

# WATER VAPOR DIAL LABVIEW SOFTWARE

BRAD SCHOENROCK  
ROBERT STILLWELL

NCAR  
July 27, 2018

# Contents

<b>1</b>	<b>Acronyms and Nomenclature</b>	<b>1</b>
	<b>Acronyms and Nomenclature</b>	<b>1</b>
1.1	Acronyms and Nomenclature . . . . .	1
<b>2</b>	<b>Design</b>	<b>2</b>
2.1	Overall Concept . . . . .	2
2.2	Back Panel Design . . . . .	3
2.2.1	The Command and Response Queues . . . . .	4
2.2.2	The Idle Loop . . . . .	4
2.2.3	Run as Stand Alone . . . . .	4
2.2.4	Run as Child . . . . .	5
2.2.5	Configure Files . . . . .	5
2.2.6	Log Files . . . . .	6
2.2.7	Type Definitions . . . . .	7
2.3	Children . . . . .	7
2.4	Individual Element Controls . . . . .	8
2.4.1	MCS . . . . .	8
2.4.2	Weather Station . . . . .	8
2.4.3	Laser Locking . . . . .	8
2.4.4	Housekeeping . . . . .	8
2.4.5	UPS . . . . .	8
2.4.6	HSRL Oven . . . . .	8
2.4.7	Wavemeter . . . . .	8
2.4.8	Thor 8000 . . . . .	8
2.4.9	Quantum Composer . . . . .	8
2.4.10	Power Switches . . . . .	9
2.4.11	NetCDF . . . . .	9
2.5	Sub-Functions and Calling . . . . .	9
<b>3</b>	<b>Data</b>	<b>10</b>
3.1	Raw Data Output . . . . .	10
3.1.1	Raw NetCDF Child Files . . . . .	11
3.1.2	Main CFRadial Merged Data Product . . . . .	11
3.1.3	Data Backups . . . . .	12
3.1.4	Eldora . . . . .	12
<b>4</b>	<b>Other</b>	<b>13</b>
4.1	Other Labview Details . . . . .	13
4.1.1	Front Panel . . . . .	13
4.1.2	Back Panel . . . . .	13
4.1.3	Priority Settings . . . . .	13
4.1.4	Password Settings . . . . .	14

# Chapter 1

## Acronyms and Nomenclature

### 1.1 Acronyms and Nomenclature

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

# 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. Algorithm 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 separate 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 individual 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 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 delimited 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

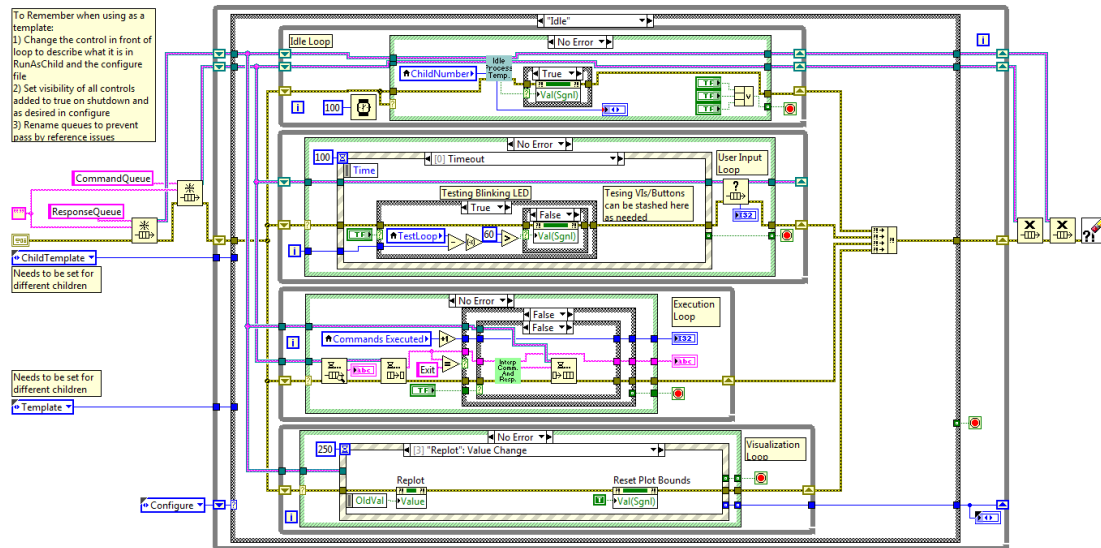


Figure 2.1: A picture of the idle loop stage of the VI template child.

