Using the PHYSS Data Analysis Console

Jonathan Turner
Mote Marine Laboratory
Ocean Technology Group

August 28, 2019

Abstract

The Programmable Hyperspectral Seawater Scanner (PHYSS) is a next-generation marine spectrophotometer based on Mote Marine Laboratory's Optical Phytoplankton Detector (OPD). Like the OPD, the PHYSS performs automated sampling and analysis of seawater in order to detect the presence of various pigments associated with harmful algae blooms, such as *Karenia Brevis*. Unlike the OPD, the PHYSS can be programmed by users, allowing far greater flexibility in both the sampling procedure and the analysis of the resulting data. In addition, it supports an extended hardware configuration, which allows two chemical reagents to be mixed with seawater samples prior to optical spectrum acquisition. This report provides an overview of the PHYSS hardware and software and describes how use the PHYSS Data Analysis Console to access and analyze the data provided by deployed PHYSS units. [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

Since early 2017, the Mote Research Laboratory's Ocean Technology Group has been developing a next generation version of the Optical Phytoplankton Discriminator (OPD) [1, 2]. The new system, called the Programmable Hyperspectral Seawater Scanner (PHYSS) was designed to build on the established legacy of the OPD, while making several improvements.

- upgraded hardware components, including new pumps, valves and a more sensitive spectrophotometer (16 bits vs 12),
- an extended configuration that enables two separate reagents to be mixed with seawater samples in user-controlled proportions,
- a fully programmable data acquisition process, giving users more control over how data is collected,
- a web-based control interface, allowing remote users to operate the PHYSS without having to install special-purpose software on their computers,
- automatic transmission of complete hyperspectral data to a cloud-server, giving users access to full spectra, not just summary metrics,
- a web-based data analysis tool, enabling users to review raw data from all operating PHYSS units, to view processed versions of that data and to expand the set of processing options.

The data produced by operational PHYSS units is automatically uploaded to a cloud server, where it can be accessed by users, using a web-resident *Data Analysis Console*. This report explains how to use the console to view and anlalyze the data produced by the PHYSS. Section 2 and 3 provide essential background material. Specifically, Section 2 gives an overview of the PHYSS hardware and software, while Section 3 describes how data is produced by the PHYSS. Section 4 describes the basic use of the console and section 5 describes how the user can modify and extend the analyses implemented by the console in order to tailor it to individual requirements.

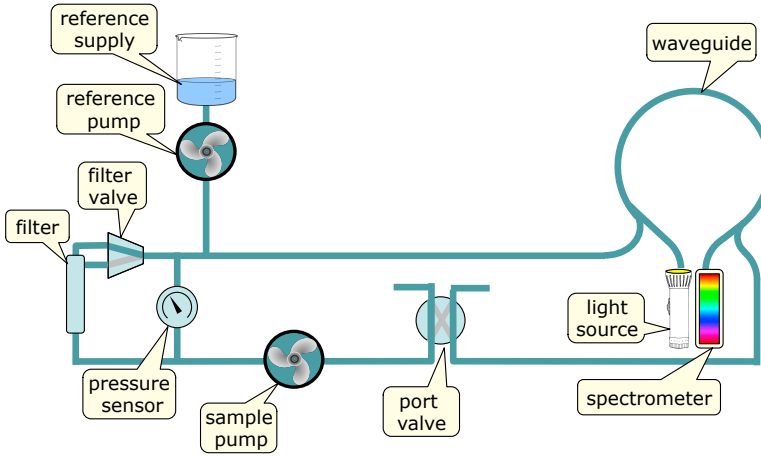

Figure 1 Basic hardware configuration

2 Overview of hardware and software

Figure 1 is a diagram showing the basic PHYSS hardware configuration. Seawater enters the system through one of two entry/exit ports and is pumped by a peristaltic *sample pump* through a filter and valve combination that allows either filtered or unfiltered seawater (the filter removes particles larger than 0.2 μm) to enter an optical waveguide, where it is illuminated by a light source to produce an optical spectrum in the visible light range (the spectrometer acquires 16 bit intensity values in 2048 bands from 190 to 880 nm). The configuration of the *port valves* is changed periodically to reverse the roles of the entry and exit ports; this prevents large particles in the seawater from accumulating on the screens that keep such particles out of the internal plumbing. A separate *reference pump* is used to pump a reference fluid into the waveguide (the reference fluid is typically seawater collected from far offshore). The spectrum of the reference fluid is compared to the spectrum of seawater samples during the data analysis process. A pressure sensor is used to monitor the pressure across the filter when collecting a filtered seawater sample, so as to avoid exceeding the transverse pressure limit on the filter

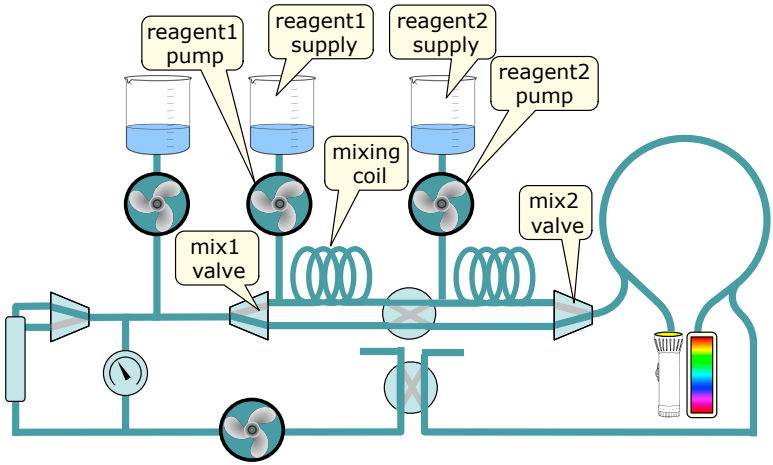

Figure 2 Extended hardware configuration

(30 psi). The PHYSS hardware is controlled by a *Beaglebone* processor [3], a 32 bit embedded processor that runs the Linux operating system.

Figure 2 shows the extended PHYSS configuration, which includes two reagent pumps with associated supply reservoirs, plus a system of valves and mixing coils that allow two distinct reagents to be mixed with seawater samples in user-specified proportions.

The major software components of the PHYSS are shown in Figure 3. The *Data Collector* is the component that operates the PHYSS hardware and implements the automated data collection process. Data collection is performed as a series of numbered *sample cycles* where the steps in each sample cycle are determined by a user-specified script. A *configuration file* is used to specify various operational parameters. The output of the data collection process is written to a raw data file.

The PHYSS is operated through a web-based control console. The console is implemented as a Javascript program that is downloaded from the PHYSS to a user's web browser and runs within the browser. Consequently, there

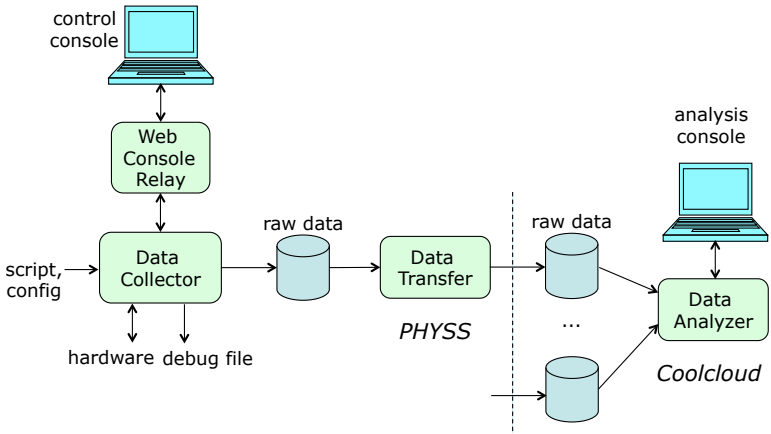

Figure 3 Software components

is no need for users to install software in order to use the PHYSS. A small *relay* component serves as an intermediary between the remote console and the data collector.

