Software for the Mote Marine Research Optical Phytoplankton Detector

Jonathan Turner

Mote Marine Laboratory Technical Report #1988

September 21, 2016

Abstract

The Optical Phytoplankton Detector (OPD) is an in situ marine spectrophotometer first developed by the Mote Marine Research Laboratory starting around 2005 [2, 3]. It performs automated sampling and analysis of seawater and has been used to detect the presence of the algae responsible for causing *red tide*. This report describes the software that controls the OPD, primarily as an aid to software developers who may be called upon to maintain the software and expand its capabilities.[1]

[1]This report has been formatted to be read conveniently on an ipad or similar tablet. Paper copies are best printed with two pages per sheet in landscape mode.

1 Introduction

The Optical Phytoplankton Detector (OPD) is an in situ marine spectrophotometer first developed by the Mote Marine Research Laboratory Phytoplankton Ecology (PE) Group, starting around 2005 [2, 3]. The Ocean Technology Group was later spun off of the PE group and now maintains the OPD. The instrument performs automated sampling and analysis of seawater and was originally developed for deployment on *Autonomous Underwater Vehicles* (AUVs) to detect the presence of the algae responsible for causing *red tide*, Karenia brevis. More recently, it has been re-configured as a low-pressure instrument for fixed-location monitoring. The internal Ocean Optics spectrophotometer covers the ultraviolet and visible light spectrum from about 180–880 nm. Unlike similar instruments, the OPD can collect filtered seawater samples, in order to analyze colored dissolved organic matter (CDOM), or unfiltered samples, to analyze raw water (dissolved and particulates). It can be deployed for periods of several weeks, allowing detailed, cost-effective monitoring of the ocean environment. This report details software version Cauv-5.0.6, the most recent version at the time this report was written (September 2016).

This report describes the software that controls the OPD, primarily as an aid to developers who may be called upon to maintain the software and expand its capabilities. In order to understand the software, one must first understand the instrument itself, the mechanisms used to collect seawater samples and the algorithms used to analyze their spectra. Section 2 of this report describes the structure and operation of the OPD. Section 3 discusses the onboard data analysis performed by the OPD software. Section 4 describes the high-pressure version of the hardware. Section 5 describes how to compile and run the software on an OPD in a lab environment. Section 6 describes the overall organization of the software and explains key aspects of the control flow. Detailed code documentation appears as a series of appendices, one for each source code file.

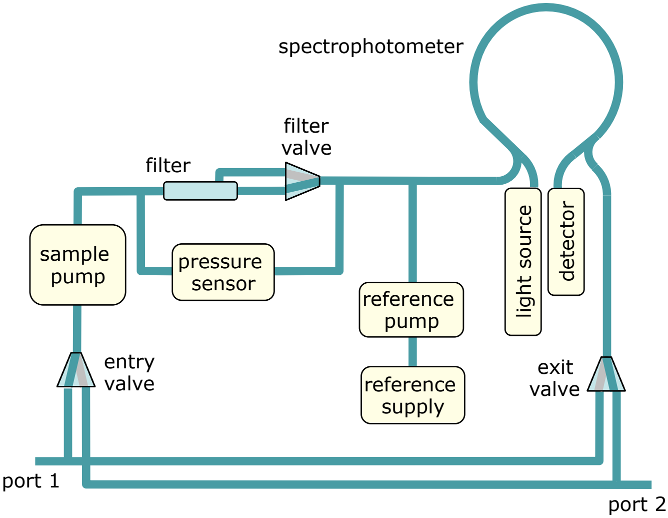

Figure 1 Structure of the OPD

2 Structure and Operation of the OPD

The organization of the OPD's main components is shown in Figure 1. The centerpiece is the spectrophotometer, which includes a *Liquid Waveguide Capillary Cell* (LWCC) through which sea water samples can be pumped. Once a sample has been pumped into the waveguide, the light source is turned on for a specified time period and the amount of light passing through the sample within each of 2048 optical wavelength bands is measured by the detector. This data can be transferred to the OPD's control processor (not shown) for analysis.

There are three types of samples that are generally collected and measured by the OPD. First, there is a *reference sample*, which is supplied from an onboard reservoir. A measured spectrum for a reference sample provides a basis of comparison for spectra obtained from seawater. This makes the OPD relatively insensitive to variations in the brightness of the light source and the sensitivity of the detector. The other two types of samples are *filtered seawater* samples and *unfiltered seawater* samples. These are usually referred to as CDOM and discriminant samples, respectively, where CDOM stands for *Colored Dissolved Organic Material* and the term discriminant alludes to the fact that the unfiltered samples are used to discriminate amongst a variety of different species of algae.

Seawater samples enter the OPD through one of two external ports and exit through the other. The OPD uses a pair of valves, called the *entry and exit valves*, to select which port is used for seawater entry and which for exit. These valves are operated in concert, so that one always connects to port 1, while the other connects to port 2. Alternating the positions of these valves reverses the flow of seawater through the two ports, preventing the buildup of material that might otherwise block the wire mesh screens that keep large particulates from entering (and potentially clogging) the OPD's internal passages. Each of the two valves is really a pair of *pinch valves*, but each pair is always used to select one of the two available ports.

Samples are sent through the OPD by the *sample pump* to the *filter* component. The filter has two "outputs", a *filtered* output and a *bypass* output. The valve downstream of the filter is normally used to select one of these two outputs, although it is also possible to close both paths. In the diagram, the filtered output is the one that comes out the top of the filter, while the bypass output is the one that goes straight through the filter component.

The sample pump is reversible, so it can pump seawater samples backwards through the filter path. This can be used to flush out small particulates that may build up in the filter during the collection of a filtered sample. The two pressure sensor components measure the water pressure at two points in the sampling path. This is primarily used to measure the difference in pressure across the filter when collecting a filtered sample. This pressure must be kept below a certain threshold (30 psi) to prevent damage to the filter. The pressure sensor can also be used to estimate the depth of the OPD in the ocean.

When the OPD is making automated measurements, it periodically performs what is referred to as a *sample cycle*. Typically, one out of eight sample cycles starts with a reference fluid measurement. This is generally referred to as the *CDOM reference phase*. The reference pump runs for a specified amount of time to fill the spectrophotometer's waveguide with reference fluid (and flush out any remaining sea water from a previous measurement). Before recording the spectrum, the OPD performs a calibration process to adjust the amount of time that the spectrophotometer's light source is turned on during data collection (this time duration is called the *integration time*). This calibration process seeks to maximize the spectrophotometer's sensitivity. Once the integration time has been adjusted, the reference fluid spectrum is acquired. The OPD also acquires a "*dark spectrum*" that measures the output of the detector across all wavelengths when a shutter is closed, preventing light from reaching the detector. The dark spectrum essentially measures the base noise level of the detector. It is subtracted from the CDOM reference spectrum during data analysis.

Every sample cycle also includes a filtered seawater measurement (called the *CDOM phase*) and an unfiltered seawater measurement (the *discriminant phase*). The CDOM phase must be done with some care to prevent over-pressure at the filter, but once a sufficient volume of filtered sea water has been pumped through the filter and into the spectrophotometer's light guide, the CDOM spectrum is acquired, along with an associated dark spectrum.

Each sample cycle concludes with a *discriminant phase* during which spectra from one or more samples of unfiltered seawater are acquired. The first such sample contains a concentrated "plug" of particles, which collected in the filter during the CDOM phase and are washed through the bypass path during the discriminant phase. This concentration of the particulates improves the instrument's sensitivity. Subsequent samples acquired during the discriminant phase more closely reflect actual seawater conditions. Because the samples collected during this phase are pumped through the bypass path, there is no significant pressure difference across the filter.

The OPD control software runs on an embedded processor system produced by *Persistor Instruments*. The processor is a Freescale M68332. This is a

16 bit processor with a maximum clock frequency of 16 MHz. By modern standards, it's relatively slow (2 MIPS), but it provides a reasonable platform for embedded applications with modest processing requirements. The embedded software system also includes mechanisms for communicating with other electronic components, including a multi-channel UART and SPI bus.

The provided operating system includes some very basic facilities for managing files and running programs, plus software libraries supporting IO, and real-time clock functions, among other things. Software developers are advised to familiarize themselves with the documentation available on the Persistor web site [4, 5, 6, 7].

3 Data Analysis

The OPD spectrophotometer produces a *transmission spectrum* which measures the amount of light passing through the sample within each of 2048 optical wavelength bands. An example of such a spectrum is shown in Figure 2. The spectrum values fall in the range of 0-65,535, but are not calibrated to any specific units. Observe that the spectrum includes high-frequency variations that mostly reflect artifacts of the spectrophotometer, rather than properties of the sample being analyzed. Consequently, the first step in the analysis process is to produce a smoothed version of the transmission spectrum. This is done by replacing the value at each frequency f with a weighted average of the values in a window of width ± 23 around f. The weighting function is a Gaussian curve with a standard deviation of 12.

After the spectrum is smoothed, it is also converted to a normalized form in which the data values are given at integer wavelengths in the range 350-799 nm (the raw spectrum has a spacing of approximately .3 nm, but this spacing is not uniform across the entire range). These normalized spectra are easier to compare and lend themselves to faster analysis calculations. For each sample spectrum, a dark spectrum is acquired and subtracted from the

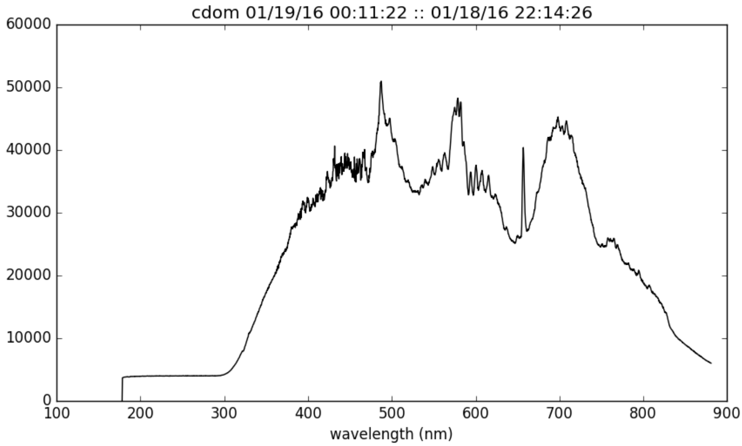

Figure 2 Raw transmission spectrum

associated sample spectrum (after both are smoothed and normalized). An example of a resulting "cooked" spectrum is shown in Figure 3.

During a CDOM phase, a filtered seawater sample is pumped through the spectrophotometer and used to produce a CDOM sample spectrum. The cooked CDOM spectrum is combined with the most recent cooked CDOM reference spectrum to produce a *CDOM absorption spectrum*. Each value in the CDOM absorption spectrum is calculated by taking the natural logarithm of the ratio of the reference transmission spectrum to the sample transmission spectrum. The resulting values are then divided by the length of the spectrophotometer's waveguide. It has been established that the CDOM absorption spectrum can be well-approximated by an exponential curve, and it is common to characterize a CDOM spectrum by its value at 440 nm and the exponential coefficient of the approximating curve. An example of such an absorption spectrum and its exponential approximation are shown in Figure 4.

During a discriminant phase, an unfiltered seawater sample is used to produce a discriminant sample spectrum. The cooked discriminant spectrum is

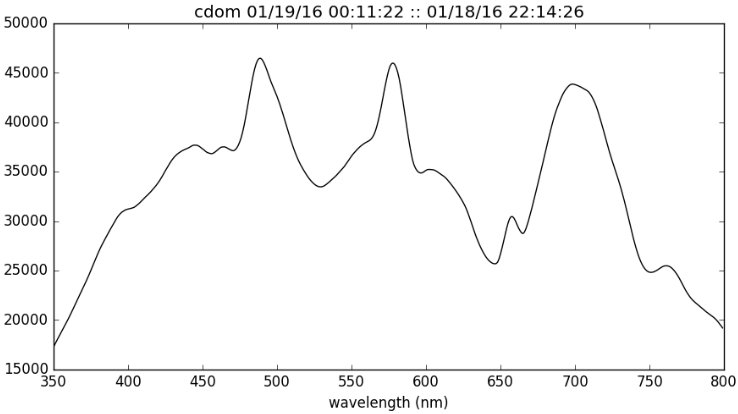

Figure 3 Cooked transmission spectrum

combined with the most recent CDOM spectrum to produce an *absorbance spectrum*, which is the base-10 logarithm of the ratio of the CDOM spectrum to the discriminant spectrum. The discriminant spectrum can be compared to one or more spectra from a library of exemplar spectra, to determine which phytoplankton species are most likely present in the given seawater sample. The comparison procedure involves taking the fourth deriviative of the discriminant absorbance spectrum and each exemplar spectrum, and then computing a correlation coefficient between each pair of fourth-derivative spectra. The resulting values are in the range $[-1, +1]$ with values larger than 0.6 signalling the likely presence of a given exemplar species in the seawater sample.

4 High Pressure OPD

Up to this point, we have described only the low pressure version of the OPD, which is the version that is most commonly used at the present time. There is actually a high pressure version as well, and the OPD control software

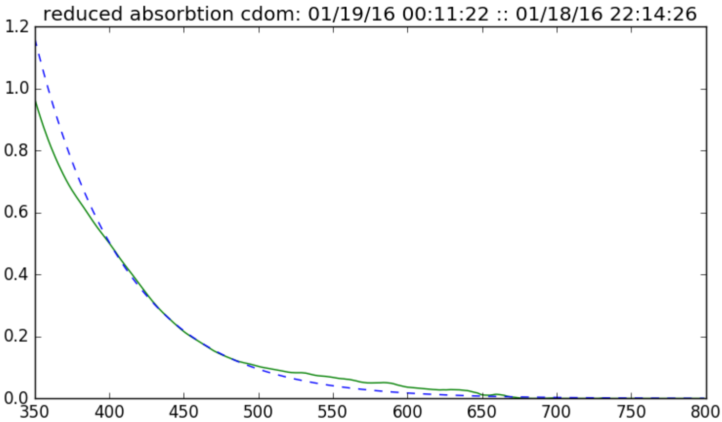

Figure 4 CDOM absorption spectrum

includes code to handle both versions, so it's important for those maintaining the software to understand how both versions work. The good news is that what has already been discussed for the low pressure version also applies to the high pressure version. The bad news is that the hardware for the high pressure version is somewhat more complicated than that for the low pressure version. That also means that the code required to control the high pressure version is more complicated.

The main difference between the low pressure and high pressure versions is the nature of the sampling pump and the valves used to control the flow of fluid to and from the sampling pump. First a word about the pumps used in the low pressure units. These are a type of pump called *peristaltic pumps*, which move fluid through a flexible tube by pinching the tube and advancing the "pinch point" to move the fluid forward (there is a very instructive article on Wikipedia that describes the operation of peristaltic pumps). Such pumps are quite simple and work well in low-pressure contexts, but they cannot handle the high pressures that occur in deep water. For this reason, the high pressure version of the OPD uses a different type of pump called a *syringe pump*. As the name suggests, a syringe pump uses one or more syringes, which draw fluid into a *barrel* by withdrawing a *plunger*, then force fluid out by moving the plunger back into the barrel. Valves are used to control where the fluid is pulled from and pushed to in the different phases of the pump's operation.

