

# WATER VAPOR DIAL LABVIEW SOFTWARE

ROBERT STILLWELL

NCAR  
Advanced Study Program  
October 31, 2017

# Contents

<b>1</b>	<b>Acronyms</b>	<b>1</b>
<b>2</b>	<b>Nomenclature</b>	<b>2</b>
<b>3</b>	<b>Design</b>	<b>3</b>
3.1	Overall Concept . . . . .	3
3.2	Individual Element Controls . . . . .	3
3.2.1	Back Panel Design . . . . .	4
3.2.2	Run as Stand Alone . . . . .	5
3.2.3	Run as Child . . . . .	6
3.2.4	Configure Files . . . . .	6
3.2.5	Log Files . . . . .	7
3.2.6	Type Definitions . . . . .	8
3.3	Sub-Functions and Calling . . . . .	8
3.4	Main Function and Calling . . . . .	8
<b>4</b>	<b>Raw Data Output</b>	<b>9</b>
4.1	Raw File Naming Convention . . . . .	9
4.2	Main Data Product Changes . . . . .	9
<b>5</b>	<b>Raw Data Ingest and Mating with Python Scripts</b>	<b>11</b>
<b>6</b>	<b>Known Issues and How This Proposal Affects Them</b>	<b>12</b>
<b>7</b>	<b>Labview Details</b>	<b>13</b>
7.1	Color Coding . . . . .	13
7.1.1	Front Panel . . . . .	13
7.1.2	Back Panel . . . . .	13
7.2	Priority Settings . . . . .	13
7.3	Password Settings . . . . .	13

# 1 Acronyms

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

## 2 Nomenclature

## 3 Design

### 3.1 Overall Concept

The design goals for the proposed labview architecture are as follows:

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

To accomplish these goals, the following concept is proposed. Small individual programs are written to control conceptually separate processes which interact with the WV DIAL hardware in various ways. These small programs can run in parallel with proper synchronization and communication. A main sub-function is used to call collections of various individual programs and opens the necessary individual program automatically. These sub-functions are all called by the main function based on user input for what the software should do\*\*\*specify here\*\*\*. The operational setups that are called by the main function are:  
\*\*\*fill in list\*\*\*

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.\*\*\*split this up into pieces, don't have one master subfunction\*\*\*
3. Testing sub-functions to check operational status of individual hardware pieces such as the laser locking or 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 individual controls, and 3) Individual controls that actually set and control the state of each individual hardware piece. This overall structure accomplishes the main goals in a number of ways. There is 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 via button clicks. Testing new hardware does not require complete integration with an all in one solution like is currently available. The system is fault tolerant because all tasks are separated. If one task fails, others are not waiting on that task. Finally, knowledge of the hardware configuration is only needed at the higher levels, main function and sub-functions, and can be more easily defined than a single all in one program.

### 3.2 Individual Element Controls

The proposed software update parses the main hardware control function into many small functions. Having multiple small sub-functions running facilitates the new software design goals in a number of key ways. First, there is expandability because new VIs can be easily added that do not affect the old VIs. They do affect the methods in which the old VIs communicate however and require standardized communication practices. Second, fault tolerance is accomplished by splitting mission critical elements from non-critical elements. Third, hardware configurations are simple to accommodate; they only require the main function to know what is currently plugged in and where. Finally, testing of hardware can be done more easily and with fewer confounding variables as each piece of code does not rely on others to be running to be tested. A list of elements that would have their own controls are:

1. MCS setup, configuration, and data saving

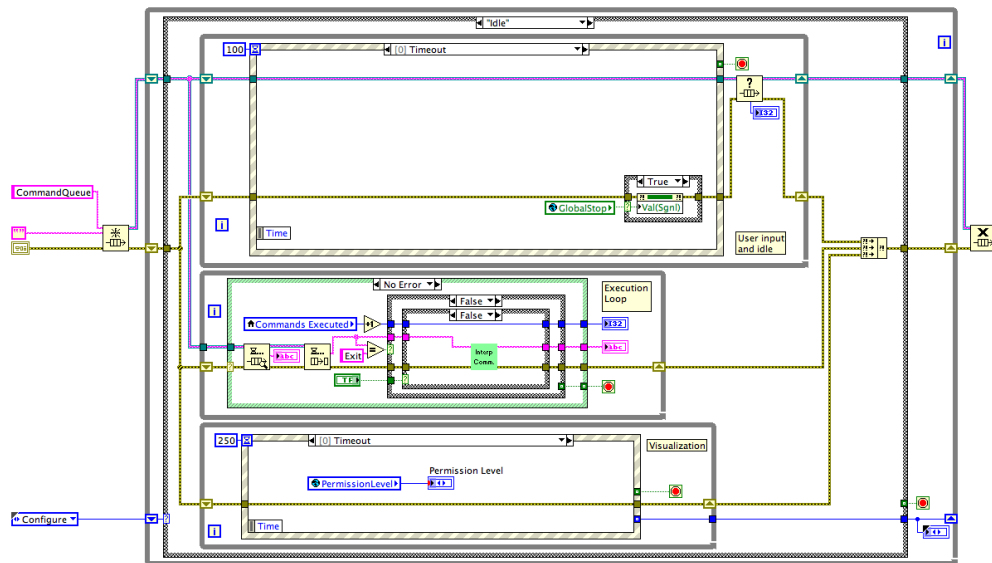
2. Weather station acquisition and data saving
3. Laser locking and housekeeping data saving
4. Receiver etalon temperature control and data saving
5. Data visualization
  - (a) Alignment profiles: raw data and background subtracted data
  - (b) Weather Station and laser housekeeping
  - (c) Retrieved profiles: optical depth, number density, absolute humidity, ...
  - (d) Retrieved contour plots: normalized relative backscatter, number density, backscatter coefficient, ...
6. Hardware Testing Routines
  - (a) Individual switch control: this has been used in the past to check on the status and wavelengths of the lasers as the lasers are being aligned or to force only a single laser into the TSOA or to test isolation and spectral purity
  - (b) Receiver wavelength scan: used to map the transmission of the receiver as a function of wavelength
  - (c) Receiver temperature scan: used to locate the transmission band of the etalon

### 3.2.1 Back Panel Design

A template was made to standardize all labview control for WV DIAL. The back panel of the template is meant to be a template as well as the front. The idea is to use a queue to execute commands using a single function. 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 delimited by a “\_”. 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
  - (d) Performs logging of any known or anomalous states
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
  - (c) Performs logging of any known or anomalous states
3. Idle
  - (a) Performs baseline commands needed to keep program running
  - (b) Waits for user commands to change the state of the system
  - (c) Performs all commands and communication with the global variables
  - (d) Allows for any data visualization
  - (e) Performs logging of any known or anomalous states
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 or mission critical
  - (d) Performs logging of any known or anomalous states
5. Commanded Exit
- (a) Performs necessary steps to shutdown hardware that are specific to being told to shut down (only accessible in RunAsChild mode)
  - (b) Performs logging of any known or anomalous states
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
  - (c) Performs logging of any known or anomalous states