### 2.2.1 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 separate 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 eliminate 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 necessary information to compute and display derived fields.

### 2.2.2 The Idle Loop

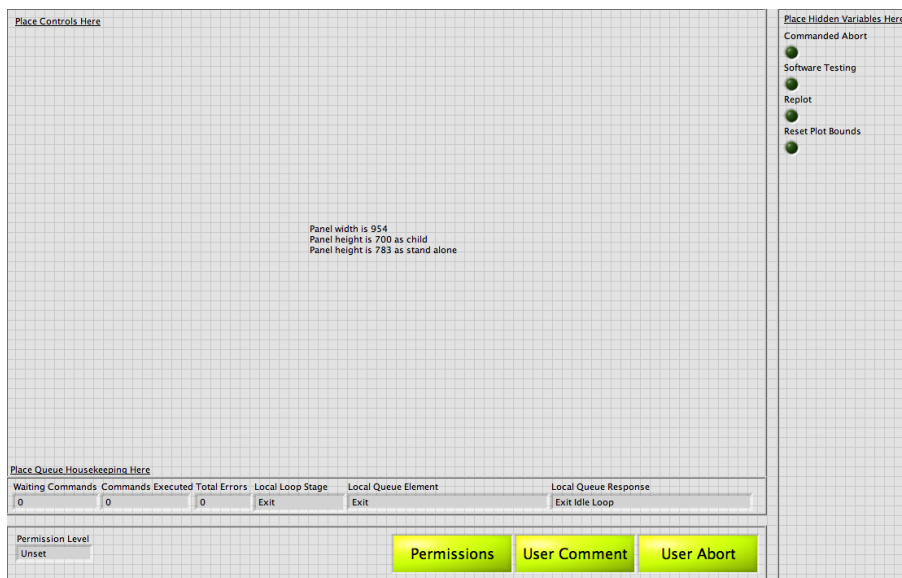
In the idle loop shown in Figure 2.1, 4 while loops run. The idle loop is executing commands that are needed to maintain communication with the main function and is used to automatically control the hardware. The user input loop is waiting for user commands and adding them to the command queue. The execution 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.

### 2.2.3 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, referred 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.



**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 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.

Each child is stored in its own library. They are WeatherStation.llb, WavelengthLocking.llb, QuantCompControl.llb, NetCDFWriter.llb, MCS\_NCAR.llb, MCSControls\_NCAR.llb, Logbook.llb, IndividualChildControl.llb, and AdvancedVis.llb. WVDIAL\_Main.vi is the definition of the main container and the main VI to start the software. The library ContainerResources.llb is used to support the main container. Furthermore there are some VI's which are useful to more than one child and are stored in SharedResources.llb. Some LabView objects are complex and need to be passed through several VIs, so type definitions are used to communicate the structure of those complex structures. Those TypeDefs are stored in TypeDefinitions.llb.

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

## 2.2.4 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 separated 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.

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.