The high pressure OPD's sample pump has a total of six syringes. Four of these are used to pump sea water to and from the external ports and through the OPD's internal plumbing. The other two are used as part of the drive mechanism for the first four. Figure 5 shows the organization of the high pressure OPD, including the syringe pump and the system of valves (called Valco valves) used to control the flow of fluid to and from the syringes. The plungers for the six syringes are all connected, so they move together, either to the left or to the right. So, when the three syringes on the left are drawing fluid in, the three on the right are pushing fluid out. There are limit switches that detect when the plungers have reached the end of their range of travel. The driver pump at the bottom of the figure, moves water back and forth between the two large syringes. This drives the plungers for the other four.

The Valco valves have two positions. In the first position (which is highlighted in the figure), the valves connect ports 1 and 2, ports 3 and 4, ports 5 and 6, and ports 7 and 8. In the second position, they connects ports 1 and 8, 2 and 3, 4 and 5, and 6 and 7. When the valves are in the first position, the content of the syringe at the top left is sent towards the filter, while the fluid from the spectrophotometer is returned to the middle syringe on the right. At the same time, sea water is drawn into the top right syringe from port 2 and sent out from the middle left syringe to port 1. When the valves are in the second position, the content of the top right syringe is sent towards the filter, while the fluid from the spectrophotometer is returned to the middle right syringe. At the same time, sea water is drawn into the top left syringe from port 1 and sent out from the middle right syringe to port 2. Observe that the high pressure OPD has a separate waste port that is used when pumping reference fluid through the spectrophotometer.

Now, the basic operation of the OPD is the same, whether it is a low pressure unit or a high pressure unit. The differences arise from the more complicated procedures that must be followed when using the syringe pump.

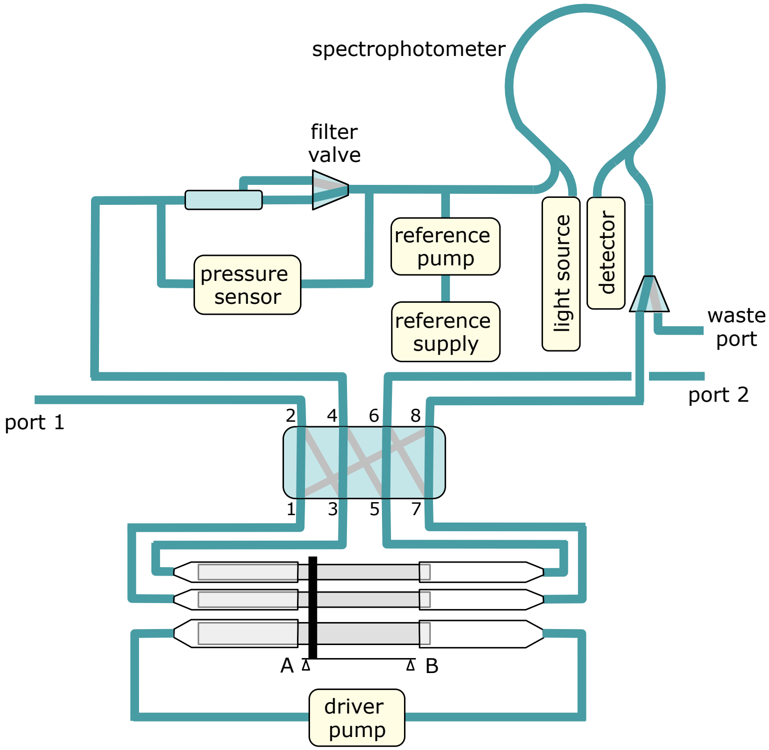

Figure 5 High pressure OPD

5 Compiling and Running the Program

The sources for the OPD control software are maintained on BitBucket (https://bitbucket.org/MoteOceanTechnology/opdclassic). Access does require a BitBucket account and permission from the site administrator. Once access has been granted, you can clone the repository using git. In addition to the current version, the repository includes earlier versions of the software going back to version CauvV4.5_rev5, February 2012.

Persistor supplies a self-contained programming environment called Code-Warrior that can be used to edit and compile the OPD control software. It provides access to both the usual C libraries and Persistor-specific libraries that support the embedded system's various hardware components. See the API documentation for details [6]. Code Warrior must be run in a Windows environment, preferably with an older version of Windows (e.g. Windows 7 or earlier). It can be run in a virtual machine environment, such as VirtualBox or VMware. The getting started guide [4] explains how to install the software from an installation disk and the instructions are accurate and complete.

There is one pitfall to look out for when installing the software. The installed folder `C:\Program Files\Persistor\Motocross Support\CFX` contains supporting code that must be included in your CodeWarrior project (this code appears in three subfolders: Headers, Libraries, Source). The documentation describes how to do this. However, the version of the supporting code on the installation disk is not completely compatible with the OPD application. For that reason, a modified copy of the supporting code is included in the BitBucket repository for the OPD software, in the support folder. These modified source files should be used in place of the ones supplied on the installation disk. The simplest way to accomplish this is to remove the original support folders and copy the folders from the BitBucket repository in their place.

Once you have CodeWarrior installed and your project configured, you can compile the software by selecting the "Bring Up To Date" item in the Project menu. When recompiling, you may want to first select the "Remove Object Code" item, to ensure that all changes are reflected in the compiled version. To link the compiled object files, select "Make" from the Project menu. This will produce an OPD.RUN file that should appear in the bin folder within your project's main folder.

The OPD is equipped with a serial interface cable that can be connected to a terminal, or a PC running terminal emulation software. This allows a user to communicate with the OPD control program, while it is running. The program includes a command-line interpreter that responds to commands

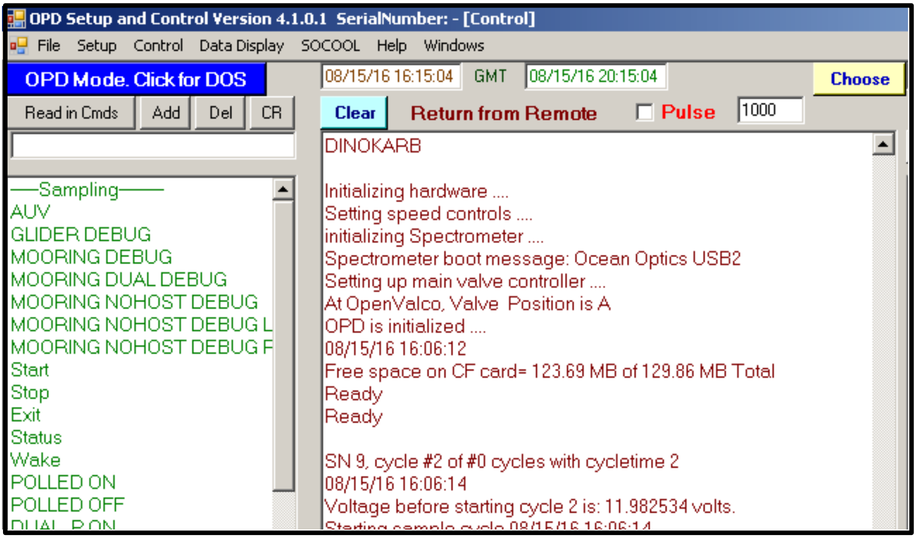

Figure 6 Console window

received on the serial input. When automated sample collection is enabled, the program sends a variety of status messages to the serial output, so that users can monitor the OPD's progress. When the OPD is run in a lab environment, a user-interface program (referred to here as the *console*) uses the serial cable to communicate with the OPD and provide convenient mechanisms for issuing commands and monitoring the OPD's status. Figure 6 shows a portion of the console window. The menu at the left lists a variety of commands that can be issued to the OPD, while the window to the right shows status messages sent by the OPD.

By convention, the executable file containing the OPD control program is called *OPD.RUN*. To run the program, this file should be copied into the top level folder of the OPD's compact flash card. This folder should also contain an *autoexec.bat* file consisting of a single line, such as

```
OPD MOORING DEBUG DUAL_P NOHOST
```

where OPD is the name of the executable file (without the .RUN suffix), and the remaining words are command-line arguments specifying various options. The *MOORING* option specifies that the OPD will be deployed at a fixed location, like a mooring. This is also the option most commonly used when running the OPD in a lab environment. The *DEBUG* option specifies that debugging messages should be displayed in the console window. The *DUAL_P* option specifies that the unit has two pressure sensors (earlier units had only a single pressure sensor). The *NOHOST* option causes the program to start automated sample collection when it starts up, rather than wait for an explicit command.

To start the program, the compact flash is inserted into the card slot on the OPD and then power is turned on. The Persistor's operating system boots up and executes the autexec.bat file, causing the OPD program to start. When it starts, it first looks for a configuration file called *config.txt* that appears in the *setup* folder. The config file contains over 30 configuration parameters that control various aspects of the OPD's operation. These are listed in Figures 7-10. Some of the configuration parameters are automatically adjusted by the software. It's generally best to not change the adjusted values from one run to the next. Many other parameters have recommended values that should be used under most circumstances. The parameters that are most commonly set by the user are *NumberCyclesToRun*, *CyclesDoneSoFar*, *CdomRefRepeatInterval*, *DiscRepeatCount* and *GPS*.

Once the configuration parameters are read in by the OPD control program, they are displayed on the console. The program then proceeds to turn on the various hardware components of the OPD and initialize them. There are two other configuration files in the setup folder. The file called *cdrfleft* contains a single value representing the number of milliliters of fluid in the CDOM reference fluid resevoir This is initialized by the user at the start of a run and is updated by the OPD, as the CDOM reference fluid is consumed. If the reservoir runs dry before the end of a sample collection run, sampling continues without the CDOM reference phase. The file called *phone.txt* contains a phone number, which can be used to make a cell phone call to a

parameter	description
NumberCyclesToRun	number of data collection cycles to run before terminating; if zero, run until forced to stop
CyclesDoneSoFar	number of data collection cycles run by program; initially 0, but updated by program to allow continuation following a premature termination
CdomRefPumpSpeed	speed at which to run CDOM reference pump, typically 200
CdomRefPumpCal	calibration constant used to calculate the amount of fluid pumped in a given amount of time at a given speed
SamplePumpSpeed	speed at which to run the sample pump when collecting unfiltered samples; typically 127
PosSensorMin	for high pressure units, specifies the end-of-range position at the low end of the syringe pump's travel range; automatically adjusted by software
PosSensorMax	for high pressure units, specifies the end-of-range position at the high end of the syringe pump's travel range; automatically adjusted by software

Figure 7 Configuration parameters

remote reporting site that records the information sent by the OPD on its serial output.

In addition to the three configuration files, the OPD is usually supplied with one or more *species files* containing discriminant absorbance spectra computed from samples containing known species of phytoplankton. These spectra are compared against the discriminant spectra obtained during a sample collection run to detect the presence of these species. These files are contained in the *species* folder on the compact flash card. The species

parameter	description
CdomStrokePos	for high pressure units specifies the position at which the syringe pump has sent a large enough volume of fluid through the spectrophotometer for a complete sample
CurrentIntegrationTime	integration time parameter used by spectrophotometer; automatically adjusted by software
WaveGuideLength	length of the spectrophotometer's waveguide; typically 0.28
DoCdomRef	flag that enables (or disables) CDOM reference phase
CdomRefFlowTime	number of seconds to run CDOM reference pump before adjusting integration time
CdomRefRepeatInterval	number of cycles between CDOM reference phases
DoCdom	flag that enables (or disables) CDOM phase
CdomVol	volume of fluid to pump during CDOM phase
DoDisc	flag that enables (or disables) discriminant phase
DiscRepeatCount	number of “extra” discriminant samples to collect during discriminant phase

Figure 8 Configuration parameters (continued)

files are in binary format. Each contains 450 double precision floating point values representing the discriminant absorption spectra of the given species.

In addition to the console messages that are output while it is running, the OPD saves results in files on the compact flash drive. There are three types of files, *log files*, *status files* and a *debug file*. These are all saved in the *results* folder on the compact flash card. The raw transmission spectra acquired by the OPD are saved in one or more log files (these have a .log filename extension). Each log file contains data collected during a four hour

parameter	description
DiscFlowTime	time to run sample pump while collecting discriminant sample
FilterSize	total width of window used by Gaussian smoothing function
FilterSigma	standard deviation parameter of Gaussian smoothing function
CorrWaveMin	first wavelength used in computation of correlation metric (similarity index)
CorrWaveMax	last wavelength used in computation of correlation metric (similarity index)
AbWaveMin	first wavelength used to fit CDOM absorption curve to exponential function
AbWaveMax	last wavelength used to fit CDOM absorption curve to exponential function
AbWaveMid	midpoint of wavelength range used to fit CDOM absorption curve

Figure 9 Configuration parameters (continued)

time period. These files are stored in a binary format consisting of character strings and data vectors. A log file consists of a series of variable length *segments*, where each segment begins with a *type character*, which is either 'C' or 'I'. The type character is followed by a 16 bit unsigned integer (with the least-significant-byte first), which specifies the length of a *vector* which comes next. If the type is 'C', this vector consists of characters. If the type is 'I', the vector consists of 16 bit unsigned integers (with the least-signifcant byte first). Here is a small sample from a log file that has been converted to a purely ASCII format.

```
DT 01/14/16 16:24:07
StartRun
CauvV4.8, Rev.3, 23 OCT 2014
```

parameter	description
MaxFilterPressure	maximum allowable pressure difference across filter
DepthCoefA	coefficient used to compute depth from pressure measurement
DepthCoefB	coefficient used to compute depth from pressure measurement
RunDepth	depth at which to perform sampling
ShutDownDepth	minimum depth
BBSerialNumber	serial number of the physical OPD
GPS	GPS coordinates of OPD location (for reporting)
GPSget	flag used to control automated setting of GPS coordinates

Figure 10 Configuration parameters (continued)

```
WavePoly: 1.781104e+02 3.825996e-01 -1.511990e-05 -1.996850e-09
Spectrophotometer Serial Number: USB2E6931
DT 01/14/16 16:24:08
...
DT 01/14/16 16:25:15
DT 01/14/16 16:23:45
StartDepth 0.0
StopDepth 0.0
ID: Dark
0 3216 3212 3235 3278 3344 3311 3326 3337 3338 ...
...
DT 01/14/16 14:53:23
DT 01/14/16 14:53:23
StartDepth 0.0
StopDepth 0.0
ID: CdomRef
    0  3213  3235  3249  3271  3326  3301  3331  3333  3372 ...
```

```
...
Absorbance value/slope: 0.1595 0.0026
DT 01/15/16 00:10:14
...
DT 01/15/16 02:13:27
Correlation Results
Species DINOKARB 0.24
```

Note that the file starts with a header, and that *date/time* strings appear at various locations throughout the file. Each spectrum is preceded by its own metadata, and other analysis results are also included.

The OPD also records summary information in one or more *status files* (these have a .stt filename extension). Each file contains one or more brief status reports that summarize the operational status of the OPD and selected analysis results. Finally, the OPD sends a variety of status and debugging messages to a *debug* file (called debug.txt).

While the OPD is running, a user can interact with it by sending text commands from the console. Using the console user-interface, these text commands are issued in response to button clicks, but to the OPD control program, the commands are simply lines of text. Figures 11 and 12 show the most commonly used commands. Each is identified by a command name enclosed in angle brackets (e.g. <command>) and some have one or more arguments.