A separate *data transfer* component sends data to a remote cloud server, where it can be viewed using a *data analysis console*, another Javascript program that is downloaded from the cloud server and runs within the user's browser.

3 PHYSS Data Collection

The PHYSS data collection process is controlled by two files whose contents are specified by the PHYSS operator. The *configuration file* includes a variety of static configuration variables that are tailored to individual PHYSS units and deployment situations. The *script file* specifies the sequence of steps that are performed during each sample cycle. Copies of the configuration and script files are included with the data transferred from the PHYSS to

the cloud server and can be viewed through the data analyzer, so that users of the data can understand how the data collection was carried out.

An example of a PHYSS configuration file is shown below. The individual parameters are explained below.

```
# basic config file
serialNumber = 33
deploymentLabel = Sanibel Marker 2
gpsCoordinates = N26.4344/W81.9648

waveguideLength = 0.28
maxFilterPressure = 25
maxDepth = 20

hardwareConfig = BASIC
execMode = LAB
autoRun = -1
singleCycleMode = 0
portSwitching = 1
```

Config files may include comments, that are introduced with a pound sign '#' and extend to the end of the line.

The *serialNumber* parameter is a unique integer identifying a specific PHYSS unit. PHYSS serial numbers are restricted to values larger than 31. The older OPD units use smaller serial numbers.

The deployment label is a text label provided by the operator, which typically describes the location of the deployed unit.

The *gpsCoordinates* parameter specifies the GPS coordinates of a deployed PHYSS unit. The first value is the latitude in degrees (with four fractional

digits), the second is the longitude. The latitude includes a prefix 'N' or 'S' to indicate north or south. The longitude includes a prefix 'E' or 'W' to indicate east or west. This parameter is set by the operator and is included with the data sent to the cloud server.

The *waveguideLength* parameter specifies the length of the waveguide in meters. This value is used by the data analyzer to compute absorption spectra.

The *maxFilterPressure* parameter specifies the maximum safe operating pressure across the filter in pounds-per-square-inch (psi). This is typically set slightly below the true maximum in order to provide some safety margin.

The *maxDepth* parameter specifes the maximum safe operating depth for the PHYSS unit.

The *hardwareConfig* parameter specifies which of the two hardware configurations is used by the PHYSS unit. The allowed values are BASIC and TWO_REAGENTS.

The *execution mode* parameter determines what the data collector should do when the all of the sample cycles specified by the script have completed. In LAB mode, the data collector simply suspends script execution and returns to manual mode. In FIELD mode, it terminates and shuts down the PHYSS.

The *autoRun* parameter determines whether automated data collection is started automatically or not. A negative value (typically -1) causes the data collector to start in manual mode. It is up to the user to start automated data collection through the console. A value of 0 causes automated data collection to start automatically, when the data collector starts. A positive value is interpreted as a delay interval (in minutes); automated data collection begins after this delay interval has transpired.

The *singleCycleMode* parameter controls the use of single cycle mode. When single cycle mode is turned on, the data collector shuts down the PHYSS

after each sample cycle. An external reset must be applied to restart the PHYSS. This feature reduces the power consumed by the device when used in situations where there is limited power is available.

The *portSwitching* parameter is used to enable/disable the port switching feature. If the value is 1, the role of the entry/exit ports is switched after each sample cycle. If the value is 0, this feature is disabled.

As mentioned earlier, the automated data collection process is controlled by a user-specified script. An example of such a script appears below, followed by a detailed explanation of each command.

```
# basic PHYSS operational script
run 100 120
on 1 8
    announce starting reference phase
    referenceSample 2 4
    optimizeIntegrationTime
    getDark dark
    checkLights
    getSpectrum reference dark
announce starting filtered sample phase (aka cdom)
filteredSample 3 2
getDark dark
getSpectrum filtered dark reference
announce starting unfiltered sample phase (aka disc)
unfilteredSample 2 4
getSpectrum concentrate dark filtered
repeat 2
    unfilteredSample 4 4
    getSpectrum unfiltered dark filtered
```

Like config files, scripts may include comments, that are introduced with a pound sign '#' and extend to the end of the line.

The *run* command must appear on the first line of any script. It has two arguments. The first specifies the number of sample cycles to be run prior to termination. A zero value causes data collection to continue indefinitely. The second argument determines the times at which sample cycles should be run. For example, a value of 120 minutes specifies that cycles should take place every two hours, starting on the hour. More generally, if the value is n minutes, each automatic sample cycle will start at a multiple of n minutes after midnight each day (the timezone is UTC). Sample cycles that are initiated manually by an operator take place on demand, so typically the first sample cycle of a deployment may take place any time, but subsequent sample cycles will be scheduled as specified by the run command.

The *on* command is used to specify operations that are only to be performed on some sample cycles. So for example, the command `on 1 8` specifies that the following commands are to be performed on the first of every group of eight cycles (so, sample cycles with numbers, 1, 9, 17, 25, etc). Indenting is used to identify which commands are controlled by a given on command (each level of indenting must add at least two spaces and the number of added spaces must be consistent). So, in the example, the five commands following the on comand are performed every eight sample cycles, while all other commands are performed every cycle. A script may include multiple on commands, but nesting of on commands is not recommended.

The *announce* command specifies a message that is to be displayed on the operations console whenever the command is executed. This can make it easier for an operator to monitor the execution of a script.

The *referenceSample* command causes the PHYSS to pump reference fluid into the waveguide. The two arguments specify the volume of fluid to be pumped (in milliliters) and the rate at which the pump should operate (in milliliters per minute). So the command `referenceSample 2 4` specifies two milliliters of reference fluid at the rate of 4 milliliters per minute. The range of pump rates depends on the specific pump being used, but the maximum rate is typically between 7 and 8 milliliters per minute. Pump rates are not at all precise; they should be interpreted with a precision of $\pm 10\%$.

The *optimizeIntegrationTime* command causes the PHYSS to adjust the spectrometer's *integration time* parameter. This specifies the amount of time (in milliseconds) that the light detector is exposed to the light passing through the waveguide. The integration time is adjusted to produce light measurements near the top end of the range of possible values. This is done to maximize the sensitivity of the spectrometer to the incoming light.

The *getDark* command causes the PHYSS to acquire a "dark" spectrum and associate a label with it. So the command `getDark dark` associates the label "dark" with the acquired spectrum. When a dark spectrum is acquired, the light source is turned on, but the shutter for the light source is kept closed, preventing the light from reaching the sample in the waveguide. Dark spectra are used to compensate for the fact that the spectrometer's light detector produces a non-zero reading even when no light is reaching the detector. Dark spectra are subtracted from sample spectra during data analysis to compensate for this effect. The light source is turned on, when acquiring a dark spectrum to enable compensation for the electrical noise generated by the light source. The acquired spectrum is saved in the raw data file produced by the PHYSS along with its label.

The *checkLights* command causes the PHYSS to perform a simple test on the light source, to ensure that it is operating correctly. If it is not, then an error message is displayed on the console and written to the debug file and raw data file.

The *getSpectrum* command causes the PHYSS to acquire a spectrum of the seawater sample currently in the spectrometer's wave guide and associate a label with it. So the command `getSpectrum reference dark` associates the label "reference" with the acquired spectrum. The second argument identifies a *prerequisite spectrum* associated with the newly acquired spectrum. In this example, there is a single prerequisite spectrum, the most recently acquired spectrum with the label "dark." When the spectrum is saved in the raw data file, the saved record includes a reference to the most recent dark spectrum, so that it can be accessed during data analysis. A spectrum may have up to two prerequisite spectra.