## 2.2.5 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.



**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.

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.

```

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 loads

### 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;;

### 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.ct1. They tell
### the main container what to populate the tabs with.
Function Types;;
;;

### the paths to logging for main container
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 delimited by a single semi-colon with the end of line denoted by a double semi-colon.

## 2.2.6 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 critical functionality and must be addressed immediately.

### 2.2.7 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 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.

## 2.3 Children

Each child was created to separate and isolate functionality. These children were built based off of the template, but several differ from that design for specific reasons. Listed below are descriptions of how the children used for main data collection differ from the template design.

WeatherStation.llb contains files needed for operation of the weather station. It is perhaps the simplest of all the children and simply sticks to the design of the template to do so.

MCS\_NCARN.llb and MCSControls\_NCARN.llb when taken together form the data acquisition system and controls for the MCS. Because of the timing sensitivity and mission critical nature of the photon counting data, the MCS\_NCARN child has an additional loop whose sole responsibility is to monitor the UDP port communications and never allow itself to get interrupted from that goal. This means it has an additional UDP loop thread dedicated to monitoring the UDP port which is separated from the normal idle loop. Furthermore MCSControls\_NCARN is used to separate the controls from the hardware communication a layer further, so as to add an extra layer of safety to ensure that users fiddling with controls may be isolated from the reading of the UDP port. To this end any failure in the MCSControls will not interrupt the collection of photon counting data.

WavelengthLocking.llb \*\*\*\*\* - Robert will have to fill this one in more detail, as he wrote this and will have more insight into its operations - \*\*\*\*\*

AdvancedVis.llb seems at first to be straightforward. It pulls information from the MCS, the weather station, etc... in order to display composite data fields. This communication method however must remain true to the goals of isolating functionality and ensure that any failures in one child do not leak through to others. The way we do this (which is also done between MCS\_NCARN and MCSControls\_NCARN) is to establish queue communications where a child may add instructions onto the queue of another child. This requires the name of the queue in order to add to it, and as such the names of the queues should not be changed in order to maintain these avenues of communication. Advanced visualization gathers data from several sources so this method of communication is quite useful in this case.

IndividualChildControl.llb contains both the housekeeping child and the UPS child. Both of these children stick to the design of the original template quite closely. The UPS does, however, have routines for sending email messages in the case of loss of power to the unit so power outages can be addressed as quickly as possible.

NetCDFWriter.llb acts as a data compiler. It takes the raw txt and bin files written by each of the other data writing children and converts them into raw NetCDF files and finally into the final data product which is a merged CFRadial file. LabVIEW does not have the capabilities to write NetCDF files directly, so for this purpose a python subroutine is called in order to deliver the final data products. These data products are described in Chapter 3.

## 2.4 Individual Element Controls

The proposed software update parses the main hardware control function into sub-functions. These sub-functions serve to control individual elements of the WVDIAL, serve as simplified routines to warm up elements of the WVDIAL, or are to test out specific functionality in isolation of the rest of the unit.

### 2.4.1 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 interruption, 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 responsiveness 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.

### 2.4.2 Weather Station

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

### 2.4.3 Laser Locking

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

### 2.4.4 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`. This is primarily to help ensure that the climate control for the unit is functioning properly.

### 2.4.5 UPS

A sub-function that calls the UPS child to monitor the state of the UPS Battery and power to the unit. The UPS child has a subroutine to automatically send out an email when the UPS Battery gets too low.

### 2.4.6 HSRL Oven

A sub-function that warms up the HSRL. This is not currently built, but the button on the front panel is there for the addition of the feature in the future.

### 2.4.7 Wavemeter

A sub-function that brings up the wavemeter to read the wavelengths of the lasers.

### 2.4.8 Thor 8000

A sub-function that controls the Thor 8000 laser diode current control module.

### 2.4.9 Quantum Composer

A sub-function that controls the Quantum Composer timing unit. For the Relampago release the only functionality is to write, the read function does not work. When writing to the QC you may have to click through a couple pop ups in order to successfully set the state of the QC.

#### **2.4.10 Power Switches**

A sub-function that controls the power switches.

#### **2.4.11 NetCDF**

A sub-function that brings up the NetCDF writer for reprocessing of data files.