These commands can be used even while automated sample collection is enabled. There is also a large set of additional comands that can only be used when automated sampling is disabled. These commands all have numerical codes assigned to them and are accessed via the <opcode> command. The commands identified by opcode are listed in Figures 13-17. If the OPD is in automated sample collection mode, you first suspend automated sampling using the <stop> command. The <cycle> command can be used to resume automated sampling. The <wake> command is used to wake up an OPD that is sleeping between sample cycles. One must wake a sleeping OPD before suspending automated sampling.

command	description
<timesync> time	set time in seconds since start of epoch
<cdomrefflushtime> time	set CdomRefFlowTime parameter
<cdomrefon>	enable CdomRef sampling
<cdomrefoff>	disable CdomRef sampling
<cdomrefmliterleft> milliliters	set CdomRefLitersLeft parameter
<cdomrefrepeatrate>	set CdomRefRepeatInterval parameter
<cdomon>	enable cdom sampling
<cdomoff>	disable cdom sampling
<closedown>	shutoff all components and exit
<cycle>	enable sampling
<cycletime> minutes	set CycleTime parameter
<discon>	enable discriminant sampling
<discoff>	disable discriminant sampling
<discrepeat> count	set DiscRepeat parameter
<debugon>	enable debugging messages to console
<debugoff>	disable debugging messages to console

Figure 11 Console commands

6 Software Organization

The OPD control software is written in the C programming language. The core software is organized into a number of sources files, which are summarized briefly below.

- *mote.h* contains constant definitions and data structure definitions.
- *cfxmain.c* contains the main() function and the top level control loop that drives the automated data collection.

command	description
<opcode> number	perform manual command specified by numerical opcode
<numbercycles> count	set the NumberCycles parameter
</ready>	set the ackReady flag
<stop>	stop automated sampling and go to manual mode
<wake>	wake up from sleep mode
<param>	update a configuration parameter
<status>	set the sendStatus flag
auv	send Ready string to glider control

Figure 12 Console commands (continued)

- *sampleCycle.c* implements a single sample collection cycle, consisting of an optional CDOM reference phase, a CDOM phase and a discriminant phase.
- *control.c* contains functions used to initialize the OPD's hardware and provide the low level interface to the hardware.
- *psense.c* contains functions relating to the pressure sensor, including functions that calculate the OPD's depth in the ocean.
- *pumps.c* contains functions that deal with the sample pump and CDOM reference pump.
- *spect.c* contains functions that deal with the spectrophotometer.
- *gio.c* contains functions that interact with the console.
- *config.c* contains functions that read and write the OPD configuration file.
- *fvalve.c* contains functions relating to the filter valve.

opcode	description
1	exit
−2	close shutter
2	open shutter
−3	turn lights off
3	turn lights on
4	set filter valve to select filter
5	set filter valve to select bypass
6	set filter valve to closed
−7	turn off sample pump
7	turn on sample pump
−8	turn off cdomRef pump
8	turn on cdomRef pump
9	set pump speeds
10	set port valve to position A
−10	set port valve to position B
11	reverse sampling ports and turn on sample pump

Figure 13 Console command opcodes

- *valco.c* contains functions relating to the port valves.
- *logDebug.c* contains functions that write to the debug file, to log files and status files.
- *analysis.c* contains the data analysis functions.
- *misc.c* contains a variety of utility functions that are of general use.

There are a few idiosyncracies to the OPD control software that can make it difficult to understand on first encounter. First, while the program has to manage multiple concurrent activities, it is single-threaded, which complicates the control flow to some extent. In particular, console command processing is done by the same thread that handles automated data collection. Since some of the steps involved in automated data collection take

opcode	description
12	turn on debug messages to console
−12	turn off debug messages to console
13	calibrate Valco port valves
14	print sensors report
15	toggle the port valves and report position
16	power down for 10 seconds, then come back up
17	adjust integration time
18	get spectrophotometer serial number
19	set filter parameters
20	read the GPS coordinates
21	sampling setup report
22	cycle power repeatedly, until stop command received
25	set spectrophotometer baud rate
27	reset spectrophotometer
29	check that lights work
30	get wavelength polynomial coefficients

Figure 14 Console command opcodes

many seconds to complete, the program must periodically check for console input while it is doing automated data collection. The main control loop for the OPD can be summarized as follows.

```
while (true) {
    while (not interrupted) {
        check for console commands and respond appropriately
        if (automated sampling is enabled) {
            collect samples until interrupted
        } else if (there is a pending opcode command) {
            carry out the command
        }
    }
```

opcode	description
31	set Override flag
−31	clear Override flag
32	set spectrophotometer compression on
−32	set spectrophotometer compression off
33	enable pumpCreep
-33	disabe pumpCreep
34	pumpCreep on
−34	pumpCreep off
35	turn on deuterium light only
36	turn on tungsten light only
37	run sample pump for 60 seconds in bypass
38	run cdom reference pump for 60 seconds
39	get a dark spectrum
44	run filter blast
45	calibrate sample pump position sensor
46	position sample pump plunger at CdomStrokePos
47	reverse sample ports and go to CdomStrokePos

Figure 15 Console command opcodes (continued)

```
    respond to interruption and resume processing or exit
}
```

There are several ways that processing can be "interrupted". A break or Ctrl-C from the console will interrupt the program and cause it to exit. A <stop> command from the console will cause a *SUSPEND_EXCEPTION* to be thrown, causing the program to continue with automated sampling disabled. Other exceptions can lead to immediate termination, or resumption of normal processing from the start of the current sample cycle.

Because the program is single-threaded and many steps in the automated data collection stretch over many seconds, the program must check for console input at fairly regular intervals, while it is collecting data. The function

opcode	description
48	check downstream pressure sensor repeatedly
49	check the upstream pressure sensor repeatedly
50	check the filter pressure repeatedly
52	report status
53	set CycleCount parameter
54	set integrationTime parameter
55	print the CdomRefLitersLeft value
56	check the battery voltage repeatedly
60	run sample pump full speed to end of travel
61	read in config parameters and the species file names
62	write out the config parameters
63	read configuration parameters silently
64	write configuration parameters silently
65	Set CdomStrokePos, HP Units only
66	cpumpCal(); break; //calibrate the cdom reference pump

Figure 16 Configuration parameters (continued)

checkConsole() (in misc.c) checks for console input and if it detects that a command has been received, it responds appropriately. Most calls to *checkConsole*() occur during explict multi-second delays that are required between successive steps in the sample collection process. The *longDelay*() function calls *checkConsole*() once per second during these long delay periods.

Most calls to *checkConsole*() find no new command and simply return immediately. When a command is detected, it can often be handled immediately, allowing *checkConsole* to return control to the calling function in the normal way. There are two exceptions to this. If a <stop> command is received, a `SUSPEND_EXCEPTION` is thrown, causing the program to return to the *main*() function where the exception handling code is found. Similarly, a <closedown> command will cause an `EXIT_EXCEPTION` which also returns control to the exception handling code in the *main*() function.

opcode	description
-67	stop pumping cdomRef
67	pump cdomRef waste overboard
68	run sample pump for 10 seconds in bypass
69	Set CdomVolume, LP Units only
70	setMaxFilterPressure(); break;
71	filterBackFlush(); break;
72	query the Valco encoder
99	setDiscFlowTime(); break;
100	send picoDos command (not implemented)
110	copy file to opd (not implemented)
111	DoBSOP = true; break;
112	DoMooring = true; break;
113	DoREMUS = true; break;
114	DoGlider = true; break;
115	turn on POLLED mode
-115	turn off POLLED mode
116	set dualpressure flag
-116	clear dualpressure flag

Figure 17 Configuration parameters (continued)

The C programming language does not have explicit language constructs for handling exceptions. However it does have a somewhat primitive mechanism that can serve the same purpose. The OPD control program uses the *setjmp*() and *longjmp*() functions to implement exception-handling. The exception handling code is defined in a `switch` statement in the *main* function. Other parts of the program can effectively throw an exception, by calling the *longjmp*() function with an integer parameter that specifies which exception is being thrown. This returns control to the appropriate exception handling code in the main program, which can then either terminate the program or continue.

Detailed documentation of the OPD control software appears in the appendices that follow. This documentation was generated using *doxygen* [1] from structured comments in the code itself.

References

[1] *Doxygen source code documentation generator*, `www.stack.nl/~dimitri/doxygen/` 27

[2] Hails, A., C. Boyes, A. Boyes, R. D. Currier, K. Henderson, A. Kotlewski and G.J. Kirkpatrick "The Optical Phytoplankton Discriminator," *Proceedings of Oceans*, 2009. 1, 2

[3] Kirkpatrick, Gary J., David F. Millie, Mark A. Moline, Steven E. Lohrenz and Oscar M. Schofield. "Automated, in-water determination of colored dissolved organic material and phytoplankton community structure using the optical phytoplankton discriminator," In *Proceedings SPIE 8030, Ocean Sensing and Monitoring III*, May 04, 2011. 1, 2

[4] *Persistor CF2 Getting Started Guide*, `www.persistor.com/doc/docindex.html`. 6, 12

[5] *Persistor CF2 Programmer's Manual*, `www.persistor.com/doc/docindex.html`. 6

[6] *Persistor CF2 API Reference*, `www.persistor.com/doc/docindex.html`. 6, 12

[7] *Persistor CF2 Managing CF2 Behavior with PicoDOS*, `www.persistor.com/doc/docindex.html`. 6

Detailed Documentation

A cfxmain.c File Reference

This file contains the main() function, which processes command-line arguments, initializes the OPD software and runs the top level control loop.

Functions

- int main (int argc, char **argv)

  Main function for OPD control program.
- void collectSamples ()

  Main sample collection loop.
- bool deployModeCheck ()

  Perform various sanity checks related to the deployment mode.
- void cleanup ()

  Prepare for a normal exit of the program.
- IEV_C_FUNCT (IRQHandler)

  Power-fail interrupt handler.

Variables

- bool OPDinitialized

  flag set when initialization has been completed.
- double RunDepth

  target operating depth for auv deployments
- double DepthMeters

  most recently measured depth value

- bool ackReady

  global flag used to indicate that a </ready> command has been received from console

- bool Overpressure

  overpressure flag, set when overpressure condition is detected

- bool CDOMRefisDone

  flag that indicates that a CDOM reference phase has been completed

- int CdomRefRepeatInterval

  specifies frequency with which CDOM reference phases are performed

- long PixelMax

  largest light intensity value in most recently acquired spectrum

- long Railed

  max threshold for validity of spectrum

- double BatteryVoltage

  most recently measured value of battery voltage

- int Status

  vector of status bits, set by digitalStatus

- int integrationTime

  spectrometer integration time parameter

- float AbsorbanceA

  characteristic parameters of the cdom absorption spectrum

- double CorrResults [MaxSpecies]

  correlation results from disc against each reference spectrum

- time_t TraceStart [2]

  time at start of CDOM or discriminant phase

- time_t TraceStop [2]

  time at end of CDOM or discriminant phase

- unsigned int NumberCycles = 0

  number of automated sample collection cycles

- int CyclesDone = 0

number of cycles that have been completed so far

- int CycleTime = 5

  time between succesive cycles (in minutes)

- enum Vehicle ActiveVehicle

  specifies deployment context

- double drivespace

  record space available on compact flash for file storage

- bool enableSampling = false

  flag controlling sampling, may be disabled by <stop> command

- bool Wake = true

  flag reflecting awake/asleep status, can be set by <wake> command

- bool DoCdomRef = true

  flag to enable/disable CDOM reference phase

- bool DoCdom = true

  flag to enable/disable CDOM phase

- bool DoDisc = true

  flag to enable/disable discriminant phase

- bool debug = false

  flag to enable/disable extra debugging messages to console

- bool dualpressure = false

  flag used to indicate units with dual pressure sensors

- bool encoder = false

  flag used to indicate units with valco encoders

- bool lowPower = false

  flag used to enable/disable low power mode

- bool cellphone = false

  flag used to enable/disable reporting of results via cell phone

- bool NoHost = false

  flag used to enable/disable automated sample collection on startup

- bool simulate = false

flag used to enable/disable simulated operation

- bool POLLED = false

  flag used to enable polled status reporting
- long pflag = 0

  flag used to track power failures
- long sawInt = 0

  flag used to record occurrence of power-fail interrupt
- int OpCode = 0

  opcode that was most recently received from console
- unsigned int SYSClock = 15040

  system clock frequency (in KHz)
- bool versionreport = false

  flag used to trigger version report and immediate exit
- jmp_buf exceptionContext

  setjmp context used to impelment exception handling

A.1 Detailed Description

This file contains the main() function, which processes command-line arguments, initializes the OPD software and runs the top level control loop.

Author

Mote Marine Lab - Ocean Technology Group

Date

2005-2016

Definition in file cfxmain.c.

A.2 Function Documentation

A.2.1 int main (int *argc*, char ** *argv*)

Main function for OPD control program.

Following command-line processing and initialization, enters the top-level control loop. If operating in manual mode, this control loop just processes console commands. If operating in automated sampling mode, invokes the collectSamples function which performs automated sampling until a specified number of sample cycles have been completed, or sampling has been interrupted for any of several reasons.

Parameters

argc	is the number of command-line arguments
argv	is the vector of argument strings

Definition at line 241 of file cfxmain.c.

A.2.2 void collectSamples ( )

Main sample collection loop.

Continues until either a specified number of sample cycles have been completed, or automated sampling is interrupted. Sampling may be interrupted by a console stop command, a pressure exception, a power exception or a leak exception. Between sample cycles, the program goes to sleep, after turning off the various devices, in order to conserve power.

Definition at line 552 of file cfxmain.c.

A.2.3 bool deployModeCheck ( )

Perform various sanity checks related to the deployment mode.

Returns

true if all checks passed, else false

Definition at line 517 of file cfxmain.c.

A.2.4 void cleanup (void)

Prepare for a normal exit of the program.

Definition at line 214 of file cfxmain.c.

A.2.5 IEV_C_FUNCT (IRQHandler)

Power-fail interrupt handler.

Control reaches here when power to the persistor is about to fail. Just enough time to do some essential cleanup.

Definition at line 167 of file cfxmain.c.

A.3 Variable Documentation

A.3.1 bool OPDinitialized

flag set when initialization has been completed.

Definition at line 80 of file control.c.

B sampleCycle.c File Reference

This file contains the code for collecting a set of samples.

Functions

- bool sampleCycle ()

  Collect a set of sea water samples.
- bool collectCdomRef (double ∗cdomRef)

  Perform CdomRef measurement.
- bool collectCdom (double ∗cdom, double ∗dark)

  Perform a Cdom measurement.
- bool collectDisc (double ∗cdomRef, double ∗cdom, double ∗dark)

  Collect a discriminant spectrum, compute fourth-derivative and compare it to all reference species.
- void discSpectMooring (double ∗cdomRef, double ∗cdom, double ∗dark)

  Collect discriminant spectrum and compare to reference spectra.
- void discSpectHP (double ∗cdomRef, double ∗cdom, double ∗dark)

  Collect discriminant spectrum and compare to reference spectra.
- void discSpectRemus (double ∗cdomRef, double ∗cdom, double ∗dark)

  Collect discriminant spectrum May collect additional discriminant samples after initial sample, based on DiscRepeat parameter, with delay specified by DiscFlowTime parameter.
- void sleepToNext (void)

  Sleep until time for next sample collection cycle.
- void sleepToNextInt (void)

  Sleep until time for next cycle, using interrupt.
- void sleepToNextMooring (void)

  Sleep until time for next cycle.