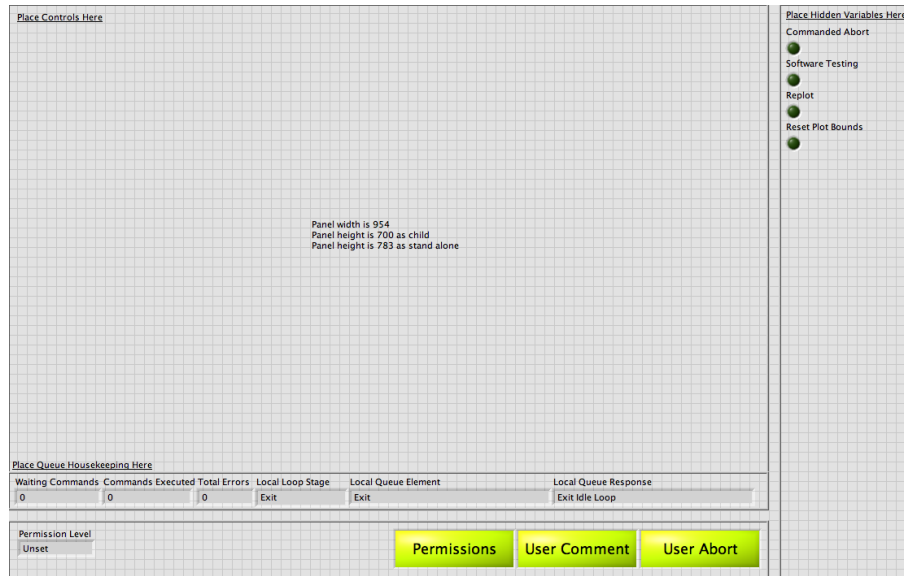


**Figure 1:** A picture of the idle loop stage of the VI template for the individual element control. In the idle loop, 3 while loops run. The top loop is waiting for user input or executing commands that are needed to maintain control of the hardware. The middle loop is waiting for user commands or idle commands and is executing them. Critically, this function is called in re-entrant execution mode as it is copied in all functions. The lowest loop is visualizing data as it is being recorded. The top loop should be filled with system specific commands relating to system specific controls.

### 3.2.2 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, referred to as running in stand along mode or running as child, respectively.

A template for the individual control elements is given in Figure 2. It is based on a queue structure in the back panel that allows both user commands and automatic commands to fill the queue that is then emptied by a single hardware function. 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



**Figure 2:** The front panel of the main VI template. The software at runtime will hide the unnecessary controls and shrink the front panel window to 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.

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.

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.

### 3.2.3 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. 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 3.

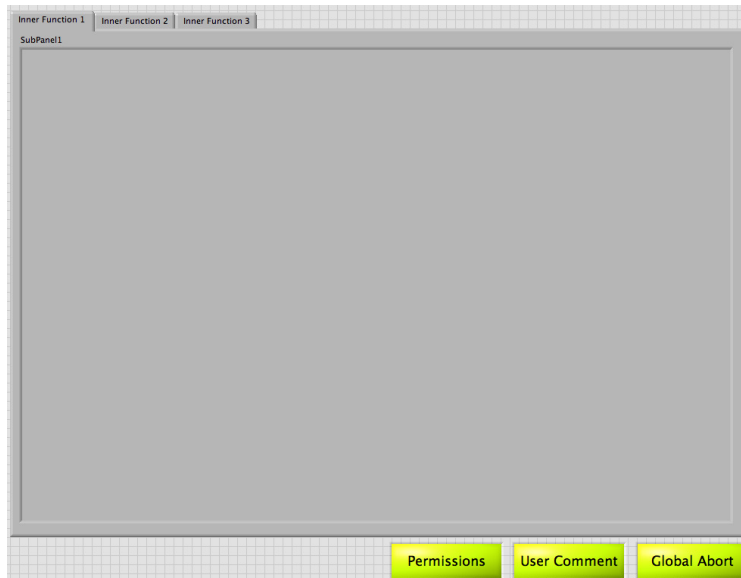
In this case, the controls that are common to all VIs, in the bottom box of Figure 2, are collected and included only once. Because the front panels of all VIs are to be projected into a single container, the front panel sizes of all VIs should be of a standard size otherwise the container would not 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 then if fewer are required, the main program can simply hide unused sub-panels.

To my knowledge, the only problem with this approach is that sub-panels can not be collected into an array. As such, the maximum number of contents is not a particularly dynamic element. I need to look into this more to see if this can be arbitrary.

### 3.2.4 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 current plan is to allow the user with the proper credentials (see Section 7.3 for more details) to write new configure files from Labview directly to save a current state. That state file can be version controlled via Git to track its changes over time. An example of a simplified configure file is given in Figure 4.