### **2.5 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 runs the children responsible for each of these tasks and stores them in a single container.

If a single VI fails, the other sub-functions are not affected by it. Its data would, however, be missing from the final data product. For example, if the weather station communication failed, that data would be missing but it is not mission critical to have at all times. When its service could be restored, the weather station data stream would be restored without interruption of other services. Details of how that data appears is outlined in Chapter 3.

# Chapter 3

## Data

### 3.1 Raw Data Output

Within the codebase is a location for data storage, the directory is called Data. Each child that writes data has a subdirectory, MCS for the MCS, Housekeeping for the thermocouples, UPS for the UPS, etc.... Within these subdirectories are daily subdirectories in the format YYYYMMDD, and within these daily directories are the data files broken up by hour. Within the MCS daily directories, as an example, are MCSData files with names formatted as MCSData\_YYYYMMDD\_HHHHHH.bin and MCSPower files with names formatted as MCSPower\_YYYYMMDD\_HHHHHH.bin. The hour formatting of these names is in fractional hours in UTC, so a time of 015000 would be 1.5 hours into the UTC day and would be a file that began at 1:30 UTC.

The MCS files are stored in binary format in order to save space, and are a log of the hex responses from the MCS with a header for each entry that contains the timestamp as well as channel assignments that were set in LabView. These are decoded by the python routines stored in the NetCDFPython directory and is automatically called by the NetCDF child when data is collected via main operations. The format for each data entry is - bytes 0-8 stores the timestamp, bytes 28-96 store channel assignments, bytes 111-112 stores the number of profiles per histogram, byte 114 stores the channel number and the sync bit from the MCS, bytes 115-116 store the number of counts per bin, bytes 117-118 store the number of bins, bytes 119-121 store the relative time, byte 122 stores the frame counter, and then each bin is read from 4 byte blocks for nBins. The first three bytes of each bin is the photon counting information, while the 4th byte contains the channel number and the sync bit for that transmission. Once all nBins have been read there will be a 4 byte footer word “ffff” and 8 bytes of space for delimitation to the next data transmission. Some seemingly unused bytes exist in the header, those are either to help delimitate information when visually inspecting the files, or are unused bytes of the MCS header.

MCS Power files share a similar but simpler structure to the MCS data files. Bytes 0-8 store the timestamp, bytes 23-67 store the channel assignments, bytes 82-85 store the relative time, and each channel power is stored in 4 byte increments, with the first three bytes being the power measurement and the fourth byte containing metadata on the returns. Again seemingly unused bytes are used to help delimitate information when inspecting the files or are unused bytes of the MCS header.

Further information about the contents of the MCS header can be read either from the MakeChildFiles.py function within this codebase, or from the MCS design documentation.

Laser Locking data files are stored in a text format, with the columns corresponding to Laser number, Wavelength measured, Difference in wavelength from desired, a boolean 0 or 1 for if it is locked or not, the desired temperature, the measured temperature, the current, the timestamp, and a date.

Etalon data files are stored in a text format, with the columns corresponding to Etalon number, Temperature, Temperature difference from desired temperature, a boolean 0 or 1 for if it is locked or not, a timestamp, and a date.

Weather Station data files are stored in a text format, with the columns corresponding to Temperature, Relative Humidity, Pressure, Absolute Humidity, and a Timestamp.

UPS data files are stored in a text format, with the columns corresponding to Timestamp, A boolean 0

or 1 for if the battery is behaving nominally, a boolean 0 or 1 for if the battery should be replaced, a boolean 0 or 1 for if the battery is in use, a boolean 0 or 1 for if the battery is low, a percentile number for battery capacity, an estimate for how much time is left for battery operations in fractional hours, the temperature of the UPS, and the number of continuous hours it has been on battery.

Housekeeping data files are stored in a text format, with the columns corresponding to a timestamp, and then an entry for each thermocouple. The file will have as many columns as there are thermocouples plus one column for the timestamp which is the first column. Locations of the thermocouples can be recorded into the final data products by creating an entry in the config file `Configure_WVDIALPythonNetCDFHeader.txt`. Entries for this purpose should be formatted as `thermocouple_location_#` followed by a description for the location. This config file is tab delimited.

The txt and bin files are the first and rawest form of data available. As long as you have these files the rest can be derived from them. The txt and bin files are then processed by the NetCDF child via some python scripts on board the WVDIAL unit's MFF computer. The main python script is `NetCDFScript.py` which sets up the needed variables and calls each of the other python scripts to perform specific functions.