Variables

- bool DoCdomRef

  flag to enable/disable CDOM reference phase
- bool DoCdom

  flag to enable/disable CDOM phase
- bool DoDisc

  flag to enable/disable discriminant phase
- jmp_buf exceptionContext

  setjmp context used to impelment exception handling
- int CdomRefPumpSpeed

  speed at which run the cdom reference pump (range is 0 to 255)
- bool enableSampling

  flag controlling sampling, may be disabled by <stop> command
- bool Overpressure

  overpressure flag, set when overpressure condition is detected
- bool cellphone

  flag used to enable/disable reporting of results via cell phone
- bool Wake

  flag reflecting awake/asleep status, can be set by <wake> command
- bool ackReady

  global flag used to indicate that a </ready> command has been received from console
- bool lowPower

  flag used to enable/disable low power mode
- bool pumpCreepEnable

  flag used to enable slow pumping during disc spectrum acquisition.
- unsigned short CdomStrokePos

  sample pump position corresponding to completed cdom sample
- double FilterPressure

most recently measured value of filter pressure

- enum Vehicle ActiveVehicle

  specifies deployment context

- unsigned int integrationTime

  spectrometer integration time parameter

- int CyclesDone

  number of cycles that have been completed so far

- int CycleTime

  time between succesive cycles (in minutes)

- double drivespace

  record space available on compact flash for file storage

- long PixelMax

  largest light intensity value in most recently acquired spectrum

- long Railed

  max threshold for validity of spectrum

- int Status

  vector of status bits, set by digitalStatus

- double DepthMeters

  most recently measured depth value

- unsigned int CdomRefFlowTime = 12

  time to run CDOM reference pump at start of CDOM reference phase

- int CdomRefRepeatInterval = 8

  specifies frequency with which CDOM reference phases are performed

- double CdomVolume = 2.0

  volume of filtered seawater to pump during CDOM phase (in milliliters)

- int DiscFlowTime = 15

  time to run sample pump during discriminant phase

- unsigned int DiscRepeat = 2

  number of extra discriminant samples to collect during discriminant phase

- bool CDOMRef_integration_time = false

flag that indicates that a valid integration time has been obtained

- bool CDOMRefisDone = true

  flag that indicates that a CDOM reference phase has been completed

- double StartDepth [2] = { 0, 0 }

  when OPD deployed on REMUS, records depth at start of CDOM and discriminant phases

- double StopDepth [2] = { 0, 0}

  when OPD deployed on REMUS, records depth at end of CDOM and discriminant phases

- time_t TraceStart [2] = { 0, 0 }

  time at start of CDOM or discriminant phase

- time_t TraceStop [2] = { 0, 0}

  time at end of CDOM or discriminant phase

B.1 Detailed Description

This file contains the code for collecting a set of samples. It also includes code that handles the inter-cycle sleep.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file sampleCycle.c.

B.2 Function Documentation

B.2.1 bool sampleCycle ( )

Collect a set of sea water samples.

A set of samples includes an optional CDOM reference sample, a CDOM sample and one or more discriminant samples

Returns

true if all steps were completed successfully, else false; may throw PRESSURE_EXCEPTION.

Definition at line 112 of file sampleCycle.c.

B.2.2 bool collectCdomRef (double * *cdomRef*)

Perform CdomRef measurement.

First, adjust the spectrometer light level to get maximum sensitivity, then collect cdomRef sample and return cooked spectrum.

Parameters

cdomRef	is a pointer to an array of NMAX doubles in which the cooked cdom spectrum is returned (after subtracting the dark spectrum).

Returns

true on success, else false; may throw PRESSURE_EXCEPTION.

Definition at line 223 of file sampleCycle.c.

B.2.3 bool collectCdom (double * *cdom*, double * *dark*)

Perform a Cdom measurement.

This may include an attempt to adjust integration time if integration time was not successfully adjusted in a previous cdomRef step.

Parameters

cdom	is a pointer to NMAX doubles in which the cooked cdom spectrum is returned (after subtracting the dark spectrum)
dark	is a pointer to NMAX doubles in which the dark spectrum is returned

Returns

true if successful, else false; may throw PRESSURE_EXCEPTION

Definition at line 303 of file sampleCycle.c.

B.2.4 bool collectDisc (double * *cdomRef*, double * *cdom*, double * *dark*)

Collect a discriminant spectrum, compute fourth-derivative and compare it to all reference species.

Parameters

cdomRef	is a cooked cdomRef spectrum from which dark has been subtracted
cdom	is a cooked cdom spectrum from which dark has been subtracted
dark	is a cooked dark spectrum to be subtracted from disc spectra

Returns

true if successful, else false; may throw PRESSURE_EXCEPTION

Definition at line 459 of file sampleCycle.c.

B.2.5 void discSpectMooring (double * *cdomRef*, double * *cdom*, double * *dark*)

Collect discriminant spectrum and compare to reference spectra.

May collect additional discriminant samples after initial sample, based on DiscRepeat parameter, with delay specified by DiscFlowTime parameter. These additional spectra are not compared to reference spectra. Copies of all raw spectra are sent to log file. This function called only in mooring deployments.

Parameters

cdomRef	is a pointer to a previously obtained cdomRef spectrum
cdom	is a pointer to a previously obtained cdom spectrum
dark	is a pointer to a previously obtained dark spectrum

Definition at line 511 of file sampleCycle.c.

B.2.6 void discSpectHP (double * *cdomRef*, double * *cdom*, double * *dark*)

Collect discriminant spectrum and compare to reference spectra.

Collect second spectrum also, but skip comparison to reference spectra. Copies of all raw spectra are sent to log file. This function is called only in high pressure units.

Parameters

cdomRef	is a pointer to a previously obtained cdomRef spectrum
cdom	is a pointer to a previously obtained cdom spectrum
dark	is a pointer to a previously obtained dark spectrum

Definition at line 550 of file sampleCycle.c.

B.2.7 void discSpectRemus (double * *cdomRef*, double * *cdom*, double * *dark*)

Collect discriminant spectrum May collect additional discriminant samples after initial sample, based on DiscRepeat parameter, with delay specified by DiscFlowTime parameter.

Used only in high pressure units.

Parameters

cdomRef	is a pointer to a previously obtained cdomRef spectrum
cdom	is a pointer to a previously obtained cdom spectrum
dark	is a pointer to a previously obtained dark spectrum

Definition at line 581 of file sampleCycle.c.

B.2.8 void sleepToNext (void)

Sleep until time for next sample collection cycle.

Turns off power to instruments and then waits for next cycle, checking for console input while waiting.

Definition at line 608 of file sampleCycle.c.

B.2.9 void sleepToNextInt (void)

Sleep until time for next cycle, using interrupt.

Turns off power to instruments, slows processor clock and then waits for interrupt to trigger next cycle.

Definition at line 665 of file sampleCycle.c.

B.2.10 void sleepToNextMooring (void)

Sleep until time for next cycle.

This version used in mooring deployments. Turns off power to instruments and then waits for next cycle, checking for console input while waiting.

Definition at line 727 of file sampleCycle.c.

B.3 Variable Documentation

B.3.1 bool pumpCreepEnable

flag used to enable slow pumping during disc spectrum acquisition.

Definition at line 55 of file pumps.c.

C config.c File Reference

This file contains functions for reading and writing configuration parameters.

Functions

- int cfg_parseLine (char *line)

  Parse a line from the configuration file.
- bool cfg_write ()

  Write the configuration file.
- bool cfg_update (char *buf)

  Update a configuration variable.
- void cfg_printDebug ()

  Write configuration parameters and species file names to the debug file.
- void cfg_consoleShow ()

  Write configuration parameters and species file names to the console.
- void cfg_limitCheck ()

  Verify that configuration parameters are within prescribed limits.
- int cfg_read ()

  Read the configuration file (Config.txt).

Variables

- unsigned int d_PosSensorMinLo = 1000

  smallest allowed value for PosSensorMin
- unsigned int d_PosSensorMinHi = 5000

  largest allowed value for PosSensorMin
- unsigned int d_PosSensorMaxLo = 20000

  smallest allowed value for PosSensorMax
- unsigned int d_PosSensorMaxHi = 40000

  largest allowed value for PosSensorMax
- unsigned int d_integrationTimeMin = 10

  smallest allowed value for integrationTime
- unsigned int d_integrationTimeMax = 500

largest allowed value for integrationTime

- unsigned int d_SamplePumpSpeedMin = 20

  smallest allowed value for SamplePumpSpeed
- unsigned int d_SamplePumpSpeedMax = 127

  largest allowed value for SamplePumpSpeed
- unsigned int d_SamplePumpSpeedDefault = 100

  default value for SamplePumpSpeed
- unsigned int d_CdomRefPumpSpeedMin = 20

  smallest allowed value for CdomRefPumpSpeed
- unsigned int d_CdomRefPumpSpeedMax = 255

  largest allowed value for CdomRefPumpSpeed
- unsigned int d_CdomRefPumpSpeedDefault = 200

  default value for CdomRefPumpSpeed
- unsigned int d_BBSerialNumberMin = 1

  smallest allowed value for BBSerialNumber
- unsigned int d_BBSerialNumberMax = 99

  largest allowed value for BBSerialNumber
- double d_WaveGuideLengthMin = .28

  smallest allowed value for WaveGuideLength
- double d_WaveGuideLengthMax = .75

  largest allowed value for WaveGuideLength
- bool ok2writeConfig = false

  flag set after configuration parameters have be read for first time; prevents writing config parameters before valid initial values have been read in
- unsigned int NumberCycles

  number of automated sample collection cycles
- int CyclesDone

  number of cycles that have been completed so far
- int CycleTime

  time between succesive cycles (in minutes)

- unsigned int CdomRefPumpCal

  Cdom reference pump calibration parameter (for determing volume pumped)

- unsigned int PosSensorMax

  syringe pump sensor value at max end of travel

- unsigned int PosSensorMin

  syringe pump sensor value at min end of travel

- unsigned int SamplePumpPosition

  current position of sample pump (for high pres.

- unsigned short CdomStrokePos

  sample pump position corresponding to completed cdom sample

- unsigned int integrationTime

  spectrometer integration time parameter

- bool DoCdomRef

  flag to enable/disable CDOM reference phase

- unsigned int CdomRefFlowTime

  time to run CDOM reference pump at start of CDOM reference phase

- int CdomRefRepeatInterval

  specifies frequency with which CDOM reference phases are performed

- bool DoCdom

  flag to enable/disable CDOM phase

- double CdomVolume

  volume of filtered seawater to pump during CDOM phase (in milliliters)

- bool DoDisc

  flag to enable/disable discriminant phase

- int DiscFlowTime

  time to run sample pump during discriminant phase

- unsigned int DiscRepeat

  number of extra discriminant samples to collect during discriminant phase

- int FilterSize

full window size for gaussian smoothing filter

- double FilterSigma

  standard deviation for gaussian smoothing filter

- unsigned int CorrWaveMax

  high end of the range of wavelengths over which the correlation metric is computed

- unsigned int CorrWaveMin

  low end of the range of wavelengths over which the correlation metric is computed

- unsigned int AbWaveMax

  high end of range of wavelengths used to fit shifted absorption curve to exponential function

- unsigned int AbWaveMid

  midpoint of range of wavelengths used to fit shifted absorption curve to exponential function

- unsigned int AbWaveMin

  low end of range of wavelengths used to fit shifted absorption curve to exponential function

- unsigned int MaxFilterPressure

  maximum allowed pressure across the filter (in psi)

- double RunDepth

  target operating depth for auv deployments

- double ShutDownDepth

  minimum operating depth for auv deployments

- double DepthCoefA

  first of two calibration coefficients used in depth measurement

- double DepthCoefB

  second of two calibration coefficients used in depth measurement

- unsigned char SerialNumber [30]

  serial number used to identify spectrometer

- enum Vehicle ActiveVehicle

specifies deployment context

- double WaveGuideLength

  length of spectrometer's waveguide
- int SamplePumpSpeed

  speed at which run the sample pump (range is -127 to +127)
- int CdomRefPumpSpeed

  speed at which run the cdom reference pump (range is 0 to 255)
- int nSpecies

  number of species files
- char Species [MaxSpecies][20]

  names of species files containing reference spectra
- struct par config [ ]

  list of configuration parameters

C.1 Detailed Description

This file contains functions for reading and writing configuration parameters.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file config.c.

C.2 Function Documentation

C.2.1 int cfg_parseLine (char * *line*)

Parse a line from the configuration file.

Parameters

line	is a character string containing the name of a configuration parameter and its desired value.

Returns

1 if the configuration parameter was successfully updated, else 0.

Definition at line 270 of file config.c.

C.2.2 bool cfg_write (void)

Write the configuration file.

Returns

true on success, else false

Definition at line 337 of file config.c.

C.2.3 bool cfg_update (char * *buf*)

Update a configuration variable.

Parameters

buf	is a pointer to a buffer specifying a configuration variable and a new value for that variable

Returns

true if the configuration variable was updated successfully, else false

Definition at line 328 of file config.c.

C.2.4 int cfg_read (void)

Read the configuration file (Config.txt).

Sets the parameters as specified in the file.

Returns

the number of parameters successfully read

Definition at line 238 of file config.c.

C.3 Variable Documentation

C.3.1 unsigned int SamplePumpPosition

current position of sample pump (for high pres.

units with syringe pumps)

Definition at line 49 of file pumps.c.

D gio.c File Reference

This file contains functions for handling command-line input from the the console.

Functions

- void gio_setMaxFilterPressure (void)

  Set the MaxFilterPressure parameter interactively.
- void gio_setDiscFlowTime (void)

  Set the DiscFlowTime parameter interactively.
- void showConfig (void)

  Display command-line arguments, configuration parameters and species files on the console.
- bool gio_handleInput ()

  Read console input, possibly processing multiple input lines.
- bool gio_getLine (char *lineBuf)

  Get a line of input from the console.
- bool gio_processLine (char *line)

  Process a line received from the console.
- bool gio_processOpcode (int opcode)

  Process opcode entered on the command line.
- void gio_pauseHold ()

  In REMUS deployment, shut down if failing to reach operating depth .
- bool gio_waitPolled ()

  Wait up to 11 minutes for a <status> request.
- void gio_phone ()

  Send commands over serial port to cell phone modem to establish remote connection for status reporting.
- void gio_gOutput (char *buf)

  Send a string to the OPD serial port.

Variables

- bool ackReady = false

  global flag used to indicate that a </ready> command has been received from console
- jmp_buf exceptionContext

  setjmp context used to impelment exception handling
- bool DoCdomRef

  flag to enable/disable CDOM reference phase
- bool DoCdom

  flag to enable/disable CDOM phase
- bool DoDisc

  flag to enable/disable discriminant phase
- bool CDOMRef_integration_time

  flag that indicates that a valid integration time has been obtained
- bool EndOfTravel

  flag set by digitalStatus to indicate that syringe pump plunger is at one end or the other of its travel range
- bool Wake

  flag reflecting awake/asleep status, can be set by <wake> command
- bool enableSampling

  flag controlling sampling, may be disabled by <stop> command
- bool debug

  flag to enable/disable extra debugging messages to console
- bool lightTungsten

  flag used to turn individually control the tungsten light
- bool lightDeuterium

  flag used to turn individually control the deuterium light
- bool encoder

  flag used to indicate units with valco encoders

- bool dualpressure

  flag used to indicate units with dual pressure sensors
- bool Leak

  flag set by digitalStatus to indicate that a leak has been detected
- bool Overpressure

  overpressure flag, set when overpressure condition is detected
- bool pumpCreepEnable

  flag used to enable slow pumping during disc spectrum acquisition.
- bool compression

  flag controlling use of compression
- bool POLLED

  flag used to enable polled status reporting
- bool NoHost

  flag used to enable/disable automated sample collection on startup
- bool simulate

  flag used to enable/disable simulated operation
- bool cellphone

  flag used to enable/disable reporting of results via cell phone
- int Status

  vector of status bits, set by digitalStatus
- int OpCode

  opcode that was most recently received from console
- int CyclesDone

  number of cycles that have been completed so far
- int CycleTime

  time between succesive cycles (in minutes)
- unsigned int NumberCycles

  number of automated sample collection cycles
- bool SamplePump

  flag used to turn on/off power to the sample pump

- bool CdomPump

    flag used to turn on/off power to the cdom reference pump

- int SamplePumpSpeed

    speed at which run the sample pump (range is -127 to +127)

- int CdomRefPumpSpeed

    speed at which run the cdom reference pump (range is 0 to 255)

- unsigned int SamplePumpPosition

    current position of sample pump (for high pres.