The *filteredSample* command causes the PHYSS to pump a seawater sample into the waveguide, with the filter valve configured to pass the sample through the filter. It has two required arguments plus two optional arguments. The first argument specifies the volume of fluid to be pumped (in milliliters) and the second specifies the rate at which to pump (in milliliters per minute). The two optional arguments are only used in the TWO_REAGENTS hardware configuration and determine how much of the sample is to be pumped from each of the two reagent supply reservoirs. So for example, the command `filteredSample 3 2 .2 .3` specifies that a total of 3 ml of fluid should be pumped at a total rate of 2 ml/m, with 20% of the total volume coming from the first reagent supply reservoir and 30% coming from the second reagent supply reservoir (so 50% is fresh seawater). The system chooses rates for all three pumps to produce the specified volumes and aggregate pump rate. During the acquisition of a filtered sample, the system monitors the pressure across the filter valve and if at any time it exceeds the specified maximum pressure, the operation is interrupted and the entire sample cycle is restarted. If the cycle continues to fail, automated sampling is suspended, after 10 failed attempts.

There is a second filtered sample command called *filteredSampleAdaptive*. This command does not include a pumping rate argument. Instead, it adapts the pump rate(s) in response to the pressure readings. If the pressure reading exceeds a safe target range, it reduces the pumping rate and if the pressure drops below the target range, it increases the pumping rate.

The *unfilteredSample* command has the same arguments as the *filteredSample* command. It causes the PHYSS to pump a seawater sample into the waveguide, with the filter valve configured to bypass the filter. Any particles that accumulated within the filter during a previous filtered sampling operation are flushed through to the waveguide, along with the new seawater sample. This increases the concentration of particles in the sample, effectively increasing the sensitivity of the spectrometer.

The *repeat* command causes a group of commands to be executed repeatedly. Its one argument specifies the total number of repetitions. The commands

to be repeated are specified using indentation. Repeat commands may be nested.

The output of the data collector is a series of data records formatted as JSON strings. Most of the output consists of spectrum records, but the output also includes various metadata records, including records containing the script and configuration files used to produce the spectra, plus status messages generated during each sample cycle. The output is written to a raw data file on the PHYSS and can be transferred from there to a remote cloud server (coolcloud.mote.org) where it can be accessed by users from anywhere in the world. A small excerpt from the raw data file is shown below (records have been abbreviated for clarity).


```
{ "serialNumber": 31, "index": 33985, "recordType": "deployment",
"dateTime": "2018-05-07 19:34:08", "label": "OT Lab", "wave
guideLength": 0.2800, "gpsCoord": "0.0/0.0", "wavelengths":
[188.65, 189.04, 189.43, 189.82, 190.21,  ...]}
{ "serialNumber": 31, "index": 33986, "recordType": "config",
"dateTime": "2018-05-07 19:34:08", "deploymentIndex": 33985,
 "configString": "logLevel = DETAILS@@@@maxFilterPressure =
 30@@gpsCoordinates ...}
{ "serialNumber": 31, "index": 33987, "recordType": "script",
"dateTime": "2018-05-07 19:34:08", "deploymentIndex": 33985,
 "scriptString": "run 100 120@@    # 100 cycles, 120 minute
between cycles@@ ...}
{ "serialNumber": 31, "index": 33988, "recordType": "debug",
"dateTime": "2018-05-07 19:34:08", "deploymentIndex": 33985,
"message": "going to idle mode [DETAILS 47.241]" }
{ "serialNumber": 31, "index": 33989, "recordType": "debug",
"dateTime": "2018-05-07 19:34:08", "deploymentIndex": 33985,
"message": "purging air bubbles [DETAILS 47.244]" }
{ "serialNumber": 31, "index": 34008, "recordType": "spectrum",
"dateTime": "2018-05-07 19:35:51", "deploymentIndex": 33985,
"prereq1index": 0, "prereq2index": 0, "label": "dark",
```

```
"spectrum":  [0.00, 0.00, 1608.76, 1551.00, 1569.68,  ...]}
{ "serialNumber": 31, "index": 34015, "recordType": "spectrum",
"dateTime": "2018-05-07 19:36:10", "deploymentIndex": 33985,
 "prereq1index": 34008, "prereq2index": 0, "label": "reference",
spectrum": [0.00, 0.00, 1591.77, 1561.67, 1590.80, ...]}
```


Note that every record starts with the *serial number* of the specific PHYSS unit, followed by a *record index*. Record indices are unique for a given PHYSS unit. Each time the data collector starts a new data collection run, it generates a *deployment record*, which includes some metadata information, including the wavelength values used by the PHYSS unit's spectrometer (these wavelengths can vary slightly from one spectrometer to another). Other records in the given run include a reference to the deployment record's index so that the data analysis console can easily locate the wavelength information associated with a given spectrum record. The *config record* contains a copy of the configuration file used to set the values of various data collector configuration variables. Similarly, the *script record* contains a copy of the script used for the specific run. The data analyzer on the cloud server reads raw data files and presents their contents to the user.

4 Using the Data Analysis Console

The *Data Analysis Console* enables users to access the data from multiple PHYSS units through a cloud server. It consists of two parts, a server program that runs on the cloud server and a Javascript program that runs in the user's web browser. To access the console, simply enter "coolcloud.mote.org/data" in the URL window of your web browser. This brings up the analysis console shown in Figure 4. The main chart area is used to display spectra acquired by the PHYSS. Spectra can be displayed in several different ways. In the figure, the black curve is an unprocessed spectrum, exactly as produced by the PHYSS. The x-axis shows the wavelengths (in

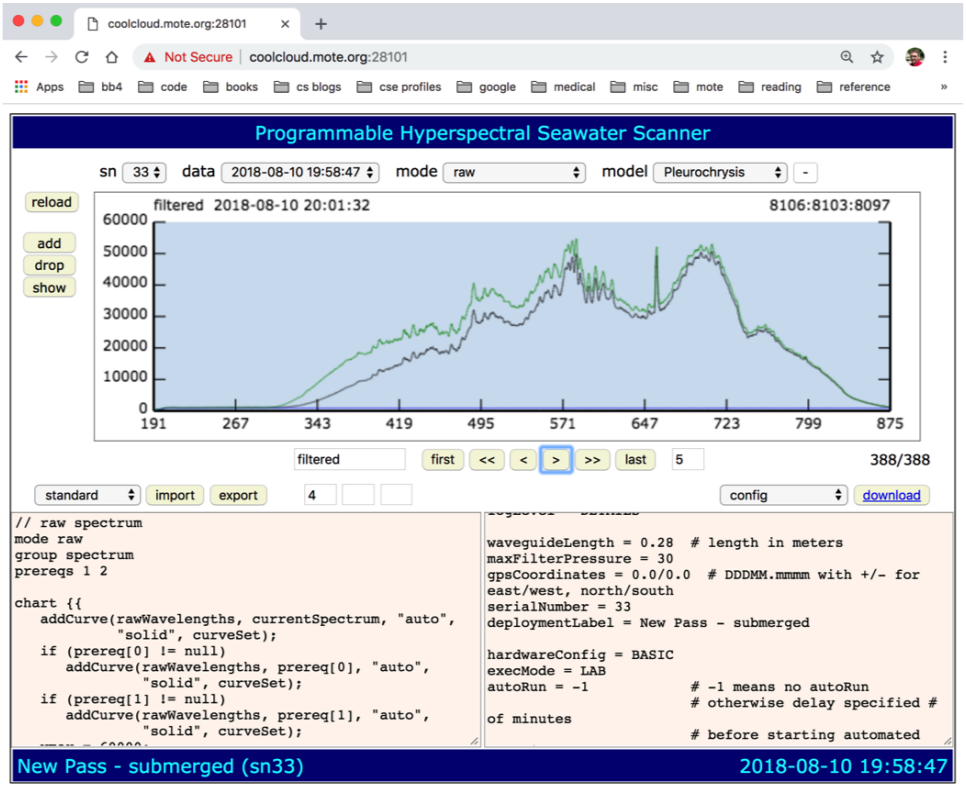

Figure 4 Data analysis console

nanometers), while the y-axis shows light intensity on a unitless scale, with a maximum possible value of 65535.

