AVID VLF DAQ Software Documentation

Bennett Fragomeni

This file is intended to give information about the AVID VLF software and to understand the structure of the program. Included will be what each module does and how they interact with each other. The software is ultimately compiled into a single executable for easy running on systems.

Overview

The AVID VLF DAQ software has a couple of main parts that facilitate the collection of data. They are as follows:

I. **Data Acquisition**
   A. DAQ Module - Actually collects the data
   B. GPS Module - Makes sure the program keeps correct time

II. **Data Processing / PostProcessors**
   A. MatFileWriter - Responsible for creating continuous files
   B. Narrowband - Responsible for creating narrowband files
   C. Spectrogram - Responsible for creating spectrogram files

III. **Control Flow**
   A. Main - Start of program; Initializes Logger, Engine
   B. Engine - Main control loop
      1. Makes sure program is on time
      2. Collects data from DAQ
      3. Sends data off to processing
   C. Tasks - file handling background tasks
      1. Cleanup, Copyfiles, Savetail

IV. **Logger** - Puts messages from the program into a text file
   A. Messages can be Status, Error, Warning etc.

AVID VLF DAQ Software Documentation

Bennett Fragomeni

Module Table of Contents

Unless specified, all files are python modules ending in .py
Directories are in bold

AVID VLF DAQ Software Documentation

Bennett Fragomeni

Module Breakdown

In this part of the file we will go through every module and file that is included in the AVID VLF DAQ Software. We will learn their functions, what the module does, and how they fit together with other modules in the software. See table of contents for specific modules. Modules are in blue, classes are in green, functions are in **bold**. Only Important functions will be discussed to keep this document easy to understand.

Main

Start of the whole program, here the settings are generated, important modules are initialized and the command line interpreter is set up, and finally the program is started by starting the Engine and TaskManager.

CLI

Command Line Interpreter; Control interface of the program; Takes in commands
Start, Stop, Quit
CLI is automatically started in **__main__**
The only command used regularly is q or Q to quit the program

__main__

Sets up Command Line Interpreter
Builds settings from VLF_settings.txt using settings_generator
Checks settings to make sure they exist
Gives high process priority to the program
Initializes and starts main_logger (DAQLogger)
Initializes Engine
Initializes TaskManager
Starts Engine and TaskManager
Loops Command line Interpreter to check for commands

Engine

This is the main processing loop for VLF Receivers, module objects are in **red**

EngineFlags - uses Event() flags for modules to indicate their state to the engine

Engine

** Constructor / Destructors **
__init__ - sets up loggers and flags for each submodule , calls **GenerateModules.**
GenerateModules - calls **GenerateDAQ**, **GenerateGPS**, **GeneratePPT**, **GenereateScheduler.**
GenerateDAQ - creates **daq** object based on settings from DaqSettings.xml.
GenerateGPS - creates **gps** object based on settings from DaqSettings.xml.
GeneratePPT - creates **ppt** object based on settings from DaqSettings.xml.
GenereateScheduler - creates **scheduler** object based on settings from DaqSettings.xml.
** Control Methods **
Start - creates new thread targeted at **MainLoop**, creates new thread targeted at **WatchDog.**
Stop - calls **daq.Quit**, **gps.Stop**, **ppt.Stop**, terminates MainLoop thread.
Quit - calls **Stop**, terminates WatchDog thread.
GetStatus - prints status of engine to console.
SignalEngineRestart - creates a new thread targeted at **EngineRestart.**
EngineRestart - calls **Stop**, resets flags, calls **GenerateModules**,
creates a new thread targeted at **MainLoop.**
SignalStop - creates new thread targeted at **SoftStop.**
SoftStop - calls **Stop**, joins WatchDog thread. (only way to start again is to rerun program).
** Engine Methods **
VerifyGPSLock - make sure the gps has a lock (sees 5 or more satellites).
GetGPSData - returns **gps.Get**, calls **CheckTiming**, calls **gps.SetInternalTime.**
GetDAQData - returns **daq.Get.**
SyncDAQwithGPS - will keep trying to restart the DAQ so that the DAQ and GPS are in sync.
CheckTiming - checks to make sure that the GPS and Engine is sync.
MainLoop - calls **gps.Start**, calls **VerifyGPSLock**, waits for next acquisition period, calls
SyncDAQwithGPS, then starts the main acquisition loop:

1. call GetGPSdata
2. call GetDAQdata
3. Send data off to processing in the PostProcessorTree

WatchDog - checks engine every 120s to make sure its running, the second time it is not running
it will **SignalEngineRestart.**

Note: There are error checks at almost every step; when an error occurs **SingnalEngineRestart** is called, if it is an unrecoverable error **SignalStop** is called.

AVID VLF DAQ Software Documentation

Bennett Fragomeni

Schedule

Used by the Engine and Post Processors to determine when to begin / end different tasks
LifetimeLog can keep track of activity (not currently in use)

LifetimeLog

__init__ - sets filename
Write - writes string to file
LogEnd - writes end timestamp to file
Log - writes timestamp to file every hour / new day

Schedule

__init__ - gets settings from xml file
SecondsLeftInAcquisition - return seconds left in current acquisition period

---

TaskManager

Controls all of the Tasks: Cleanup, CopyFiles, SaveTail
Messages are in **orange**, Task functions are in **gray**

TaskManager

__init__ - set up queues, give **TM_Process** its own process
Start - puts START to queue
Stop - puts STOP to queue
Quit - puts QUIT to queue
TM_Process - creates **_TaskManager** object, listens for messages and calls **Start** or **Stop** of **_TaskManager** object

_TaskManager

__init__ - calls **GenerateModules**
GenerateModules - constructs each Task as an object from the DaqSettings xml
Start - creates a thread targeted at **MainLoop**
Stop - calls **Stop** of each Task and kills **MainLoop** thread
Quit - calls **Stop**
SignalRestart - creates a thread targeted at **Restart,** terminates itself when done
Restart - terminates **MainLoop** thread, **Stops** Tasks, calls **GenerateModules**, relaunches **MainLoop** thread
MainLoop - **Start** Tasks, wait for stop, **Stop** Tasks

SettingsGenerator

Generates DaqSettings.xml based on VLF_settings.txt

_xml

add_entry - add an entry to the xml document string

MakeSettingsFile

Goes through VLF_settings.txt and turns entries into an xml document using **add_entry**

CalibrateVLF

File used to calibrate the VLF antenna, outputs graphs as PDFs and a CalibrationVariables.mat file for actually calibrating the system.

square_antenna_params - Calculate values for square antenna:
Resistance, Inductance, Magnetic field sensitivity, Electric field sensitivity, Cutoff frequency, Area
isosceles_right_triangle_antenna_params - Calculate values for triangular antenna:
Resistance, Inductance, Magnetic field sensitivity, Electric field sensitivity, Cutoff frequency, Area
plot_calibration_number - Plots Calibration Number (most important for antenna calibration)
plot_frequency_response - Plots Frequency Response
plot_response_ratio - Plots Response Ratio
plot_noise_floor - Plots Noise Floor
calibrate_2ch - Creates calibration variables based on params and EW NS .mat files

---

NIDAQmx

Parent Class of the DAQ process, takes in commands from the Engine and sends messages to DaqCard

NIDAQMx

__init__ - sets up queues, sets up process with high priority for **DAQ_Process**
Start - empties queues, sends START message to **DAQ**
Stop - sends STOP message to **DAQ**
Quit - sends QUIT message to **DAQ**, closes queues, terminates daq process
Restart - sends RESTART message to **DAQ**
RestartAnalog - sends RESTART_A message to **DAQ**
RestartLineReciever - sends RESTART_LR message to **DAQ**
Get - retrieve the oldest data block from queue
Flush - empty the queue
DAQ_Process - sets up child class **DAQ** from DaqCard, calls **DAQ**.**MainLoop**