### 3.1.1 Raw NetCDF Child Files

The function `MakeChildFiles.py` is used to translate these misc. txt and bin files into a common NetCDF format. NetCDF4 was chosen for this purpose. This function translates the data files as they were written and performs no manipulation on that data in order to present it in its raw form straight off the hardware in an easier to read format. Each child file is written on its own native timeseries in order to represent the caidence the data was actually collected on. This means that the MCSPower data (to pick an example) is interwoven data from all channels, and that in order to look at just WVOffline power (to pick another example) you have to do some translation using the provided `ChannelAssignment` variable to pick out just the power of the WVOffline that you might be looking for. This requires some processing on the users part, but if raw hardware returns or time resolution and averaging etc... are what you are looking for then that is a nessecity. If such details are not desired then the simpler merged data product can be examined instead.

### 3.1.2 Main CFRadial Merged Data Product

The function `MakeMergedFiles.py` reads in the raw NetCDF files created by `MakeChildFiles.py` and writes out merged CFRadial files. In these files all data products are merged onto one common timegrid which is defined by photon counting data. When photon counting data is present merged files will be created with start and end times that correspond to the start and end times of the photon counting data. For gaps in the photon counting data, merged files are made which function on a default 1/2 Hz timegrid. The presence or non-presence of photon counting data defines the creation of files and the creation of the master timestamps that all data products are put on.

Most data products are taken less frequently than photon counting data, so to push them onto the same time grid we perform an interpolation. The Weather Station (WS) as an example is taken at 1/10Hz. To do this we need to ensure that we have WS data from before the first photon counting data in a file which nessecitates investigation into the WS file before the times that are seemingly nessiary, as well as the WS file after. We load these products from the previous and next files in order to avoid extrapolation while filling the first and last few entries in data files.

Power monitoring is taken at a faster caidence (10Hz) than photon counting data. To better reflect the power monitoring data onto the common timegrid an average of the power data is considered. All power returns that came in between the nth data return and the n+1 data return are considered, an average is calculated, and that average is assigned to the n+1 data return.

Once these interpolations are completed we apply CFRadial formatting to the file, ensuring all relevent dimentionions and variables exist, and adding any additional descriptions to objects.

The merged files are designed to be made under varied conditions, that is if one or many subsystems fail the rest of the available data will be compiled into a merged file. This means that merged files can be created without data from the weather station, or considerably more dramatically, without photon counting data.

### 3.1.3 Data Backups

SyncBackup.py is the last script called by NetCDFScript.py. As a one line script it seems unassuming but it is doing something far trickier than might first appear. The one line is telling the computer to RSync our data from the Data directory embedded within the codebase to a backup location specified in the NetCDF writers config file. The location of that backup is intended to be an external hard drive. RSync is a linux utility, and in order to be able to call it from our windows environment at all the CWRsync utility must be installed, and the Data directory embedded within the codebase needs to be set up as an RSync server with that utility. On top of the fact that we are forcing a linux utility to work on windows RSync is not capable of sending information from a remote server (which CWRsync is doing in order to find the data at all) to another remoter server (which the external hard drive qualifies as according to windows). In order to get around this we need to change directories to the external hard drive, perform the RSync from there, and change directories back. This is handled within the NetCDFScript.py script.

Note: As of the time of writing this documentation DIAL2's computer (cuttlefish) was our development machine and as such does not conform to the standards that were outlined in the instalation procedures stored in the WVDIAL shared google drive. It is reccomended that cuttlefish be presented to IT for wiping and reinstallation of windows in order to conform to the other DIAL MFF computers. Until that is done DIAL2's NetCDFScript.py will need to differ from the git repo on line 111 to reflect the fact that cuttlefish's user has a different name.

### 3.1.4 Eldora

For the Relampago deployment data is also collected on Eldora. A series of cronjobs are set up to RSync the merged files as well as Error and Warning files from the DIAL unit to Eldora for processing and display on the field catalog. As of the time of writing this document the processing of data for the field catalog has been completed, but there are still problems with the RSync pulling data from the field causing the RSync to spontaneously fail, and the alert scripts are not yet set up to look for files that do not start exactly on the hour in case the unit is restarted for any reason. The alert scripts are also not set up to look for the presence of Error or Warning files which are being transfered from the DIAL unit to alert of problems. The mere presence of growing data files is not sufficient to alert of failures in operations. Because the merged files are set up to make as many data products available as possible even in the face of sub-system failure, if a sub-system fails the current alert scripts on eldora are insuficiant to alert the responsible monitor for field deployment.

# 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	
	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	

#### 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 accessible 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