The text at the top left of the chart area shows the label associated with the displayed spectrum by the script used for data collection, plus the date and time at which the spectrum was acquired. The three numbers at the top right are record indices; the first is the record index for the current spectrum and the other two are optional record indices for associated *prerequisite spectra*. So, in the chart shown in the figure, the black curve is the filtered seawater spectrum contained in record 8106. The blue curve is the first of the two

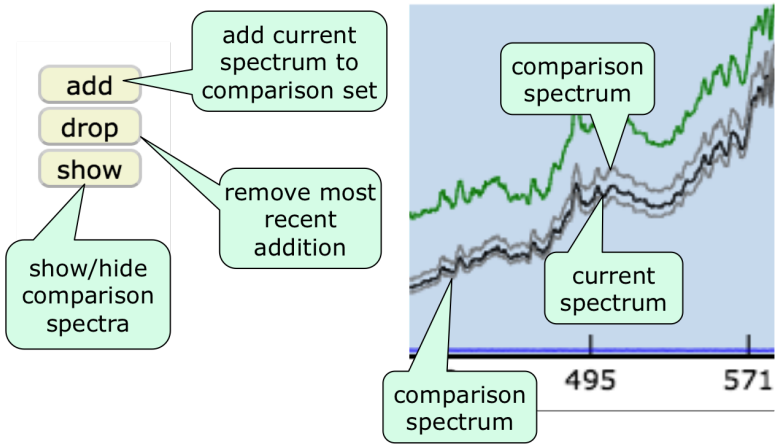

Figure 5 Comparing Spectra

prerequisite spectra (record 8103). In this case, it is the dark spectrum associated with the filtered sample spectrum. The green curve is the second of the two prerequisite spectra (record 8097), which in this case is the spectrum for the reference sample associated with the filtered sample spectrum.

The buttons just below the chart area are used to navigate within the current file. The button labeled '>' advances to the next spectrum in the current data file, the button labeled '<' goes back to the previous spectrum. Similarly, the '>>' and '<<' buttons advance several spectra forward or back, where the size of the 'jumps' is determined by the value in the textbox to the right. The textbox to the left is used to limit the displayed spectra to only those with a specified label (to see all spectra, leave it blank or type "any").

The *add* button to the left of the chart area is used to add the current spectrum to a set of "comparison curves". Curves in this set are shown in gray, allowing the user to display several related curves together for ease of comparison. Comparison curves are removed using the *drop* button and the display of comparison curves is turned on or off with the *show/hide* button. This is illustrated in Figure 5.

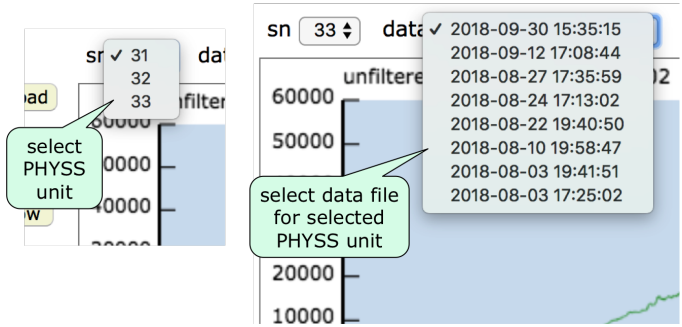

Figure 6 File Selection

The first of the menus just above the chart area specifies the current PHYSS unit (using the unit's serial number). The second menu specifies a specific data file for that PHYSS (specified by the data and time at which sample collection started for that data file). An example of these menus is shown in Figure 6.

The third menu specifies the *chart mode* that determines how the data is processed and displayed (in the figure, the chart shows an unprocessed spectrum as collected by the spectrometer). The menu of chart modes appears in Figure 7. The menu is divided into three groups. The *spectrum group* shows individual spectra. The *time series group* shows data values computed from selected spectra within the file and presents these data values as a time series. The *status group* shows the values of operational status variables and presents them as a time series. The various chart modes will be discussed in detail later in this section.

The fourth menu contains a list of phytoplankton model files, that can be compared to particle absorbance spectra in order to test for the presence of various phytoplankton species in the sample. The contents of this menu is shown in Figure 8. The use of this menu is discussed later in this section.

The text area at the bottom right of the display shows the configuration parameters that were used when the displayed spectra were acquired. The

menu just above the text area can be used to select other information, including the script used to control the sampling process, the status messages generated during data collection and the *cycle summary* records that are generated by the PHYSS at the end of each sample cycle. It can also be used to display the numerical data that appears in the current chart; the displayed data can then be copied and pasted into a text file stored on the user's computer. That file can then be imported into a spreadsheet for further analysis. Clicking on the *download* button to the right of the menu downloads a copy of the current data file to the user's computer (typically to the user's Downloads folder).

The text area at the bottom left contains a library of *chart mode definitions*. This defines the various ways in which the data in a given file can be analyzed and presented. The chart modes defined here are the same chart modes that appear in the chart mode menu, just above the chart area. Users can modify the standard chart modes, or define new chart modes by editing the information in this text area. This will be discussed in detail in the next section. The remainder of this section describes the various chart modes that are available in the default library, called the *standard* library.

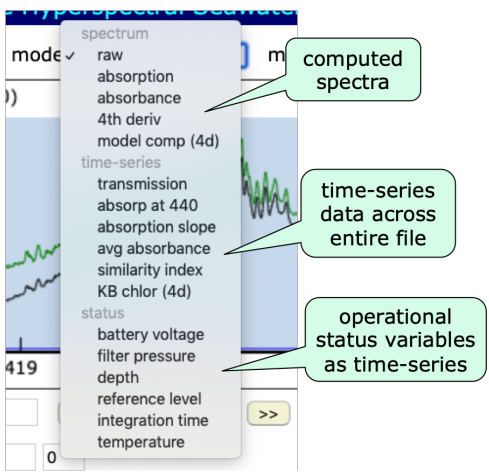

Figure 7 Chart Modes

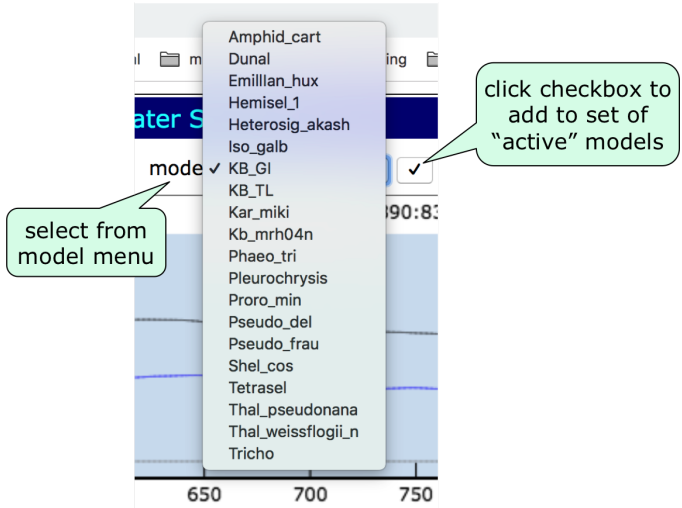

Figure 8 Phytoplankton Model Files

The simplest chart mode simply shows the raw spectra that are acquired by the PHYSS and transferred to the cloud server. Figure 4 shows a spectrum displayed in this way, along with its prerequisite spectra.

The standard chart mode library implements a set of data analysis procedures that were originally developed for the OPD (the predecessor of the PHYSS). To make effective use of the data produced by the analysis console, it's important to understand what these analysis procedures do. There is a preprocessing procedure that is done before proceeding to more specific analysis procedures. This consists of three steps *smoothing*, *wavelength standardization* and *dark subtraction*.