DaqCard

Child Class of the DAQ process, takes in messages and actually communicates with the DAQ driver
Messages are in **orange**, driver calls are in **purple**

DAQ

__init__ - initialize the DAQ class by reading the xml DaqSettings, call:
_DeviceReset, _DeviceSelfTest, _DeviceDigitalOutputSetup, _LineRecieverRestart, _DeviceSetup
MainLoop - listen for messages coming from NIDAQmx, and call functions
START → **DAQStart** STOP → **DAQStop** RESTART → **DAQRestart**('FULL')
RESTART_LR → **DAQRestart**('LR') RESTART_A → **DAQRestart**('AN')
DAQStart - calls **_DeviceAnalogStart**
DAQStop - calls **_DeviceAnalogStop**
DAQQuit - clears buffers, calls **_DeviceAnalogStop, _DeviceAnalogClear**
DAQRestart - runs specified restart, Engine starts task
FULL - calls **_DeviceAnalogStop, _DeviceAnalogClear, _LineRecieverRestart, _DeviceAnalogSetup**
LR - calls **_DeviceAnalogStop, _LineRecieverRestart**
AN - calls **_DeviceAnalogStop**
sampleClockCallback - check if counter is stable, calls DAQmxStopTask, DAQmxClearTask
_StopAndSignalRestart - signal the Engine that DAQ needs to restart by setting error flag
_LineRecieverRestart - call **_LineRecieverOff, _LineRecieverOn**
_LineRecieverOff - turn off LR by calling DAQmxWriteDigitalScalarU32
_LineRevieverOn - turn on LR by calling DAQmxWriteDigitalScalarU32
_DeviceSelfTest - test DAQ by calling DAQmxSelfTestDevice
_DeviceReset - reset DAQ by calling DAQmxResetDevice
_DeviceAnalogSetup - setup DAQ by following steps
create task by calling DAQmxCreateTask
create voltage channel by calling DAQmxCreateAIVoltageChan
set config: DAQmxCfgInputBuffer, DAQmxCfgSampClkTiming, DAQmxCfgDigEdgeStartTrig
setup callback, call **EveryNCallback** after reading 1s of data:
DAQmxRegisterEveryNSamplesEvent
_DeviceAnalogStart - calls DAQmxStartTask
_DeviceAnalogStop - calls DAQmxStopTask
_DeviceAnalogClear - calls DAQmxClearTask
_DeviceDigitalOutputSetup - calls DAQmxCreateTask, DAQmxCreateDOChan, DAQmxStartTask
_DeviceSetup - calls **_DeviceSelfTest**, **_DeviceAnalogSetup**
_DeviceErrorCheck - called with every driver call to check for errors
EveryNCallback - reads the data from the DAQ and puts it into a queue
read DAQ DAQmxReadBinaryI16, put timestamp and data into queue
if anything goes wrong call **_StopAndSignalRestart**

DAQExceptions

Contains all exceptions for the DAQ card, implemented under class DAQError.

DAQError

DAQNoDataError, DAQInitError, DAQStartError, DAQStopError, DAQSampleError, DAQInternalError, DAQNotFoundError

---

VirtualCard

A DAQ card that is completely software based. This includes basic functions and methods, as well as dummy data collection. Used to test the software without being hooked up to the DAQ.

VirtualCard

__init__ - initialize the class
InitBuffers - part of __init__
Start - create a new thread with the target of MainLoop
Quit - stop the thread, stop OnePPS, Flush the queue
Get - return the oldest element in the queue
MainLoop - create 1s of dummy data and put it to the queue every OnePPS pulse

---

GPSExeptions

ClockError

ClockSkipError, ClockQueueFull, ClockMsgQueueFull, ClockNoDataError, ClockInternalError

---

MotorolaClock

Interface for communicating with Synergy GPS using Motorola byte commands.
There are more messages than the @@HA timestamp message but they are not nearly as important.