**Figure 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.

```
Written By: Robert Stillwell
Written For: NCAR
This file is used to define the initial state of the WVDIAL main function.

Tab Names;;
Main VIs; Testing VIs; System Status; Currently Running VIs;;

Global Variables;;
Global Variable 1; Global Variable 2; Global Variable 3; Global Variable 4;;

File Path Data;;
C:\Users\WVDIAL\Desktop\WVDIALData\Raw\;
C:\Users\WVDIAL\Desktop\WVDIALData\Zipped\;;

Global File paths;;
C:\Users\WVDIAL\Desktop\WVDIALLabview\DataLogs\OperationsLog.txt;
C:\Users\WVDIAL\Desktop\WVDIALLabview\DataLogs\SystemStatusLog.txt;
C:\Users\WVDIAL\Desktop\WVDIALLabview\DataLogs\UserCommentLog.txt;
C:\Users\WVDIAL\Desktop\WVDIALLabview\MainFunctions.llb\SystemWarmUp.vi;
C:\Users\WVDIAL\Desktop\WVDIALLabview\MainFunctions.llb\Operations.vi;
C:\Users\WVDIAL\Desktop\WVDIALLabview\TestingFunctions.llb\MCSTesting;
C:\Users\WVDIAL\Desktop\WVDIALLabview\TestingFunctions.llb\LaserTesting;
C:\Users\WVDIAL\Desktop\WVDIALLabview\TestingFunctions.llb\WavemeterTesting;
C:\Users\WVDIAL\Desktop\WVDIALLabview\TestingFunctions.llb\SwitchTesting;;
```

**Figure 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 delimited by a single semi-colon with the end of line denoted by a double semi-colon.

### 3.2.5 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 Log: Describes when the system starts and stops and any errors observed
2. System Status Log: Tracks the user defined changes from the configure status
3. User Comment Log: Allows for users to make comments about the operation of the system. An example would be if the system is found in an anomalous state and the debugging information used to recover or the reasons for a system status change.

### 3.2.6 Type Definitions

One of the most annoying 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 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 will be used for all 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 cleaner. One example of this is the control used to define the main loop cases from Section 3.2.1.

### 3.3 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 and housekeeping, and visualization. The proposed 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.

### 3.4 Main Function and Calling

The concept of the main function is simply as a separate function to organize all the possible software possibilities of WV DIAL and to serve as a watchdog function. The current plan is to have each sub level report a status to the level above it. The idea is to make each level report that it is okay. If not, the level above it can fire off a warning signal to a user that the software is not running properly. I imagine a status email being sent periodically when the status is abnormal for example.

## 4 Raw Data Output

### 4.1 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.

### 4.2 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 losing 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.

## 5 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 than 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 into a single flag

## 6 Known Issues and How This Proposal Affects Them

1. Data for on and offline wavelengths written in two different files. Matt mentioned this as a possible problem that I believe is linked to the serial nature of laser locking and data collection. By making the two asynchronous and including a laser locking flag instead of waiting for ti to complete, one can read data from the MCS and write data in 2 sets and not 4. Just have to make sure that a file can not be initialized before writing is complete.
2. Single errors crash the whole system. This overall design is more fault tolerant. Additionally, error logging will be included in the initial version such that we can record what is crashing the system and come up with systematic ways to deal with it.
3. Slow laser locking: This outline is agnostic to the type of locking but can certainly accommodate a new PID or simple integral locking scheme.
4. Blue screen of death! The hope is that making each communication a smaller piece allows for easier debugging of the culprit. (NOTE: moving to the new micro form factor computer may have solved this issue, continuing testing needed to confirm)

## 7 Labview Details











### 7.1 Color Coding

#### 7.1.1 Front Panel

#### 7.1.2 Back Panel

I always forget the Labview standard back panel color coding scheme so I thought I would just write it down. The things I think are needed are given in Table 1.

**Table 1:** Labview Standard back panel color coding scheme.

Color	Type	Picture
Blue		
	Integrers	
	VI or Queue Reference	
Green		
	Boolean	
	File Path	
	Error Cluster	
	Robert's functions (might like to change this)	
Orange		
	Double	
Pink		
	Cluster	
	String	
Purple	Hardware Resource	

### 7.2 Priority Settings

Because the main structure of the software is as a queue, it is possible to remove elements from the 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. However, if for example an error occurs, the error logger is given precedent over other commands because the current state of the system is sought. Furthermore, it is reasonable to assume that if an error occurs, some action will need to be performed in order to recover the system back to its operational state. Finally, the user abort command has first priority and has the ability to flush the queue of other commands. This command flushes the queue of current commands and inserts the set of commands to shut the system down.

### 7.3 Password Settings

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. A list of password protected actions is given here where the level is specified as either 1,2, or 3. 3 is the highest and 1 is the lowest. The hardware 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 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 do nothing 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
Writing configure files	2	Default states change but hopefully not by accident