The smoothing step replaces each data value in a raw spectrum with a weighted sum of the data values in its neighborhood. The weights follow a truncated Gaussian distribution so nearby values are weighted more heavily than those that are further away. An example of a smoothed spectrum appears in Figure 9. As with the original raw spectra, the figure shows both

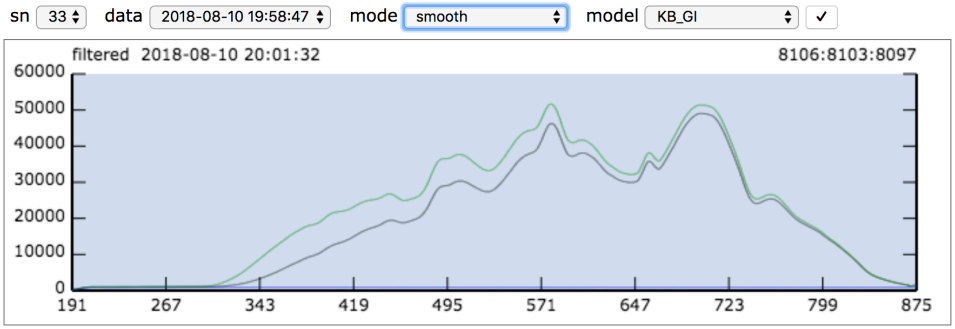

Figure 9 Smoothed spectra

the current spectrum (in black) and its two smoothed prerequisites (in blue and green).

The wavelength standardization step replaces the original 2048 data values in the spectrum with 450 values at integer wavelengths from 350 to 800 nm. This is justified by the fact that the spectral features of interest are wider than 10 nm, so the finer wavelength resolution provided by the spectrometer is not really useful. Also, the use of a standard set of wavelengths makes it easier to compare spectra from different spectrometers. Finally, the use of a smaller number of equally spaced wavelengths enables more efficient computational procedures.

As mentioned in the previous section, a spectrometer produces a non-zero reading even when no light reaches the detector. This is due to thermal noise that is intrinsic to the detector, plus other electrical noise that may be present in the system. The purpose of the dark subtraction step is to compensate for this. Given a spectrum from a seawater sample and a dark spectrum, the two steps described above are applied to both spectra. Then the processed dark spectrum is subtracted from the processed sample spectrum, resulting in a *cooked* version of the original raw spectrum. This cooked version is what is used in subsequent processing steps. An example of a cooked spectrum is shown in Figure 10 (the current spectrum is the black curve, its second prerequisite is the blue curve).

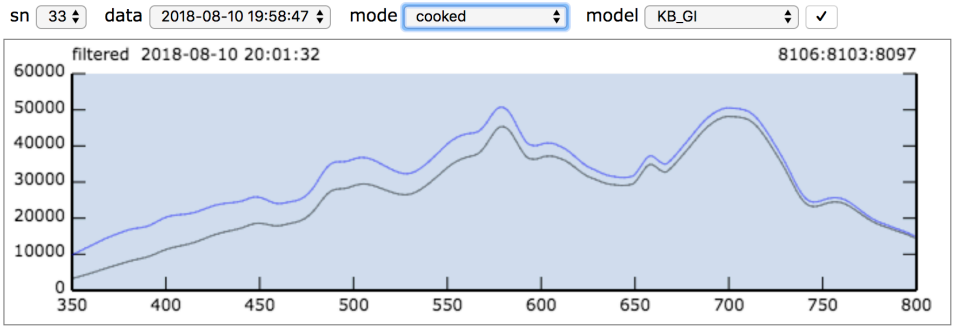

Figure 10 Cooked spectra

Figure 11 shows the absorption spectrum for the given filtered seawater sample. This is the natural log of the ratio of the reference spectrum to the sample spectrum. Two versions of the absorption spectrum are shown. The black curve is the initially computed value. The blue curve is a "shifted" version that is brought down to contact the horizontal axis. A decaying exponential function is fitted to this blue curve and used to compute two parameters that are conventionally used to characterize absorption spectra from filtered seawater samples. These are the absorption value at 440 nm (1.102 in this case) and the absolute value of the coefficient of the exponent

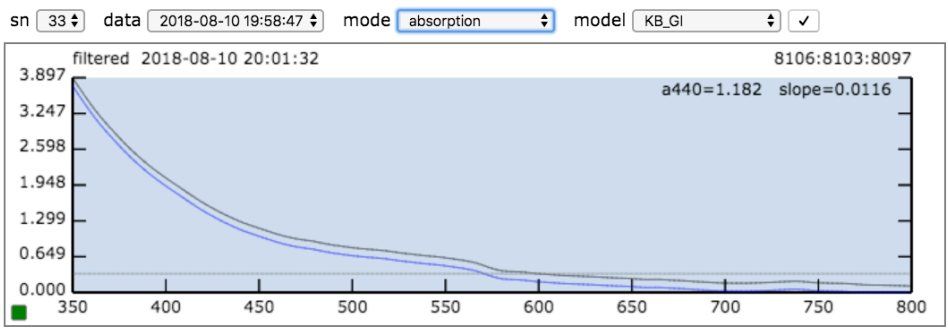

Figure 11 Absorption spectrum

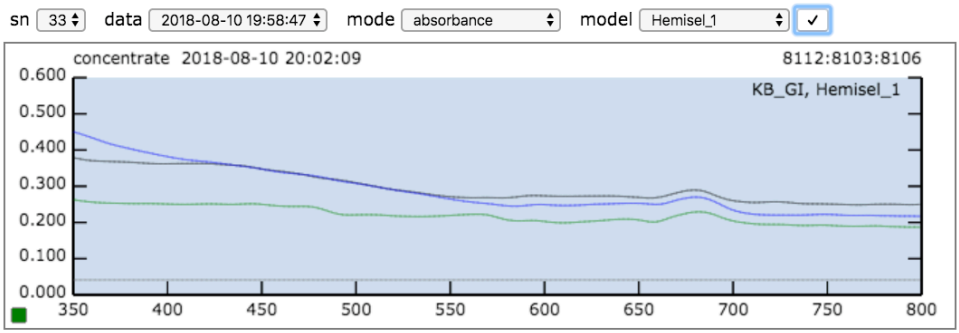

Figure 12 Absorbance spectrum

in the exponential function (referred to as the slope parameter, 0.0116 in this case). The gray line near the bottom shows where the absorption corresponds to a 10% reduction in transmitted light from the sample to the reference. Absorption values that fall substantially below this threshold may be unreliable and should be treated with caution. The small green square at the lower level of the chart area is a *quality assurance indicator*. A green square indicates that the computed spectrum has passed a series of tests designed to detect bad data. A red square indicates that the spectrum has not passed one or more of the tests and a yellow square indicates that it passed, but should be treated with caution. Details of the criteria used to identify invalid or marginal spectra can be found in the next section.

Figure 12 shows the absorbance spectrum for an unfiltered seawater sample containing concentrated particulate matter. This spectrum is the base 10 logarithm of the ratio of an associated filtered spectrum to the spectrum for the unfiltered concentrate. As usual, the current spectrum is the black curve. The other two curves are the absorbance spectra for two phytoplankton species, selected from the menu of phytoplankton models. The gray line shows where the absorbance corresponds to a 10% difference in light transmission.