MotorolaClock

__init__ - Initializes SerialClock, sets up all of the commands / messages
MainLoop - Tests all messages, then sends message (@@HA) to GPS to emit timestamps every 1s, calls _WaitForMessage
_WaitForMessage - Waits for a correct response from the GPS
_MatchMessageHeader - Matches messages received from GPS to a message in __init__, processes the message based on the processing function.

_DecodeTimestamp - One of the processing functions, decode the message from GPS into a datetime object, latitude, longitude, altitude, and number of GPS satellites connected to.

SerialClock

Interface for communicating with the serial port of the computer in order to send messages to GPS and receive GPS messages through USB.

SerialClock

__init__ - Initializes Clock, gets settings from DaqSettings.xml
Start - Opens the serial port, start **_SerialReaderThread** thread and **MotorolaClock MainLoop** thread
Stop - Sends Stop Command to GPS, joins both threads, flushes GPS queues, closes serial port.
Restart - Calls **Stop**, resets values, calls **Start**
WriteToSerial - writes a byte message to the serial port
ReadFromSerial - reads one byte from serial port
_SerialReaderThread - calls ReadFromSerial, adds that byte to the buffer, calls **MotorolaClock _MatchMessageHeader**

Clock

Handles mostly the queue that the GPS uses to store its timestamps.

Clock

__init__ - sets up the queue
HaveLock - makes sure the GPS can see at least 5 satellites
SetInternalTime - sets internal time to what the GPS receives, calls **SetInternalLocation**
SetInternalLocation - sets internal location to what the GPS receives
Get - return the top element of the GPS queue
GetFast - increment internal time by 1s, call **Get**, call **HaveLock**, return gpsData = [timestamp, [time, [lat, lon, alt], [quality]]]
Flush - clear all elements from queue
_PostToQueue - makes sure timestamp is correct, put timestamp on queue
_log_qsize - print out the size of the queue to VLFDAQ.log

VirtualClock

Used to test the system with no GPS present, puts the current time as a timestamp in the GPS queue every 1s.

LogScrapper

Scrapes VLFDAQ.log to determine system performance from a start time to an end time.
Counts errors and restarts, you can add more functionality if you like.
StartTime - timestamp where you want to start scraping

PostProcessorTree

Controls the post processors of the program:
IndexWriter, MatFileWriter, Narrowband, NarrowbandC, Specgram
Constructs post processors, receives data from Engine and passes it to post processors.
Post processor functions are in gray.

PostProcessorTree

__init__ - uses **ConstructSequence** to construct post processors
Stop - calls **Stop** function for each post processor
Process - sends DAQ and GPS data to **ProcessSequence**
ConstructSequence - returns a list of constructed post processor objects
ProcessSequence - copies data and sends it to **Process** function of each post processor

---

IndexWriter

Creates an index xml file that contains information on each data file that has been written.
This includes Filename, start time, duration, and sample rate.

IndexWriter

Process - checks if a new file is needed, calls **AppendEntry**
AppendEntry - opens file, writes Filename, start time, duration, sample rate to file, closes file

---

MatFileWriter

Creates MATLAB mat files and appends data to them. These files are stored in the continuous folder.

MatFileWriter

__init__ - gets settings from DaqSettings.xml and sets up the directory “Continuous”, replaces .mau files with .mat files.
Process - checks if a new file is needed, calls **rename_mau**,
calls **WriteHeader**, then **AppendData.**
WriteHeader - writes the station settings to the file.

AppendData - writes the 1s of data to the file.
rename_mau - renames .mau files to .mat

Narrowband

Demodulates Narrowband Channels to track amplitude and phase

NarrowbandC

Demodulates Narrowband Channels to track amplitude and phase, wraps c++ code

---

PPTExceptions

PPTError, MatFileWriterError, MFWDiskFullError

Specgram

Creates .jpg images of the intensity versus wavelength of the VLF spectrum

---

Cleanup

Cleans up files from Continuous, Narrowband, Spectrogram that are old. (Not on external drives)

Cleanup

__init__ - gets settings from DaqSettings.xml
DoTask - checks files in directories specified, checks if they are old, removes old files
FileLifetimeFilter - True if file is old, False otherwise

CopyFiles

Copies files from the computer onto external hard drives.