- int CdomRefRepeatInterval

    specifies frequency with which CDOM reference phases are performed

- unsigned int CdomRefFlowTime

    time to run CDOM reference pump at start of CDOM reference phase

- double CdomVolume

    volume of filtered seawater to pump during CDOM phase (in milliliters)

- unsigned short CdomStrokePos

    sample pump position corresponding to completed cdom sample

- unsigned int DiscRepeat

    number of extra discriminant samples to collect during discriminant phase

- int DiscFlowTime

    time to run sample pump during discriminant phase

- unsigned int integrationTime

    spectrometer integration time parameter

- double WavePoly [4]

    wavelength polynomial, true value read from the spectrometer

- unsigned char SerialNumber [30]

    serial number used to identify spectrometer

- double FilterPressure

    most recently measured value of filter pressure

- unsigned int MaxFilterPressure

    maximum allowed pressure across the filter (in psi)

- double Pressure0

  most recent pressure reading downstream of filter

- double Pressure3

  most recent pressure reading upstream of filter

- unsigned int PosSensorMax

  syringe pump sensor value at max end of travel

- unsigned int PosSensorMin

  syringe pump sensor value at min end of travel

- double DepthMeters

  most recently measured depth value

- double RunDepth

  target operating depth for auv deployments

- double ShutDownDepth

  minimum operating depth for auv deployments

- int FilterSize

  full window size for gaussian smoothing filter

- double FilterSigma

  standard deviation for gaussian smoothing filter

- int nSpecies

  number of species files

- double BatteryVoltage

  most recently measured value of battery voltage

- long sawInt

  flag used to record occurrence of power-fail interrupt

- enum Vehicle ActiveVehicle

  specifies deployment context

D.1 Detailed Description

This file contains functions for handling command-line input from the the console.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file gio.c.

D.2 Function Documentation

D.2.1 bool gio_getLine (char * *lineBuf*)

Get a line of input from the console.

Uses SCIxxxx commands for the Persistor Main RS-232 console comms via the MAX 3222 driver IC.to pins 43 and 44

Parameters

lineBuf	is a pointer to a character buffer in which line is returned.

Returns

true if a complete line was processed, else false

Definition at line 115 of file gio.c.

D.2.2 bool gio_processLine (char * *line*)

Process a line received from the console.

Parameters

line	is pointer to a command line to be processed

Returns

true if line has been processed, else false; may throw suspend exception or exit exception

Definition at line 148 of file gio.c.

D.2.3 bool gio_processOpcode (int *opcode*)

Process opcode entered on the command line.

This function is called from main when sample collection has been suspended.

Parameters

opcode	is the value of the opcode to be processed.

Returns

true if opcode processed normally, false if an error occurred that requires termination

Definition at line 303 of file gio.c.

D.2.4 void gio_pauseHold (void)

In REMUS deployment, shut down if failing to reach operating depth .

Throws exit exception if OPD depth is less than ShutDownDepth and does not reach RunDepth within 60 seconds.

Definition at line 1096 of file gio.c.

D.2.5 bool gio_waitPolled (void)

Wait up to 11 minutes for a <status> request.

Used to delay status reporting until NOAA ready for it.

Definition at line 1122 of file gio.c.

D.2.6 void gio_gOutput (char * *buf*)

Send a string to the OPD serial port.

Intended for use when serial port used to communicate with cell phone modem or glider's science computer, etc.

Parameters

buf	is pointer to string to be sent to serial port

Definition at line 1208 of file gio.c.

D.3 Variable Documentation

D.3.1 bool pumpCreepEnable

flag used to enable slow pumping during disc spectrum acquisition.

Definition at line 55 of file pumps.c.

D.3.2 unsigned int SamplePumpPosition

current position of sample pump (for high pres.

units with syringe pumps)

Definition at line 49 of file pumps.c.

E logDebug.c File Reference

This file contains functions for recording information in log files, status files and debug file.

Functions

- void logInit ()

  Initialize (or re-initialize) debug and log files.
- void logClose ()

  Close the log file and debug file.
- void logString (char *s)

  Write a character string to the log file.
- void logIvec (ushort *v, ushort n)

  Write a vector of integers to the log file.
- void logDateTime ()

  Write current date/time string to the log file.
- void logSpectrum (char *label, ushort *spect)

  Write a spectrum to the log file.
- void logHeader ()

  Write a log file header to the log file.
- void debugTime ()

  Write the time of day to the debug file.
- void debugDateTime ()

  Write the date and time of day to the debug file.
- void debugEtime ()

  Write the elapsed time to the debug file.
- void bprintf (char *format,...)

  Write a debug message to both the console and the debugFile.
- void dprintf (char *format,...)

  Write a debug message to the debug file and optionally, to the console.
- void reportStatus ()

  Generate a status report.
- void print2statusFile (char *status)

  Add a json status report to the .stt file.

Variables

- bool debug

  flag to enable/disable extra debugging messages to console
- bool POLLED

  flag used to enable polled status reporting
- bool OPDinitialized

  flag set when initialization has been completed.
- enum Vehicle ActiveVehicle

  specifies deployment context
- double StartDepth [2]

  when OPD deployed on REMUS, records depth at start of CDOM and discriminant phases
- double StopDepth [2]

  when OPD deployed on REMUS, records depth at end of CDOM and discriminant phases
- int Status

  vector of status bits, set by digitalStatus
- int nSpecies

  number of species files
- double CorrResults [MaxSpecies]

  correlation results from disc against each reference spectrum
- unsigned int integrationTime

  spectrometer integration time parameter
- double WavePoly [4]

  wavelength polynomial, true value read from the spectrometer
- unsigned char SerialNumber [30]

  serial number used to identify spectrometer
- double BatteryVoltage

  most recently measured value of battery voltage

- time_t TraceStart [2]

  time at start of CDOM or discriminant phase
- time_t TraceStop [2]

  time at end of CDOM or discriminant phase
- double FilterPressure

  most recently measured value of filter pressure
- double AbsorbanceA

  characteristic parameters of the cdom absorption spectrum
- char Species [MaxSpecies][20]

  names of species files containing reference spectra
- jmp_buf exceptionContext

  setjmp context used to impelment exception handling
- double WaveGuideLength

  length of spectrometer's waveguide
- FILE * debugFile = NULL

  pointer to debugging file structure
- FILE * logFile = NULL

  pointer to file structure for current log file

E.1 Detailed Description

This file contains functions for recording information in log files, status files and debug file.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file logDebug.c.

E.2 Function Documentation

E.2.1 void logInit ( )

Initialize (or re-initialize) debug and log files.

This function should be called when program starts and at the start of each sample cycle. Will open a new log file when a new reporting period has started. Is designed to tolerate power outages without losing log file data.

Debug messages are printed using dprintf(...).

Definition at line 75 of file logDebug.c.

E.2.2 void logClose ( )

Close the log file and debug file.

Definition at line 126 of file logDebug.c.

E.2.3 void logString (char * *s*)

Write a character string to the log file.

Writes character 'C', followed by the string length (as a 16 bit value with the low-order byte first), followed by the characters in the string (does not include '\0' at the end).

Parameters

s	is a pointer to a string.

Definition at line 138 of file logDebug.c.

E.2.4 void logIvec (ushort ∗ *v*, ushort *n*)

Write a vector of integers to the log file.

Writes character 'I', followed by the vector length (as a 16 bit value with the low-order byte first), followed by the integer values (as 16 bit values with the low-order byte first).

Parameters

v	is a pointer to a vector.
n	is the length of the vector.

Definition at line 154 of file logDebug.c.

E.2.5 void logSpectrum (char ∗ *label*, ushort ∗ *spect*)

Write a spectrum to the log file.

Parameters

label	is a pointer to one of "Cdom", CdomRef", "Dark" or "Disc".

spect	is a pointer to a vector of OMAX ushorts.

Definition at line 180 of file logDebug.c.

E.2.6 void debugEtime ( )

Write the elapsed time to the debug file.

The format is [sss.mmm] where ss is seconds, mm is milliseconds.

Definition at line 239 of file logDebug.c.

E.2.7 void bprintf (char ∗ *format*, ...)

Write a debug message to both the console and the debugFile.

Parameters

is	a printf-style format string, which may include formatting directives for optional arguments that follow

Definition at line 251 of file logDebug.c.

E.2.8 void dprintf (char ∗ *format*, ...)

Write a debug message to the debug file and optionally, to the console.

These messages are shown on the console only if the global debug variable is enabled

Parameters

format	is a printf-style format string, which may include formatting directives for optional arguments that follow

Definition at line 269 of file logDebug.c.

E.2.9 void reportStatus ( )

Generate a status report.

The report is output in terse form and as a json string.

Definition at line 288 of file logDebug.c.

E.2.10 void print2statusFile (char * *status*)

Add a json status report to the .stt file.

Parameters

status	is a json-formatted status report

Definition at line 399 of file logDebug.c.

E.3 Variable Documentation

E.3.1 bool OPDinitialized

flag set when initialization has been completed.

Definition at line 80 of file control.c.

F control.c File Reference

This file contains various functions for controlling the OPD hardware.

Functions

- void setDigitalIO (void)

  Set various digital controls based on global flags.
- void digitalStatus (void)

  Read the digital status of the hardware and set status flags.
- bool initialize (void)

  Perform overall system initialization.
- void instrumentsOn ()

  Turn on power to instruments, initialize control flags.
- void instrumentsOff ()

  Turn off power to instruments, clear control flags.
- void powerUp ()

  Turn on power to the instruments and initialize them.
- void powerDown ()

  Turn off power to the instruments.
- void setupControl ()

  Initialize the various devices to their startup states.
- void resetControl (void)

  Turn off power to instruments.
- void idleMode ()

  Go to idle mode.
- ushort getADSample (int chan)

  Read a sample from the AD converter (Burr-Brown ADS8344).
- double getBatteryVoltage (void)

Read the battery voltage.

- void handleOverpressure ()

  Attempt to clear overpressure condition caused by buble or dirt.
- void filterBlast (void)

  Run the sample pump in short bursts in the reverse of its current direction.
- bool haveLeak (void)

  Check for a leak.
- void setOverride (bool on)

  Turns on/off the Override flag.

Variables

- QPBDev digital_io

  SPI bus device structure used for communicating with power control board.
- QPBDev potentiometer

  SPI bus device structure used for communicating with potentiometer that controls pump speeds.
- double BatteryVoltage

  most recently measured value of battery voltage
- double FilterPressure

  most recently measured value of filter pressure
- bool OPDinitialized = false

  flag set when initialization has been completed.
- int Status = 0

  vector of status bits, set by digitalStatus
- bool Power

  flag used to control overall power setting
- bool Override

  flag used to turn on/off power to the port valves
- bool SamplePump

flag used to turn on/off power to the sample pump

- bool CdomPump

  flag used to turn on/off power to the cdom reference pump
- bool Lights

  flag used to turn on/off the spectrometer lights
- bool lightDeuterium = true

  flag used to turn individually control the deuterium light
- bool lightTungsten = true

  flag used to turn individually control the tungsten light
- bool Shutter

  flag used to open/close the shutter for the light source
- bool Filter

  flag used to control the filter valve
- bool FilterSelected

  flag set by digitalStatus when filterValve set to allow fluid to flow through filter
- bool BypassSelected

  flag set by digitalStatus when filterValve set to send fluid through bypass path (around filter)
- bool EndOfTravel

  flag set by digitalStatus to indicate that syringe pump plunger is at one end or the other of its travel range
- bool Leak

  flag set by digitalStatus to indicate that a leak has been detected
- jmp_buf exceptionContext

  setjmp context used to impelment exception handling
- int SamplePumpSpeed

  speed at which run the sample pump (range is -127 to +127)
- int CdomRefPumpSpeed

  speed at which run the cdom reference pump (range is 0 to 255)

- bool encoder

    flag used to indicate units with valco encoders
- bool Overpressure

    overpressure flag, set when overpressure condition is detected
- bool enableSampling

    flag controlling sampling, may be disabled by <stop> command
- enum Vehicle ActiveVehicle

    specifies deployment context

F.1 Detailed Description

This file contains various functions for controlling the OPD hardware.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file control.c.

F.2 Function Documentation

F.2.1 void setDigitalIO (void)

Set various digital controls based on global flags.

Configures the hardware to match the global variables Power, Shutter, Lights, lightDeuterium, lightTungsten, SamplePump, Filter, CdomPump, Override.

Definition at line 145 of file control.c.

F.2.2 void digitalStatus (void)

Read the digital status of the hardware and set status flags.

Global status flags EndOfTravel, FilterSelected, BypassSelected and Leak are updated.

Definition at line 166 of file control.c.

F.2.3 bool initialize (void)

Perform overall system initialization.

Reads configuration variables, and names of species files, initializes various hardware components.

Definition at line 182 of file control.c.

F.2.4 void instrumentsOn ( )

Turn on power to instruments, initialize control flags.

Definition at line 213 of file control.c.

F.2.5 void idleMode ( )

Go to idle mode.

Turns pumps off, lights off and selects bypass path at filter valve.

Definition at line 343 of file control.c.

F.2.6 ushort getADSample (int *chan*)

Read a sample from the AD converter (Burr-Brown ADS8344).

Full-scale reading is 65535, corresponding to 2.5 VDC.

Parameters

chan	is the AD channel to read; channel 1 is the battery voltage, channel 0 is the pressure downstream of the filter, channel 2 is the sample pump position (high-pressure units), channel 3 is the pressure upstream of the filter.

Returns

the average value from 50 samples, taken with a spacing of 2 ms (so takes 100 ms to complete)

Definition at line 356 of file control.c.

F.2.7 double getBatteryVoltage (void)

Read the battery voltage.

Returns

the battery voltage in volts.

Definition at line 373 of file control.c.

F.2.8 void handleOverpressure ( )

Attempt to clear overpressure condition caused by buble or dirt.

Runs the sample pump at slow speed, in bypass mode in effort to clear obstruction that is assumed to be causing over-pressure. Runs pump in 5 second bursts, with 5 seconds between bursts, stopping after two unsuccessful attempts, or when over-pressure is relieved. Throws OVERPRESSURE exception, if unable to clear over-pressure condition; otherwise simply returns

Definition at line 387 of file control.c.

F.2.9 void filterBlast (void)

Run the sample pump in short bursts in the reverse of its current direction.

Does 10 quick bursts in which the pump is turned on for 50 ms, then off for 20 ms. This is only done under manual control. There are not checks for over-pressure, but hopefully these short bursts will not damage filter.

Definition at line 439 of file control.c.

F.2.10 bool haveLeak (void)

Check for a leak.

Returns

true if a leak is detected and set the global Leak variable.

Definition at line 453 of file control.c.