Figure 13 shows the fourth derivative of the previous absorbance spectrum, along with the fourth derivative spectrum of two selected phytoplankton

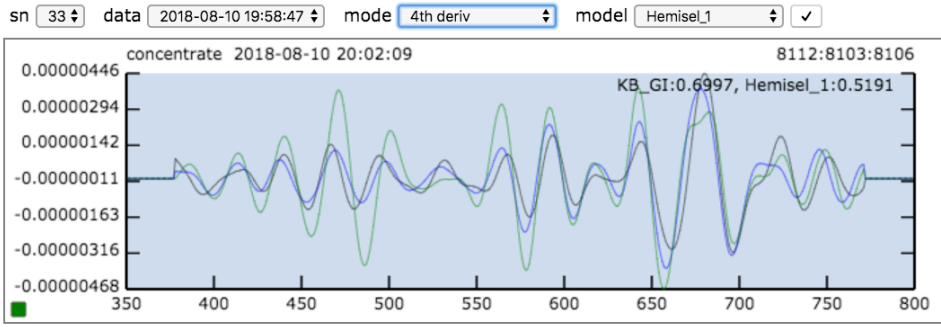

Figure 13 Fourth derivative spectrum

model files. These fourth derivatives are compared to produce a *similarity index* for each of the selected model files. These are shown at the top right of the chart area. Values of 0.7 and above are considered strong evidence for the presence of a specific species. The similarity index quantity is more conventionally know as the *angular similarity* and is based on the angular separation of the two spectra when viewed as vectors in n-dimensional space. It is computed on the portion of the spectra from 400 to 700 nm. The fourth derivative value at each data point is computed based on a seventh-order polynomial approximation to the curve defined by the data point and its 56 neighboring data points.

Figure 14 shows the absorbance spectrum of an unfiltered seawater sample (black curve), and a computed approximation obtained by taking a weighted sum of selected phytoplankton models. The green curve shows the difference between the sample absorbance and the approximation. The combination of phytoplankton models that produced the best match to the given sample spectrum are listed in the top line of text in the upper right corner fo the chart area. The numbers show the average absorbance for each of the selected models. The next line shows a selected scattering model (with slope parameter .2) and a detrital absorbance model (with slope parameter .0135). The third text line shows metrics that indicate how well the computed approximation fits the data. Specifically, the first two numbers are the average

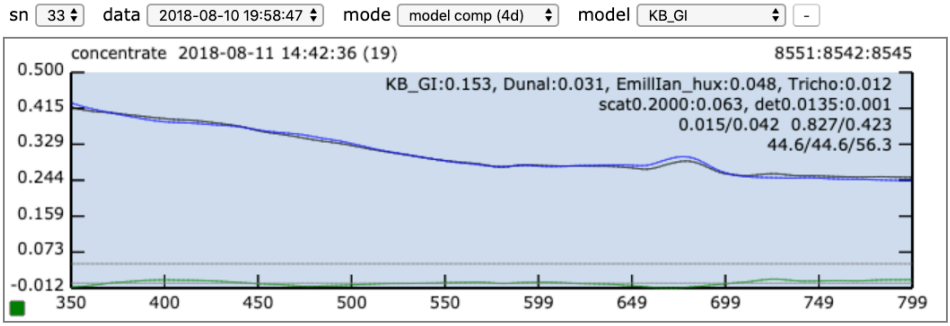

Figure 14 Model composition

and maximum relative error of the computed absorbance spectrum, relative to the sample absorbance. The third number is the similarity index of the fourth derivatives of the two spectra and the fourth number is a normalized errror metric for the fourth derivative spectra (defined as the sum of the absolute differences divided by the sum of the sample fourth derivative magnitudes). The fourth line shows the estimated chlorophyll contributions from Karenia Brevis. Specifically, the first value is the estimated KB chlorophyll for the computed approximation to the sample spectrum and the third value is the total chlorophyll for all the phytoplankton components. The middle value is used to display an estimate based on the n best computed approximations to the sample, where n is a user-specified parameter (in this example $n = 1$).

The data analysis console can also display various values computed from spectra in the file in a time-series format. For example, Figure 15 shows the absorption at 440 nm for all the filtered spectra in the current data file. The x-axis is labeled in hours since the start of the deployment. The history of the slope parameter can be displayed similarly. Figure 16 shows the values of the similarity index for all concentrate spectra (relative to a model file for Karenia Brevis). Observe that most of the values in this curve are above the threshold of 0.6 and many are above 0.7. This data was acquired during a strong Karenia Brevis bloom.

A variety of operational status variables can also be displayed by the analysis console in time-series format. These are primarily used by operators, monitoring the status of deployed PHYSS units. These are not computed from spectra, but are extracted from *cycle summary records* produced by the PHYSS at the end of each sample cycle. Figure 17 shows the power supply voltage to the PHYSS unit. This particular PHYSS was powered from a battery connected to a solar panel. This accounts for the 24 hour variation in the voltage readings. The cycle-to-cycle variation in the readings is caused by the port-switching feature (the port valves consume more power in one state than the other). Other variables that can be displayed include the maximum pressure across the filter in each sample cycle, the depth of the PHYSS below the water level, the spectrometer integration time, the amount of reference fluid remaining in the reservoir and the ambient temperature within the PHYSS enclosure.

Before wrapping up this section, there is one last feature of the mode library subsystem to discuss. First, note that just to the right of the import and export buttons, there are four text boxes. These are used to define three Javascript variables, `userArg1`, `userArg2`, `userArg3` and `userArg4`. Chart mode libraries can use the values of these variables to modify their behavior. In fact, the standard library uses `userArg1` as a curve smooothing param-

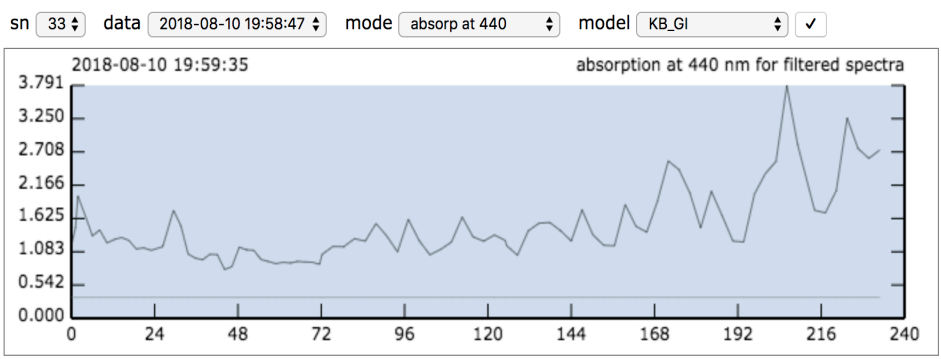

Figure 15 Time series of absorption at 440 nm for filtered spectra

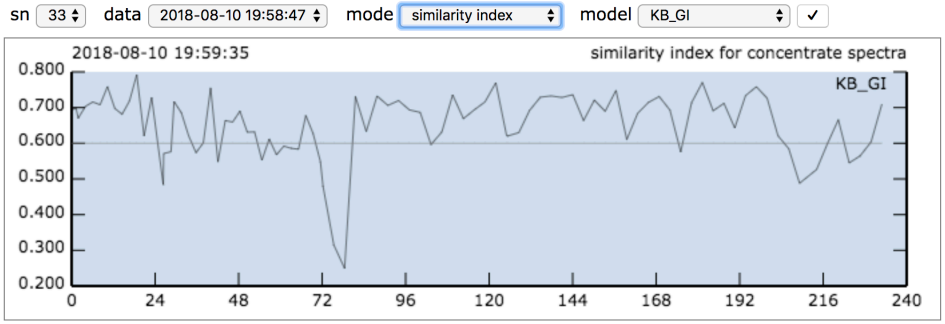

Figure 16 Time series of similarity index for Karenia Brevis

eter, and several of the time-series and status modes use it. The battery voltage chart is one of these. If you enter a value of 2 in the first text box and hit enter, the voltage curve will be replaced by one in which each datapoint is the average of two successive values. This will eliminate the jagged sample-by-sample variations in the voltage curves. A value of 12 will average out the time-of-day variations, allowing you to better observe any long-term drift in the voltage levels. In addition, `userArg3` is used to specifiy the number of computed absorbance models to use when estimating KB chlorophyll contributions.

