is looking for, an return string will note that the command is unrecognized.

A list of actions that require passwords and the level required that is proposed is given below.

Command	Level	Reason
example_command	Level#	Thing It Does