F.2.11 void setOverride (bool *on*)

Turns on/off the Override flag.

This flag apparently controls power to the Valco board in high pressure units and the "serial mux" board in low pressure units. Includes 1 second delay when turning Override on.

Parameters

on	determines whether the override is turned on (when true) or off (when false).

Definition at line 469 of file control.c.

F.3 Variable Documentation

F.3.1 bool OPDinitialized = false

flag set when initialization has been completed.

Definition at line 80 of file control.c.

G psense.c File Reference

This file contains various functions for using the pressure sensors.

Functions

- double psense_getPressure ()

    Read the pressure at the filter.
- double psense_getPressure0 (void)

    Read the pressure sensor downstream of the membrane filter (dual sensor units).
- double psense_getPressure3 (void)

    Read the pressure sensor upstream of the membrane filter (dual sensor units).

- void psense_autoZero ()

    Calibrate the pressure sensors (dual pressure sensor units).

- double psense_getDepth ()

    Determine the approximate depth, from a pressure sensor reading.

- void psense_zeroDepthAdjust ()

    Calibrate the depth measurement for zero depth.

Variables

- int MaxFilterPressure = 30

    maximum allowed pressure across the filter (in psi)

- double RunDepth = 1.

    target operating depth for auv deployments

- double ShutDownDepth = .5

    minimum operating depth for auv deployments

- double DepthMeters

    most recently measured depth value

- double DepthCoefA

    first of two calibration coefficients used in depth measurement

- double DepthCoefB

    second of two calibration coefficients used in depth measurement

- bool Overpressure = false

    overpressure flag, set when overpressure condition is detected

- double Pressure0

    most recent pressure reading downstream of filter

- double Pressure3

    most recent pressure reading upstream of filter

- double M_Pressure0 = .001589481

    scaling parameter for downstream pressure sensor

- double B_Pressure0 = 0

    offset parameter for downstream pressure sensor
- double M_Pressure3 = .001589481

    scaling parameter for upstream pressure sensor
- double B_Pressure3 = 0

    offset parameter for upstream pressure sensor
- bool dualpressure

    flag used to indicate units with dual pressure sensors
- double FilterPressure

    most recently measured value of filter pressure
- int Status

    vector of status bits, set by digitalStatus
- enum Vehicle ActiveVehicle

    specifies deployment context

G.1 Detailed Description

This file contains various functions for using the pressure sensors.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file psense.c.

G.2 Function Documentation

G.2.1 double psense_getPressure (void)

Read the pressure at the filter.

Pressure value is saved in the global FilterPressure variable. For units with dual pressure sensors, use the difference between the pressure readings upstream of the filter and downstream; in units with a single pressure sensor, just uses the pressure value. If the pressure exceeds the MaxFilterPressure threshold, sets the global Overpressure flag also.

For units with single sensor, pressure computed directly from AD channel. Most of these units use 12V power supply, producing a 120mV signal at 30 psid. This is amplified by a factor of 20, giving 2.4 volts. The max reading on the AD channel is 65,535, corresponding to 2.5 volts, giving a scaling constant of 30/((2.4/2.5)∗65535) or .000477. The BSOP uses 10V supply voltage, yielding larger constant of .000572. These units use an Omega px-26 sensor.

For dual pressure sensor units, the supply voltage is 12v, producing a 240mV signal at 100 psi. This is then amplified by factor of 10 to get 2.4V. This yields scaling factor of 100/((2.4/2.5)∗65535) or .001589. The global constants M_Pressure0 and M_Pressure3 are set to this value and used when converting AD readings to psi. These units use two Micron MP50A 100-G sensors.

Returns

the pressure value

Definition at line 106 of file psense.c.

G.2.2 double psense_getPressure0 (void)

Read the pressure sensor downstream of the membrane filter (dual sensor units).

Returns

the pressure value in psi

Definition at line 144 of file psense.c.

G.2.3 double psense_getPressure3 (void)

Read the pressure sensor upstream of the membrane filter (dual sensor units).

Returns

the pressure value in psi

Definition at line 153 of file psense.c.

G.2.4 void psense_autoZero ( )

Calibrate the pressure sensors (dual pressure sensor units).

Sets offset values for each of the pressure sensors, where the pressure is assumed to be zero when the sample pump is turned off and the filter valve is in the bypass position.

Definition at line 163 of file psense.c.

G.2.5 double psense_getDepth ( )

Determine the approximate depth, from a pressure sensor reading.

Uses the sensor downstream of the filter valve, so is only valid in units with dual pressure sensors.

Definition at line 181 of file psense.c.

G.2.6 void psense_zeroDepthAdjust ( )

Calibrate the depth measurement for zero depth.

Should be called only when the OPD is at the surface.

Definition at line 189 of file psense.c.

H pumps.c File Reference

This file contains various functions for controlling the OPD's pumps.

Functions

- float flowRate (int speed)

  Calculate flow rate associated with a given pump speed.
- void setPumpSpeeds (void)

  Set the speeds of the sample pump and the cdom pump.
- void setPumpSpeedsToValues (short sampleSpeed, short cdomSpeed)

  Set the speeds of the sample pump and the cdom pump.
- void cpumpOnOff (bool on)

  Turn cdom reference pump on/off.
- void spumpOn ()

  Turn sample pump on.
- void spumpOff ()

  Turn off the sample pump.
- int adjustSpumpSpeed (int target, int currSpeed, int minSpeed)

  Adjust the sample pump speed to maintain target pressure.
- void sampleCdomByVol (void)

  Collect a cdom sample by volume.
- void pumpCreep (bool enable)

  Run sample pump at quarter speed.
- void calPosSensor ()

  Calibrate sample pump position sensor.
- void fullSpeedToEnd (void)

  Run sample pump at full speed until an end-of-travel condition is detected.
- bool getEndOfTravel (void)

  Check if sample pump at end-of-travel condition.
- void spumpGetPosition ()

  Determine the current position of the sample pump.
- void spumpGoto (ushort pos)

  Move sample pump syringes to a specified position.

- void spumpReverse (void)

    Reverse sample ports used by sample pump.
- void sameSpeedToEnd (void)

    Run sample pump to the end-of-travel range.
- void spumpGotoNearEnd (void)

    Reposition syringes.
- void sampleCdom ()

    Collect a Cdom sample while avoiding over-pressure.
- void cpumpCal (void)

    Calibrate the CDOM reference pump interactively.

Variables

- int SamplePumpSpeed = 127

    speed at which run the sample pump (range is -127 to +127)
- int CdomRefPumpSpeed = 200

    speed at which run the cdom reference pump (range is 0 to 255)
- unsigned int PosSensorMax

    syringe pump sensor value at max end of travel
- unsigned int PosSensorMin

    syringe pump sensor value at min end of travel
- unsigned int SamplePumpPosition

    current position of sample pump (for high pres.
- unsigned short CdomStrokePos = 75

    sample pump position corresponding to completed cdom sample
- bool pumpCreepEnable = false

    flag used to enable slow pumping during disc spectrum acquisition.
- bool PurgeBubbles = true

    flag used to control purging of bubbles from syringes when reversing sample pump

- unsigned int CdomRefPumpCal

  Cdom reference pump calibration parameter (for determing volume pumped)

- bool CdomPump

  flag used to turn on/off power to the cdom reference pump

- bool SamplePump

  flag used to turn on/off power to the sample pump

- jmp_buf exceptionContext

  setjmp context used to impelment exception handling

- bool FilterSelected

  flag set by digitalStatus when filterValve set to allow fluid to flow through filter

- bool BypassSelected

  flag set by digitalStatus when filterValve set to send fluid through bypass path (around filter)

- bool Override

  flag used to turn on/off power to the port valves

- double FilterPressure

  most recently measured value of filter pressure

- int CycleTime

  time between succesive cycles (in minutes)

- bool encoder

  flag used to indicate units with valco encoders

- int Status

  vector of status bits, set by digitalStatus

- bool EndOfTravel

  flag set by digitalStatus to indicate that syringe pump plunger is at one end or the other of its travel range

- bool Overpressure

  overpressure flag, set when overpressure condition is detected

- enum Vehicle ActiveVehicle

specifies deployment context

- bool enableSampling

    flag controlling sampling, may be disabled by <stop> command

- double CdomVolume

    volume of filtered seawater to pump during CDOM phase (in milliliters)

H.1 Detailed Description

This file contains various functions for controlling the OPD's pumps.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file pumps.c.

H.2 Function Documentation

H.2.1 float flowRate (int *speed*)

Calculate flow rate associated with a given pump speed.

This assumes we're pumping through the filter. Note, rates are non-linear: (a∗X^N)+ BX + C Note: THE BELOW COEFFICIENTS AND EXPONENT requires CALIBRATING MOORING pump controller POT. R9 to deliver +/-2mL/min. at pump speed +/-127 A-D units.

Parameters

speed	is a sample pump speed

Returns

the corresponding flow rate in ml/minute

Definition at line 329 of file pumps.c.

H.2.2 void setPumpSpeeds (void)

Set the speeds of the sample pump and the cdom pump.

The global variables SamplePumpSpeed and CdomRefPumpSpeed specify the desired pump speeds. This function sets the speeds in the physical devices. The sample pump values are limited to the range [-127,+127]. The cdom pump values are also limited to [-127,+127] in the glider deployment, and [0,255] in other cases

Definition at line 89 of file pumps.c.

H.2.3 void setPumpSpeedsToValues (short *sampleSpeed*, short *cdomSpeed*)

Set the speeds of the sample pump and the cdom pump.

Parameters

sampleSpeed	is the desired speed of the sample pump; the sample pump speeds are limited to the range [-127,+127].
cdomSpeed	is the desired speed of the cdom pump; the cdom pump values are limited to [-127,+127]

Definition at line 99 of file pumps.c.

H.2.4 void cpumpOnOff (bool *on*)

Turn cdom reference pump on/off.

Parameters

on	determines if the pump is turned on or off (true for on, false for off)

Definition at line 132 of file pumps.c.

H.2.5 void spumpOn ( )

Turn sample pump on.

Pump speed is ramped up to target speed over 100 ms. Duration: 1.4 seconds, largely due to 1 second delay for setting Override flag

Definition at line 144 of file pumps.c.

H.2.6 int adjustSpumpSpeed (int *target*, int *currSpeed*, int *minSpeed*)

Adjust the sample pump speed to maintain target pressure.

The pump is assumed to be running initially. The pump is left running on return with pump speed in global variable SamplePumpSpeed. May throw PRESSURE_EXCEPTION.

Parameters

target	is the desired target pressure
currSpeed	is the current pump speed; make adjustments from here
minSpeed	is a miniumum acceptable pump speed

Returns

the approximate volume pumped during the speed adjustment or -1 if over-pressure condition is detected.

Definition at line 272 of file pumps.c.

H.2.7 void sampleCdomByVol (void)

Collect a cdom sample by volume.

Collects a filtered sample of sea water (cdom sample), adjusting the sample pump speed as needed to avoid over-pressure and continuing until the estimated amount of water pumped reaches a limit. For use in low-pressure units only. Assumes sample pump is already running.

Notes from earlier version Adaptive control of SamplePumpSpeed while filtering to stay at a target FilterPressure - ARH 18 Oct 2010 Convert to absolute value of speed, then correct for direction of travel at the end. Change the pump's actual speed for use while filtering, but return SamplePumpSpeed to its original value and direction before returning from this function, so that it's fast again for Bypass parts of the cycle.

For Mooring, pumps & checks FilterPressure every 10 sec, adjusting it to get PressureTarget, calculates incremental flow volume, repeating until 2mL of CDOM filtrate has been pumped

For glider and other cases, checks FilterPressure once, adjusting it to get PressureTarget, then breaks back to driver.c to control volume via PosSensor

13 May 2011: to keep using the last adjusted speed, currSpeed[sc], at the beginning of CDOM for subsequent cycles: set sc=0 at boot, test if it is ==0 at the begining of this function. If it is==0, set it to sps as we do now. If not, then use it as it is, i.e do not change sc.

Definition at line 222 of file pumps.c.

H.2.8 void pumpCreep (bool *enable*)

Run sample pump at quarter speed.

Parameters

enable	determinees whether the pump is turned on or off; if enable is true, pump is turned on and the sample pump is saved; if enable is false, pump is turned off and the original sample pump speed is restored (into the global SamplePumpSpeed variable)

Definition at line 345 of file pumps.c.

H.2.9 void calPosSensor ( )

Calibrate sample pump position sensor.

Only for use in high-pressure units. Runs the pump in both directions until end-of-travel sensor is activated. Record points where sensor activated in global variables PosSensorMin, PosSensorMax. Configuration variables are written out to save these new values. Sample pump is left running on return.

Definition at line 375 of file pumps.c.

H.2.10 void fullSpeedToEnd (void)

Run sample pump at full speed until an end-of-travel condition is detected.

For use in high-pressure units only. Should only be used with bypass selected. Pump is assumed to be running already.

Definition at line 405 of file pumps.c.

H.2.11 bool getEndOfTravel (void)

Check if sample pump at end-of-travel condition.

For use in high-pressure units only.

Returns

true if pump is at end-of-travel, else false

Definition at line 446 of file pumps.c.

H.2.12 void spumpGetPosition ( )

Determine the current position of the sample pump.

For use in high-pressure units only. Sets the global variable SamplePump-Position.

Definition at line 452 of file pumps.c.

H.2.13 void spumpGoto (ushort *pos*)

Move sample pump syringes to a specified position.

For use in high-pressure units only. Pump is assumed to already be moving in the appropriate direction.

Parameters

pos	is the desired position

Run to A SPECIFIED SYRINGE PUMP POSITION for up to 120 seconds, assumes already moving before entering this function 22 OCT 2010 ARH INCREASED TO 200 SECONDS TO ALLOW SLOW SPEED PUMPING TO SUCCEED. based on reversesamplepump: if samplepumpspeed is >0 we are moving in the B direction, and the SamplePumpPosition values are decrementing In the B direction case, then, we want to go to possensormin+[(max-min)∗(100-CdomStrokePos)∗.01]; while in the A direction, samplepumpspeed is <0, go to possensormin+[(max-min)∗(CdomStrokePos)∗.01]

Definition at line 482 of file pumps.c.

H.2.14 void spumpReverse (void)

Reverse sample ports used by sample pump.

The OPD is equipped with two entry/exit ports for sampling sea water. Each port has a filter, which keeps out large particulates when it's acting as an entry port. To keep these filters from getting clogged, the software alternates the uses of the entry/exit ports. So on one cycle, a given port is used to bring in sea water, while in the next cycle, it is used to expel sea water. For low pressure units, this reversal is accomplished by simply changing valve settings. For high pressure units, which use syringe pumps, it also requires reversing the direction in which the pumps syringes travel. The sign of the pump speed determines the direction in which the pump travels.

Definition at line 574 of file pumps.c.

H.2.15 void sameSpeedToEnd (void)

Run sample pump to the end-of-travel range.

For use in high-pressure units only. Assumes pump is already on. Allows up to 100 seconds to reach end of travel.

Definition at line 677 of file pumps.c.

H.2.16 void spumpGotoNearEnd (void)

Reposition syringes.

For use in high-pressure units only.

Definition at line 703 of file pumps.c.

H.2.17 void sampleCdom ( )

Collect a Cdom sample while avoiding over-pressure.

For use in high-pressure units only.

Definition at line 753 of file pumps.c.

H.3 Variable Documentation

H.3.1 unsigned int SamplePumpPosition

current position of sample pump (for high pres.

units with syringe pumps)

Definition at line 49 of file pumps.c.