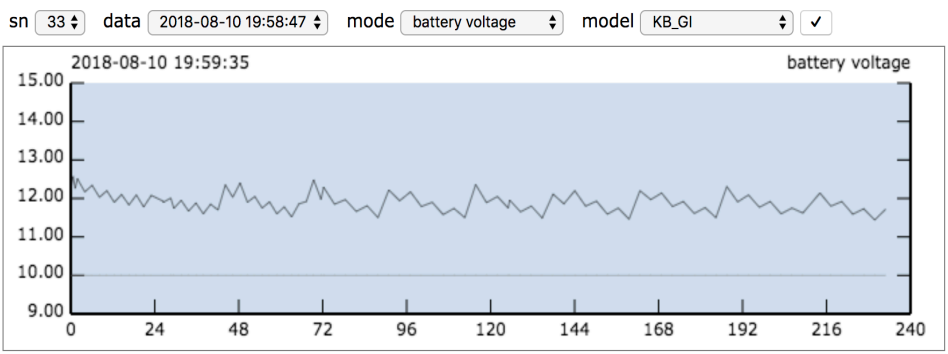

Figure 17 PHYSS Battery Voltage

5 Defining Chart Modes

As discussed in the last section, chart modes provide different ways to analyze and view the data produced by the PHYSS. Chart modes are defined in a *chart mode library*, which is displayed in the text box at the lower left of the analysis console. Just above this text box is a menu listing all the available libraries. This menu always includes a default library, called the *standard library*, a library based on the original OPD analysis methods, called the *classic library*, a library containing chart modes that are under development, called the *experimental library* and a user-defined library called the *user library*.

An example of a chart mode from the standard library appears below.

```
// raw spectrum
// current spectrum and its direct prereqs
mode raw
group spectrum
prereqs 1 2

chart {{
   this.addCurve(lib.rawWavelengths,
         this.currentSpectrum, "auto", "solid");
   if (this.prereq[0] != null)
      this.addCurve(lib.rawWavelengths,
            this.prereq[0], "auto", "solid");
   if (this.prereq[1] != null)
      this.addCurve(lib.rawWavelengths,
            this.prereq[1], "auto", "solid");
   this.ymax = 60000;
   if (this.spectrumLabel == "dark")
      this.ymax = 3000;
}}
```

The first two lines are comments. In general, a comment is introduced by a pair of foward slashes and continues to the end of the line. The third line specifies the name of the chart mode. This is the name that appears in the chart mode menu. The fourth line specifies that this chart mode belongs to the *spectrum group*. The fifth line specifies the prerequisite spectra that are used to produce the chart; specifically, it specifies prerequisite 1 and 2 of the current spectrum. One can specify additional prerequisite spectra relative to the prerequisites of the current spectrum. For example, the prerequisite string 21 specifies prerequisite 1 of prerequisite 2 of the current spectrum. Similarly 211 specifies prerequisite 1 of prerequisite 1 of prerequisite 2.

The Javascript program that implements the analysis console defines a *Mode object* for each mode defined in the library. Each Mode object includes data values that are used to produce charts using that mode. This data includes the mode name, group and prerequisite list, among other things.

In the mode definition, there is a group of lines preceded by the string "`chart {{`" and followed by the string "`}}`". These lines are *Javascript* code and are used to define a member function of the Mode object. This function is executed during the production of a chart using that mode.

The first two lines of the `chart {{..}}` block for the raw mode are repeated below.

```
this.addCurve(lib.rawWavelengths,
      this.currentSpectrum, "auto",  "solid");
```

This code invokes the method `this.addCurve` of the Mode object, which adds a new curve to a set of curves used to produce the current chart. It takes four arguments. The first two are vectors which specify the x and y coordinates of the datapoints forming a single curve; `lib.rawWavelengths` contains the wavelengths used by the spectrometer for the PHYSS unit and `this.currentSpectrum` contains the light intensity values for the current

spectrum. (The object `lib` is referred to as the "core library" and contains a collection of broadly useful data definitions and methods that are intended for use by multiple mode libraries.) The last two arguments are strings that control the color and line style to be used for this curve. The color can be specified using any standard javascript color name ("red", "green", etc) or can be specified as "auto". In the latter case, colors for different curves within a chart are assigned automatically from a fixed menu of colors. The line style can be solid, dashed or dotted.

The next three lines of the raw mode definition are repeated below.

```
if (this.prereq[0] != null)
   this.addCurve(lib.rawWavelengths,
                 this.prereq[0], "auto", "solid");
```

The vector `this.prereq` contains the prerequisite spectra specified in the *prereqs* line. Since the prereqs line specified prerequisites 1 and 2 of the current spectrum, *this.prereq[0]* is the actual prerequisite 1 spectrum and `this.prereq[1]` is the prerequisite 2 spectrum. When additional prerequisites are specified, the additional spectra just appear in sequence in `this.prereq`. If the current spectrum does not have a valid prerequisite 1, `this.prereq[0]` is null. Similarly, if it does not have a valid prerequisite 2, `this.prereq[1]` is null.

The last few lines of the mode definition are repeated below.

```
this.ymax = 60000;
if (this.spectrumLabel == "dark")
   this.ymax = 3000;
```

These lines specify a default maximum value for the y axis. The actual value used will be larger than this if there is some datapoint in the chart with a y value that exceeds the default value. `This.spectrumLabel` is the label associated with the current spectrum in the raw data file.

Here are the first few lines of the mode definition for the *absorption* chart mode.

```
// absorption spectrum
// shows absorption, shifted version and
// parameters derived from fitted exponential
mode absorption
group spectrum
labels filtered
prereqs 1 2 21
```

The `labels` line is used to restrict the application of a chart mode to those spectra with matching labels. So in this case, the absorption mode can only be applied to spectra with the label "filtered." In general, there may be several labels. If the `labels` line is omitted, it means that the mode can be applied to any spectrum.

The next few lines of the *absorption* mode appear below.

```
chart{{
   if (this.prereq[0] == null ||
       this.prereq[1] == null ||
       this.prereq[2] == null) {
      this.addCurve(lib.rawWavelengths,
            this.currentSpectrum, "auto", "solid");
      return;
   }
```

To produce an absorption spectrum, we generally need the associated reference spectrum and two dark spectra. If any of the required prerequisites is missing, we simply add the raw spectrum to the set of curves for this chart. Note the `return` statement. Recall that the *Javascript* code is part of a function definition, so this line specifies an early return from the function.

The next few lines of the *absorption* mode appear below.

```
var sample = slib.cook(this.currentSpectrum, this.prereq[0]);
var reference = slib.cook(this.prereq[1], this.prereq[2]);
var spect = slib.absorption(sample, reference);
var shiftedSpect = lib.vectorShift(spect);
```

The standard library method `slib.cook` produces a cooked spectrum from a given sample spectrum and dark spectrum. Recall that this involves smoothing both spectra, integerizing their wavelengths and then subtracting the processed dark spectrum from the processed sample spectrum. The standard library method `slib.absorption` computes an absorption spectrum from a cooked sample spectrum and its associated reference spectrum. The core library method `lib.vectorShift` shifts a given vector of values so that it's minimum values are zero.

The next few lines of the *absorption* mode appear below.

```
this.addCurve(lib.cookedWavelengths, spect,
              "auto", "solid");
this.addCurve(lib.cookedWavelengths, shiftedSpect,
              "auto", "solid");
this.addLine(lib.cookedWavelengths,
      Math.log(1.1)/lib.waveguideLength, "gray", "solid");
```

The method `this.addLine` adds a horizontal line to the display. It is similar to `this.addCurve`, but the second argument is a constant value, rather than vector. In this case, the line shows where the absorption values correspond to a difference in light transmission of 10%. Absorption values below this threshold are somewhat less reliable.

Here are the next few lines of the *absorption* mode.

```
var abp = slib.absorptionParameters(shiftedSpect);
this.chartLabel = "a440=" + abp[0].toFixed(3) +
      "   slope=" + abp[1].toFixed(4);
this.ymax = 3; this.yprecision = 3;
```

The method `slib.absorptionParameters` fits the shifted spectrum to an exponential curve, then returns the value of that curve at 440 nm and the absolute value of the exponent (referred to as the slope parameter). These parameters are returned as a two element vector. The values are then inserted into the chart label. The last line specifies a maximum y-axis value of 3 and sets the precision of the values on the y-axis to 3 decimal places.