CopyFiles

__init__ - gets settings from DaqSettings.xml
DoTask - makes sure disk has 5gb free space using **checkDisk**, calls **CopyFiles**
CopyFiles - copies files from computer to external hard drive, deletes files on computer
checkDisk - returns amount of free space on hard drive

SaveTail

Saves the last n characters of VLFDAQ.log to a new file.

SaveTail

__init__ - gets settings from DaqSettings.xml
DoTask - copies characters from VLFDAQ.log to new log file

Task

Helper class for all of the tasks: Cleanup, CopyFiles, SaveTail; has functions that all tasks use

Task

__init__ - sets start / end times for tasks
Start - creates new thread that targets **MainLoop**
Stop - kills **MainLoop** thread
MainLoop - checks **TimeToDoTask**, if true calls task.**DoTask**
decodeTimeStamp - takes time string and returns datetime.time object
TimeToDoTask - returns True if now is between start / end times, False otherwise

---

MemoryTest

Uses tracemalloc module to track memory usage of the program for testing.

ProfileSorter

Used to look at how different parts of the program are performing for testing.

SerialChecker

Checks the serial connection with the GPS.

ConsoleHandler

Used by DAQLogger to write messages to the console.

ConsoleHandler

__init__ - sets stream as stdout
flush - empties stdout
cformat - default formatting
emit - write string to stdout with formatting

ColoredFormatter

cformat - colors text in string for better readability

EmailHandler

Used by DAQLogger to write email messages.

EmailHandler

__init__ - initialize email handler
getSubject - return subject string
emit - create email, login to email server, send email

LogLevels

Maps log levels to values.

DAQConfig

Holds function to read parameters from DaqSetting.xml

DAQConfig

__init__ - set setting file
GetSubTree - return subtree corresponding with tag
GetFirstSubTree - return 0th element of **GetSubTree**
GetSubTreeStrElm - return **GetFirstSubTree** as string
GetElmVal - return element corresponding to tag, return default if no match, return 0 otherwise
GetIntElmVal - cast **GetElmVal** to int
GetStrElmVal - cast **GetElmVal** to string
GetDblElmVal - cast **GetElmVal** to float

DAQLogger

Handles all of the logging to the output file VLFDAQ.log

DAQLogger

__init__ - get settings from DaqSetting.xml, print settings to console, setup queues
start - start new process targeted at **_log_processing**
stop - join queues and terminate process
_log - create log entry and put it to queue

DAQLogClient

Handles putting a specific log level into log_queue,
Log levels: debug, status, timestamping, info, warning, error, critical, exception

_log_listener - push messages from log_queue to raq_queue continuously
_log_processing - Setup handlers: Logfile, Console, Email, HTTP Post,
pull items from raw_queue and push to logger

---

loadMATdata

Loads Matlab data files for use in Spectrogram.

OnePPSSignal

generates a pulse every 1s, used for VirtualCard and VirtualClock

---

Restart_counter

Updates restart.count with the number of current restarts, called from Engine

update - update the value in restart.count
check - check if value in restart.count is correct, return True or False
peek - return value from restart.count
current_value - return lastValue

DaqSettings.xml

Settings file, constructed from default_VLF_settings.txt using settings_generator

Filter_taps.txt

used by NarrowbandC

Main.spec

File used by PyInstaller to construct the packaged executable for deployment. Instructions for running can be found in the file. More information on spec files can be found HERE

---

nb.conf

used by Narrowband and NarrowbandC

Restart.count

File for holding the number of restarts that the system has undergone, updated by restart_counter

---

VLFDAQ.log

Log file where messages are stored, written to by DAQLogger

VLF_settings.txt

Settings used by settings_generator to make DaqSettings.xml