H.3.2 bool pumpCreepEnable = false

flag used to enable slow pumping during disc spectrum acquisition.

Definition at line 55 of file pumps.c.

I spect.c File Reference

This file contains functions used to control the spectrometer.

Functions

- bool spect_adjustIntegrationTime ()

  Adjust the integration time for the spectrometer.
- bool spect_checkLights ()

  Verify that spectrometer lights are operational.
- short spect_getBoot ()

  Get the boot message from the spectrometer.
- short spect_getStoredString (short slot, unsigned char *ans, int lng)

  Read a stored string from the spectrometer.
- short spect_getWavePoly ()

  Retrieve the wavelength polynomial coefficients from the spectrometer.
- void spect_getDark (ushort *rawSpect, double *cookedSpect)

  Get a dark spectrometer trace and "cook" it.
- bool spect_getTrace (ushort *rawSpect, double *cookedSpect)

  Get a raw spectrometer trace and "cook" it (perform pre-processing steps).
- bool spect_getSpect (ushort *spect)

  Read spectrum from spectrometer, retrying in the event of failure.

- bool spect_getSpectQuiet (ushort *spect)

  Read spectrum from spectrometer.
- void spect_setBaudRate (long rate)

  Set the spectrometer's baud rate.
- short spect_setScansToAdd (short n)

  Set spectrometers "scansToAdd" parameter.
- short spect_setCompressionMode (short mode)

  Set the compression mode used for retrieving spectra.
- short spect_setIntegrationTime (int inttime)

  Set the spectrometer integration time parameter.
- void spect_showMaxTrace (ushort *rawSpect)

  Print the largest pixel value in a raw spectrum.
- short spect_openUART ()

  Open the channel used to communicate with the spectrometer.
- void spect_closeUART ()

  Close the channel used to communicate with the spectrometer.
- long spect_writeUART (char *buf, long n)

  Write to spectrometer.
- short spect_setup ()

  Initialize the spectrometer.
- short spect_shutDown ()

  Shut down the spectometer.
- short spect_reset (void)

  Reset the spectrometer, by cycling power.
- void spect_shutterClosed ()

  Close shutter for light source.
- void spect_shutterOpen ()

  Open shutter for light source.
- void spect_lightsOff ()

  Turn off spectrometer lights.

- void spect_lightsOn ()

    Turn on spectrometer lights.

- short spect_getAck ()

    Attempt to read an acknowledgement from the spectrometer.

- int spect_getIntegrationTime ()

    Retrieve the current integration time from the spectrometer.

- short spect_getSerialNumber ()

    Read the serial number from the spectrometer.

- void spect_decompress (char spectBuf[ ], unsigned short rawSpect[ ], int n)

    Decompress trace data.

- unsigned long spect_readUART (unsigned char *buf, long maxBytes)

    Read data from the spectrometer.

- unsigned long spect_readUARTTrace (unsigned char *buf, long maxBytes)

    Read a spectrometer trace.

- unsigned long spect_fillBuffer (unsigned char *buf, long maxBytes)

    Read spectrum data and return in a buffer.

Variables

- unsigned int integrationTime = 100

    spectrometer integration time parameter

- double WaveGuideLength = 0.28

    length of spectrometer's waveguide

- double WavePoly [4] = { 179.6747106, 0.374580227, -1.46066e-05, -1.34397e-09 }

    wavelength polynomial, true value read from the spectrometer

- TUPort * spectPort

    uart port for communicating with spectrometer

- bool compression = true

  flag controlling use of compression

- long PixelMax = 0

  largest light intensity value in most recently acquired spectrum

- long Railed = 60000

  max threshold for validity of spectrum

- long SpectBaudRate = 9600

  baud rate used to communicate with spectrometer

- unsigned char SerialNumber [30]

  serial number used to identify spectrometer

- unsigned int CurrentScansToAdd = 15

  specifies number of scans that spectrometer should make during spectrum acquisition; returned spectrum is the sum of those scans

- bool Power

  flag used to control overall power setting

- bool Shutter

  flag used to open/close the shutter for the light source

- bool Lights

  flag used to turn on/off the spectrometer lights

- bool lightDeuterium

  flag used to turn individually control the deuterium light

- bool lightTungsten

  flag used to turn individually control the tungsten light

- int Status

  vector of status bits, set by digitalStatus

- bool DoCdomRef

  flag to enable/disable CDOM reference phase

- int WaveInt [NMAX]

  used to speed up wavelength normalization calculations

I.1 Detailed Description

This file contains functions used to control the spectrometer.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file spect.c.

I.2 Function Documentation

I.2.1 bool spect_adjustIntegrationTime (void)

Adjust the integration time for the spectrometer.

The integration time is the length of time to keep the light source on when measuring a sample. If integration time is too long, the light sensors become saturated and we don't get useful data. Still, we want the readings for cdom-Ref to be near the high end of the scale in order to maximize the sensitivity.

Returns

true if integration time was successfully adjusted to produce maximum intensities in the target range, else false

Definition at line 439 of file spect.c.

I.2.2 bool spect_checkLights (void)

Verify that spectrometer lights are operational.

Checks the raw spectrum values of the deuterium lamp at 440 nm, and the the tungsten lamp at 580 nm. If the spectrum values with the lights on are not at least 200 units larger than the values reported for a dark spectrum, the test fails.

Returns

true if both the deuterium or tungsten lamp are working, else false.

Definition at line 507 of file spect.c.

I.2.3 short spect_getStoredString (short *slot*, unsigned char * *ans*, int *lng*)

Read a stored string from the spectrometer.

Parameters

slot	is the slot number associated with the desired string.
ans	is a pointer to character buffer in which result is returned
lng	is the length of the buffer

Returns

1 on success, else 0

Definition at line 263 of file spect.c.

I.2.4 short spect_getWavePoly ( )

Retrieve the wavelength polynomial coefficients from the spectrometer.

The retrieved coefficients are stored in the global WavePoly vector.

Definition at line 286 of file spect.c.

I.2.5 void spect_getDark (ushort * *rawSpect*, double * *cookedSpect*)

Get a dark spectrometer trace and "cook" it.

Preprocessing includes gaussian smoothing and wavelength normalization. Takes about 35 seconds.

Parameters

rawSpect	is a pointer to an array of OMAX ushorts in which the raw spectrum is returned.
cookedSpect	is a pointer to an array of NMAX doubles in which the processed (no longer raw) spectrum is returned.

Definition at line 560 of file spect.c.

I.2.6 bool spect_getTrace (ushort * *rawSpect*, double * *cookedSpect*)

Get a raw spectrometer trace and "cook" it (perform pre-processing steps).

Preprocessing includes gaussian smoothing and wavelength normalization. Takes about 35 seconds.

Parameters

rawSpect	is a pointer to an array of OMAX ushorts in which the raw spectrum is returned.
cooked-Spect	is a pointer to an array of NMAX doubles in which the processed (no longer raw) spectrum is returned.

Returns

true on success, false on failure

Definition at line 575 of file spect.c.

I.2.7 bool spect_getSpect (ushort * *spect*)

Read spectrum from spectrometer, retrying in the event of failure.

Duration: 20 seconds, 40 retry is needed

Parameters

spect	is a pointer to an array of OMAX unsigned shorts in which a raw spectrum is returned.

Returns

1 if operation is successful, else 0

Definition at line 596 of file spect.c.

I.2.8 bool spect_getSpectQuiet (ushort * *spect*)

Read spectrum from spectrometer.

Duration: 20 seconds

Parameters

spect	is a pointer to an array of OMAX unsigned shorts in which a raw spectrum is returned.

Returns

true on success, else false

Definition at line 620 of file spect.c.

I.2.9 void spect_setBaudRate (long *rate*)

Set the spectrometer's baud rate.

Parameters

rate	specifies the desired new baud rate

Definition at line 302 of file spect.c.

I.2.10 short spect_setScansToAdd (short *n*)

Set spectrometers "scansToAdd" parameter.

Parameters

n	is value to set the parameter to

Definition at line 414 of file spect.c.

I.2.11 short spect_setCompressionMode (short *mode*)

Set the compression mode used for retrieving spectra.

See the documentation for spect_decompress for a description of the delta-coding method used.

Definition at line 363 of file spect.c.

I.2.12 short spect_setIntegrationTime (int *inttime*)

Set the spectrometer integration time parameter.

This is the time that the light source is turned on when a spectrum is being acquired. Typical values around 100-200. Increases as lightguide becomes soiled over time.

Parameters

inttime	is the desired integration time in ms

Definition at line 383 of file spect.c.

I.2.13 void spect_showMaxTrace (ushort * *rawSpect*)

Print the largest pixel value in a raw spectrum.

Parameters

rawSpect	is a pointer to an array of OMAX ushorts containing a raw spectrum.

Definition at line 698 of file spect.c.

I.2.14 long spect_writeUART (char * *buf*, long *n*)

Write to spectrometer.

Parameters

buf	is a pointer to a character buffer
n	is the number of bytes from buf to be written

Definition at line 867 of file spect.c.

I.2.15 short spect_shutDown ( )

Shut down the spectometer.

Definition at line 122 of file spect.c.

I.2.16 short spect_reset (void)

Reset the spectrometer, by cycling power.

Returns

1 if spectrometer was successfully reset, else 0

Definition at line 127 of file spect.c.

I.2.17 void spect_shutterClosed ( )

Close shutter for light source.

Definition at line 141 of file spect.c.

I.2.18 void spect_shutterOpen ( )

Open shutter for light source.

Definition at line 149 of file spect.c.

I.2.19 void spect_lightsOn ( )

Turn on spectrometer lights.

The individual control flags for the deuterium and tungsten lamps must also be set to actually turn the lights on.

Definition at line 167 of file spect.c.

I.2.20 int spect_getIntegrationTime (void)

Retrieve the current integration time from the spectrometer.

Returns

the integration time

Definition at line 230 of file spect.c.

I.2.21 short spect_getSerialNumber ( )

Read the serial number from the spectrometer.

The resulting serial number is stored in the global string SerialNumber.

Returns

1 on success, else 0.

Definition at line 252 of file spect.c.

I.2.22 void spect_decompress (char *spectBuf[]*, unsigned short *rawSpect[]*, int *n*)

Decompress trace data.

The spectrometer uses a simple delta compression scheme. Most bytes in spectBuf specify the difference between the value of the "previous" pixel and the "current" pixel. A value of -128 serves as an "escape", indicating that the next two bytes specify the absolute value of the current pixel. The escape mechanism is always used for the first pixel and whenever the absolute difference between two successive pixels is larger than 127. Using this method, the spectrum can be represented using as few as 2050 bytes. In the worst-case, it requires 3∗2048. Duration: 100 ms

Parameters

spectBuf	is an array of characters containing data obtained from the spectrometer
rawSpect	is an array of OMAX ushorts in which the results are returned
n	is the number of valid bytes in spectBuf

Definition at line 667 of file spect.c.

I.2.23 unsigned long spect_readUART (unsigned char * *buf*, long *maxBytes*)

Read data from the spectrometer.

Parameters

buf	is a pointer to a character buffer
maxBytes	is the maximum number of bytes to read into buf

Returns

the number of characters read

Definition at line 741 of file spect.c.

I.2.24 unsigned long spect_readUARTTrace (unsigned char * *buf*, long *maxBytes*)

Read a spectrometer trace.

Parameters

buf	points to a buffer in which trace is returned; it must start with an odd address so that two byte values that start at odd distances from start of buffer are aligned properly.
maxBytes	is an upper bound on the number of characters that can be read into buf

Definition at line 791 of file spect.c.

I.2.25 unsigned long spect_fillBuffer (unsigned char * *buf*, long *maxBytes*)

Read spectrum data and return in a buffer.

Parameters

buf	is a pointer to a buffer in which data is returned
maxBytes	is the size of the buffer

Returns

the number of bytes being returned

Definition at line 821 of file spect.c.

J fvalve.c File Reference

This file contains various functions for controlling the filter valve.

Functions

- void fvalve_runMotor (bool on)

  Turns the filter valve motor on or off.
- void fvalve_setFilter (bool bypass, bool filter)

  Set the valve associated with the filter.
- void fvalve_selectBypass ()

  Set filter valves to send flow through the bypass path.
- void fvalve_selectClosed ()

  Close both filter valves to block all flow.
- void fvalve_selectFilter ()

  Set filter valves to send flow through the filter.

Variables

- jmp_buf exceptionContext

  setjmp context used to impelment exception handling
- bool Filter

  flag used to control the filter valve
- bool BypassSelected

  flag set by digitalStatus when filterValve set to send fluid through bypass path (around filter)
- bool FilterSelected

  flag set by digitalStatus when filterValve set to allow fluid to flow through filter
- bool Status

  vector of status bits, set by digitalStatus
- bool Overpressure

  overpressure flag, set when overpressure condition is detected

J.1 Detailed Description

This file contains various functions for controlling the filter valve.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file fvalve.c.

J.2 Function Documentation

J.2.1 void fvalve_runMotor (bool *on*)

Turns the filter valve motor on or off.

Parameters

on	determines whether the motor is turned on (when true) or off (when false).

Definition at line 264 of file fvalve.c.

J.2.2 void fvalve_setFilter (bool *bypass*, bool *filter*)

Set the valve associated with the filter.

The filter valve is an Instech pinch valve that has a small motor that turns a cam. The position of the cam determines which of the two tubes are pinched, or if both tubes are pinched.

Parameters

bypass	specifies that the valve be placed in the bypass position (no filtering)
filter	specifies that the valve be placed in the filter position (so that seawater samples are filtered, to enable cdom measurement) If both arguments are false, the valve is closed so that seawater flows through neither the bypass path nor the filter. If both arguments are true, nothing is done.

Definition at line 80 of file fvalve.c.

K valco.c File Reference

This file contains functions used to control the OPD's valves.

Functions

- unsigned long ReadValco (unsigned char *buf, long MaxBytes)

  Read state of Valco valve.
- void GetGPS ()

  Get GPS fix from Valco board.
- short OpenValco ()

  Open UART channel to communicate with Valco (or serial mux) board.
- void CloseValco ()

  Close UART used to communicate with Valco board.
- short SetValve (uchar pos)

  Set the valve position.
- short ValcoCal ()

  Calibrate the valco valves.

Variables

- short ValcoBaudRate = 9600

  baud rate used for communicating with valco board
- int valcoLoop = 6

  number of times to attempt to reverse selected sampling ports following a failed attempt to set the valve position
- TUPort * vprt

  pointer to UART port structure used for communicating with Valco board
- bool enableSampling

  flag controlling sampling, may be disabled by <stop> command
- bool encoder

  flag used to indicate units with valco encoders
- double BatteryVoltage

  most recently measured value of battery voltage
- double FilterPressure

  most recently measured value of filter pressure
- int Status

  vector of status bits, set by digitalStatus
- enum Vehicle ActiveVehicle

  specifies deployment context

K.1 Detailed Description

This file contains functions used to control the OPD's valves. In high pressure units, valves are controlled by Valco control board. In low pressure units, simpler valves are controlled by serial mux board. Communication is done using a UART channel. Same protocol is used to communicate with both types of board.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file valco.c.

K.2 Function Documentation

K.2.1 unsigned long ReadValco (unsigned char * *buf*, long *MaxBytes*)

Read state of Valco valve.

Parameters

buf	is a buffer in which result is returned.
MaxBytes	is the length of buf

Returns

number of characters returned in buf

Definition at line 136 of file valco.c.

K.2.2 void GetGPS ( )

Get GPS fix from Valco board.

GPS fix is saved in global string GPS. Currently, the string "No Fix" is written.

Definition at line 53 of file valco.c.

K.2.3 short SetValve (uchar *pos*)

Set the valve position.

Sets the valves controlling which port the OPD uses to bring in a new sea water sample, and which is used to expel the previous sample.

Parameters

pos	specifies whether the valve is to be set in position 'A' or 'B'.

Definition at line 206 of file valco.c.

L analysis.c File Reference

This file contains the data analysis functions used to analyze the acquired spectra.

Functions

- void anlys_setupWave ()

  Precompute vectors used to speed-up spectrum normalization.
- void ShowDblTrace (double *cookedSpect)

  Print a sample of cooked spectrum on console Displays every 50-th value in the spectrum.
- void anlys_discAbsorb (double *cdom, double *disc, double *absorb)

  Compute absorbance spectrum for a disc sample.
- double anlys_corrMetric (double *f1, double *f2)

  Compute correlation metric (similarity) for two fourth derivative spectra.
- void anlys_cookSpect (ushort *rawSpect, double *cookedSpect)

  Produce a cooked spectrum from a raw spectrum.
- void anlys_gaussFilter (ushort raw[ ], ushort smooth[ ])

  Apply a smoothing filter to a raw spectrometer vector.
- void anlys_normalize (unsigned short *smooth, double *cooked)

  Convert spectrum to use integral wavelengths in range WAVEMINIMUM-WAVEMAXIMUM (normally 350-799) nm.
- void anlys_subDark (double *cooked1, double *cooked2)

  Subtract one cooked spectrum from another.
- void anlys_cdomAbsorption (double *cdomRef, double *cdom)

  Calculate cdom absorption parameters.
- void anlys_4dSpect (double absorb[ ], double fourth[ ])

  Compute approximate fourth derivative spectrum.
- void check4badSpectra (double *cdomRef, double *cdom, double *disc)

  Check for spectra that are inconsistent and probably incorrect.
- void anlys_discSpecies (double *cdomRef, double *cdom, double *disc)

  Calculate correllation metric for disc sample against a set of reference spectra.
- void anlys_getSpeciesFileNames ()

  Read names of species files from species directory.

Variables

- int FilterSize = 47

  full window size for gaussian smoothing filter
- double FilterSigma = 12.0

  standard deviation for gaussian smoothing filter
- unsigned int CorrWaveMin = 400

  low end of the range of wavelengths over which the correlation metric is computed
- unsigned int CorrWaveMax = 700

  high end of the range of wavelengths over which the correlation metric is computed
- unsigned int AbWaveMin = 390

  low end of range of wavelengths used to fit shifted absorption curve to exponential function
- unsigned int AbWaveMax = 490

  high end of range of wavelengths used to fit shifted absorption curve to exponential function
- unsigned int AbWaveMid = 440

  midpoint of range of wavelengths used to fit shifted absorption curve to exponential function
- double AbsorbanceA

  characteristic parameters of the cdom absorption spectrum
- int nSpecies

  number of species files
- char Species [MaxSpecies][20]

  names of species files containing reference spectra
- double CorrResults [MaxSpecies]

  correlation results from disc against each reference spectrum
- int WaveInt [NMAX]

  used to speed up wavelength normalization calculations

- double WaveFl [NMAX]

  used to speed up wavelength normalization calculations
- int Status

  vector of status bits, set by digitalStatus
- bool DoCdomRef

  flag to enable/disable CDOM reference phase
- bool DoCdom

  flag to enable/disable CDOM phase
- bool DoDisc

  flag to enable/disable discriminant phase
- double WavePoly [4]

  wavelength polynomial, true value read from the spectrometer
- double WaveGuideLength

  length of spectrometer's waveguide
- double coef4 [57]

  coefficients used to compute approximate 4-th derivative of discriminant absorption spectra

L.1 Detailed Description

This file contains the data analysis functions used to analyze the acquired spectra.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file analysis.c.

L.2 Function Documentation

L.2.1 void anlys_setupWave ( )

Precompute vectors used to speed-up spectrum normalization.

Computes the global vectors WaveInt and WaveFl, where WaveInt[i] is the index of the smallest wavelength in the spectrometer's spectrum that is larger than WAVEMINIMUM+i (normally 350+i). WaveFl[i] is is used to interpolate data values for integral wavelengths that fall between two spectrometer wavelengths. If w1 and w2 are the closest spectrometer wavelengths with w1 <= WAVEMINIMUM+i < w2, then WaveFl[i] = (WAVEMINIMUM+i - w1) / (w2 - w1).

Definition at line 194 of file analysis.c.

L.2.2 void ShowDblTrace (double * *cookedSpect*)

Print a sample of cooked spectrum on console Displays every 50-th value in the spectrum.

Parameters

cookedSpect	is a pointer to a cooked spectrum

Definition at line 222 of file analysis.c.

L.2.3 void anlys_discAbsorb (double * *cdom,* double * *disc,* double * *absorb*)

Compute absorbance spectrum for a disc sample.

Parameters

cdom	is a cooked cdom spectrum with dark subtracted
disc	is a cooked disc spectrum with dark subtracted
absorb	is an array of NMAX doubles in which the absorbance spectrum is returned

Definition at line 488 of file analysis.c.

L.2.4 double anlys_corrMetric (double * *f1*, double * *f2*)

Compute correlation metric (similarity) for two fourth derivative spectra.

Computation time about 100 ms.

Parameters

f1	is a fourth derivative spectrum
f2	is a second fourth derivative spectrum

Definition at line 454 of file analysis.c.

L.2.5 void anlys_cookSpect (ushort * *rawSpect*, double * *cookedSpect*)

Produce a cooked spectrum from a raw spectrum.

Parameters

rawSpect	is a pointer to a raw spectrum (OMAX unsigned shorts)
cooked-Spect	is a pointer to a cooked spectrum in which the result is returned (NMAX doubles).

Definition at line 93 of file analysis.c.

L.2.6 void anlys_gaussFilter (ushort *raw[]*, ushort *smooth[]*)

Apply a smoothing filter to a raw spectrometer vector.

Replaces each data point of the input vector with a weighted sum of the neighboring values; the weighting function is a Gaussian curve. The number of neighbors in the weighted sum is specified by the global parameter FilterSize and the sigma value for the Gaussian is specified by the global parameter FilterSigma. Note: this routine is doing floating point calculations on a processor with no floating point hardware. The inner loop of the filter calculation requires over 150 us and the entire function takes about 17 seconds.

Parameters

raw	is a vector of input values of length OMAX
smooth	is a vector used for output values (also of length OMAX)

Definition at line 115 of file analysis.c.

L.2.7 void anlys_normalize (unsigned short ∗ *smooth*, double ∗ *cooked*)

Convert spectrum to use integral wavelengths in range WAVEMINIMUM--WAVEMAXIMUM (normally 350-799) nm.

For each integral wavelength, find closest wavelengths in the spectrometer's wavelength set and interpolate the data value.

Parameters

smooth	is a smoothed spectrum using spectrometer's wavelengths
cooked	is resulting output spectrum using normalized wavelengths

Definition at line 171 of file analysis.c.

L.2.8 void anlys_subDark (double * *cooked1*, double * *cooked2*)

Subtract one cooked spectrum from another.

Parameters

cooked1	is an array of NMAX doubles (a cooked spectrum)
cooked2	is another array of NMAX doubles, which is to be subtracted from cooked1; result returned in cooked1

Definition at line 235 of file analysis.c.

L.2.9 void anlys_cdomAbsorption (double * *cdomRef*, double * *cdom*)

Calculate cdom absorption parameters.

This function calculates the absorption spectrum for the most recent cdom sample (computed from global vectors Cdom and CdomRef). It then reduces the spectrum by subtracting the average value in the 690-709 nm range from the entire spectrum. It then fits an exponential curve to the portion of the spectrum from 390-489 nm and computes the value of this curve at 440 nm and its "slope parameter". These values are saved in AbsorbanceA and AbsorbanceB and printed to the log file. They also get printed to the STT file.

Parameters

cdomRef	is the cooked cdom reference spectrum with dark spectrum subtracted
cdom	is the cooked cdom spectrum with dark spectrum subtracted

Definition at line 254 of file analysis.c.

L.2.10 void anlys_4dSpect (double *absorb[]*, double *fourth[]*)

Compute approximate fourth derivative spectrum.

Parameters

absorb	is an absorbance spectrum
fourth	is a vector in which fourth derivative spectrum is computed

For each data point in absorb, computes a 7-th order polynomial approximation of the surrounding data points, then computes fourth derivative of this polynomial; exploits the fact that wavelengths are evenly spaced to substantially reduce the computation time

Inner loop takes about 150 us, entire function about 3 seconds.

Definition at line 354 of file analysis.c.

L.2.11 void check4badSpectra (double * *cdomRef*, double * *cdom*, double * *disc*)

Check for spectra that are inconsistent and probably incorrect.

If the spectra appear to be faulty, set the CorruptData bit in the global Status word.

Parameters

cdomRef	is pointer to a cooked cdomRef spectrum
cdom	is pointer to a cooked cdom spectrum
disc	is pointer to a cooked disc spectrum

Definition at line 435 of file analysis.c.

L.2.12 void anlys_discSpecies (double * *cdomRef*, double * *cdom*, double * *disc*)

Calculate correllation metric for disc sample against a set of reference spectra.

Specifically, the code computes the fourth derivative of a given absorbtance spectrum and compares this to the fourth derivative of one or more reference spectra read from files in the Species folder. Comparison results are saved in CorrResults and printed to the log file.

Parameters

cdomRef	is the cooked cdomRef spectrum (with spectrum subtracted)
cdom	is the cooked cdom spectrum (with dark subtracted)
disc	is the cooked discriminant spectrum (with dark subtracted)

Definition at line 384 of file analysis.c.

L.2.13 void anlys_getSpeciesFileNames ( )

Read names of species files from species directory.

Enumerates files in file system's species directory and save the file names in the global array of filenames, Species. If there is a file named "DINOKAR-B" (must be all caps), it is placed at the beginning of Species. Also sets the value of the global variable nSpecies, which specifies the number of files in the species directory.

Definition at line 506 of file analysis.c.

M misc.c File Reference

This file contains a variety of miscellaneous utility functions.

Functions

- void flushLine ()

  Read characters from stdin up to the end of the line.
- bool strComp (char *a, char *b)

  Species two strings for equality, ignoring case.
- bool isprefix (char *p, char *s)

  Determine if one string is a prefix of another.
- int readLine (FILE *f, char *buf, int n)

  Read a line from a file.
- char * dateTimeString ()

  Construct a string with the current date and time.
- void checkConsole ()

  Check for termination conditions and exit if necessary.
- void delayMseconds (unsigned long mSeconds)

  Delay for specified number of milliseconds.
- void delaySeconds (unsigned long seconds)

  Delay for specified number of seconds.
- void longDelay (unsigned long seconds)

  Delay for several seconds, while checking for console input.
- ulong elapsedTime ()

  Return the elapsed time, since first invocation.
- void LWCCFlush (int flushtime)

  Pump cdom reference fluid through the spectrometer.
- void cdomrefLitersUpdate (int speed, int pumpTime)

Update the volume of CDOM ref fluid remaining.

- void cdomrefLitersSave (void)

  Save the current estimated cdom reference fluid volume to the CDRFLEFT file.
- void filterBackFlush (void)

  Backflush filter interactively.
- void reportVersionEtc ()

  Print assorted configuration information to the console.

Variables

- jmp_buf exceptionContext

  setjmp context used to impelment exception handling
- int CdomRefPumpSpeed

  speed at which run the cdom reference pump (range is 0 to 255)
- int SamplePumpSpeed

  speed at which run the sample pump (range is -127 to +127)
- unsigned int CdomRefPumpCal

  Cdom reference pump calibration parameter (for determing volume pumped)
- unsigned char SerialNumber [30]

  serial number used to identify spectrometer
- double WavePoly [4]

  wavelength polynomial, true value read from the spectrometer
- enum Vehicle ActiveVehicle

  specifies deployment context

M.1 Detailed Description

This file contains a variety of miscellaneous utility functions.

Author

Mote Marine Lab - Ocean Technology Group

Date

2008-2016

Definition in file misc.c.

M.2 Function Documentation

M.2.1 void flushLine ( )

Read characters from stdin up to the end of the line.

Definition at line 38 of file misc.c.

M.2.2 bool strComp (char * *a*, char * *b*)

Species two strings for equality, ignoring case.

Parameters

a	is a pointer to a null-terminated string
b	is a pointer to a second null-terminated string

Returns

true if the two strings are equal, except for case

Definition at line 48 of file misc.c.

M.2.3 bool isprefix (char * *p*, char * *s*)

Determine if one string is a prefix of another.

Parameters

p	is a null-terminated character string
s	is another null-terminated character string

Returns

true if p is a prefix of s, else false

Definition at line 65 of file misc.c.

M.2.4 int readLine (FILE * *f*, char * *buf*, int *n*)

Read a line from a file.

Returns the next non-blank line from the input; carriage returns and line feeds are removed.

Parameters

f	is an open file
buf	is a pointer to a buffer in which the line is returned
n	is the size of buf; at most n characters are copied to buf (including the terminating EOS character); other characters in the current line are ignored.

Definition at line 82 of file misc.c.

M.2.5 char∗ dateTimeString ( )

Construct a string with the current date and time.

Returns

a pointer to an internal buffer containing the current date and time.

Definition at line 100 of file misc.c.

M.2.6 void checkConsole ( )

Check for termination conditions and exit if necessary.

Reads console input as a side effect. Performs cleanup before calling exit(). This function should be called about once per second to ensure that program terminates should a termination condition arise. Checks for leaks and for <stop> or <closedown> requests from the console.

Definition at line 118 of file misc.c.

M.2.7 void delayMseconds (unsigned long *mSeconds*)

Delay for specified number of milliseconds.

Parameters

mSeconds	is number of milliseconds to delay before returning

Definition at line 154 of file misc.c.

M.2.8 void delaySeconds (unsigned long *seconds*)

Delay for specified number of seconds.

Parameters

seconds	is number of seconds to delay before returning

Definition at line 161 of file misc.c.

M.2.9 void longDelay (unsigned long *seconds*)

Delay for several seconds, while checking for console input.

May throw a SUSPEND if <stop> command is received or an EXIT exception if a <closedown> command is received.

Parameters

seconds	specifies number of seconds to delay before returning

Definition at line 170 of file misc.c.

M.2.10 ulong elapsedTime ( )

Return the elapsed time, since first invocation.

Assumes that the real-time clock tick is 25 microseconds.

Returns

the time in milliseconds since the clock started; time value wraps around after about 4 million seconds

Definition at line 182 of file misc.c.

M.2.11 void LWCCFlush (int *flushtime*)

Pump cdom reference fluid through the spectrometer.

Parameters

flushtime	is the number of seconds to run the pump

Definition at line 212 of file misc.c.

M.2.12 void cdomrefLitersUpdate (int *speed*, int *pumpTime*)

Update the volume of CDOM ref fluid remaining.

Updates both the internal value and the value stored in the CDRFLEFT file. Should be called after any CdomRef pump operation.

Parameters

speed	is the cdom pump speed
pumpTime	is the number of seconds the pump was turned on

Definition at line 233 of file misc.c.

M.2.13 void reportVersionEtc ( )

Print assorted configuration information to the console.

Definition at line 312 of file misc.c.