The last few lines of the absoprtion mode appear below. These lines define the value of a *quality assurance* variable. If this variable is assigned a value of 1, 0 or -1, it causes a colored square (green, yellow or red) to appear in the lower left corner of the chart (as in Figure 11).

```
// compute quality assurance value (qa)
this.qa = 1;
this.qa = Math.min(this.qa,
             slib.qaReference(this.prereq[1]));
this.qa = Math.min(this.qa,
             slib.qaAbsorption(spect,
                shiftedSpect, abp[0], abp[1]));
```

The method *slib.qaReference* computes a quality assurance value for a reference spectrum. The method *slib.qaAbsorption* computes a quality assurance value for an absorption spectrum.

The time-series and status modes are somewhat more complicated than the spectrum modes. Here is the definition of the *battery voltage* mode, which displays the battery voltage as a function of time.

```
mode battery voltage
group status

datapoint {{
   return [ this.cycleSumRecord.battery ];
}}

chart {{
   this.addCurve(this.xvec, this.yvecs[0],
                         "black", "solid");
   this.addLine(this.xvec, 10, "gray", "solid");
   this.rightHeader = "battery voltage";
   this.ymin = 9; this.ymax = 15; this.yprecision = 2;
   this.yticks = 6;
}}
```

The main new element here is the `datapoint` {{..}} code block. This code is used to define a method that is invoked once for every cycle summary record in the current data file. Before it is called *this.cycleSummaryRecord* is set to the current cycle summary record, making all of the status variables defined in the record available. The datapoint method returns a vector of values (in this case that vector has a single element) representing y-values for one or more curves.

Before the `chart` code is called, `this.xvec` is initialized to contain the time values associated with the cycle summary records and `yvecs` is initialized to

contain the y-values returned by the datapoint method calls. So in this case, `yvec[0]` contains the battery voltages returned by the datapoint methods.

Here is the mode definition for the transmission mode. This is an instance of the time-series group.

```
mode transmission
group time-series
labels filtered

datapoint {{
   if (this.currentSpectrum == null) return null;
   var i440 = lib.functionValue(lib.rawWavelengths,
                                this.currentSpectrum, 440.);
   var imax = lib.vectorMax(this.currentSpectrum);
   return [i440, imax];
}}

chart {{
   for (var i = 0; i < this.yvecs.length; i++)
      this.addCurve(this.xvec, this.yvecs[i],
                    "auto", "solid");
   this.rightHeader = "transmission values for " +
                      "filtered spectra";
}}
```

For modes in the time-series group, the *datapoint* method is called once for every spectrum whose label matches one of the labels in the *labels* line. So in this case, it is invoked for all of the filtered spectra. The datapoint method returns a pair of values, one representing the raw transmission value at 440 nm and another representing the maximum transmission value in the current spectrum. The method `lib.functionValue` interprets its first two arguments as the x and y coordinates of a function and its third argument as a specific x value. It returns the corresponding y value.

Each mode library includes a special mode called `initialize`, which contains initialization code that is executed when the mode library is loaded. Here is a small portion of the `initialize` mode for the standard mode library.

```
mode initialize
group special

init {{
   this.userArg1 = 1;
      // controls smoothing of selected time-series
      // and status curves
   this.userArg2 = 0;
   this.userArg3 = 1;
      // specifies # of models to incorporate into
      // results for model composition and KB chlor
      // time-series
   this.userArg4 = 0;

   // wavelength range for computing cdom
   // absorption parameters
   slib.CDOM_MINWAVE = 390;
   slib.CDOM_MAXWAVE = 490;

   // wavelength range for computing sim index
   slib.SIM_MINWAVE = 400;
   slib.SIM_MAXWAVE = 700;
```

The first few lines assign default values to the user argument text boxes. In the standard library, the first of these is used to control the smoothing of time-series data for selected time-series and status modes. The default value of 1 specifies no smoothing. A value of $k > 1$ creates a "moving average" that shows the average of overlapping sequences of k datapoints. The third user argument is used by the model composition and KB chlorophyll chart

modes to control the number of constructed approximate models to include in the displayed results.

The next group of lines define constants that specify certain ranges of wavelengths that are used by various methods. The remainder of slib contains definitions of methods that can be used by the chart modes in the standard library. Here is an example, including explantory comments.

```
/** Compute a "cooked" spectrum for a given
 *  raw spectrum.
 *  @param sample is a raw spectrum to be cooked
 *  @param dark is the associated dark spectrum
 *  @param wavelengths is an optional vector of raw
 *  wavelengths; it defaults to lib.rawWavelengths
 *  @return a spectrum obtained by first smoothing
 *  and integerizing sample and dark, then taking
 *  the difference between the results.
 */
slib.cook = function(sample, dark, wavelengths) {
   if (wavelengths === undefined)
      wavelengths = lib.rawWavelengths;
   return lib.vectorSubtract(
         lib.integerize(wavelengths,
             lib.smooth(sample, 23, 12),
             lib.MINWAVE, lib.MAXWAVE),
         lib.integerize(wavelengths,
             lib.smooth(dark, 23, 12),
             lib.MINWAVE, lib.MAXWAVE));
};
```

The complete text of the standard mode library can be found in the analysis console. To develop a more complete understanding of the library, you may find it convenient to create a copy in a text file on your own computer

by performing a copy/paste operation from the library text window in the analysis console to your local file. Each chart mode and method includes some basic documentation that should help you understand what they do.

For the remainder of this section, the focus is on how users can define their own mode libraries. The first step is to export a copy of the currently selected library by clicking on the *export* button. This creates a copy of whatever library is currently displayed and makes that copy editable. When you click on the export button, you'll notice that the background color in the mode library window changes to white (indicating that the text can be edited) and the mode menu option changes to *user*. At this point, you can make any changes you want to the chart modes and make your changes active by clicking on the *import* button. As a simple example, you might try changing the default `ymax` value in the raw mode from 60000 to 50000. Notice how this changes the display of the charts.

Observe that modes within a mode library are separated by lines containing nothing but equal signs. To define a new mode, insert a new line of equal signs next to an existing one and type your new mode definition between the two lines. You might start by simply copying an existing mode, giving it a new name and making some small changes. When you're ready to try out your new chart mode, just click on the import button and you'll see that the new chart mode appears in the chart mode menu. Select the new mode from the menu to see it in action.

You can also make changes code to the *init* mode. For example, you could edit the cook method to adjust the smoothing parameters. Or you could define a new method to implement some computation that will be used repeatedly in new chart modes that you create.

Because chart modes define new Javascript methods, they will typically have to be debugged like any other program. To make debugging easier, you will want to turn on the Javascript console provided by your web browser.

You can then use the standard method `console.log()` to insert debugging statements into your code. In general, it's a good practice to develop new chart modes in an incremental fashion, verifying each new line of code as you go along. If you are not familiar with Javascript there are many online resources that you can use to learn the basics. The website `https://www.w3schools.com/` is one excellent source.

Note that changes to the chart mode definitions are local to a particular analysis console. That is, they affect only the code running in your browser and will not affect other users, so you can feel free to experiment. This also means that your changes are not saved on the cloud server, so if you close your browser session, any changes you have made in the mode library window will be lost. However you can save your chart modes to your local computer, simply by selecting all the text in the mode library window, then doing a copy-paste operation to transfer the text into a file on your computer. To configure a new browser session with your personal library, simply copy-paste the contents of your local file back into the mode library window.

As mentioned earlier, some chart modes make use of the user argument text boxes to enable users to modify the default behavior. The battery voltage mode is one of these. Here is the relevant part of its implementation.

```
this.addCurve(this.xvec,
         lib.vectorAverage(this.yvecs[0], this.userArg1),
         "black", "solid");
```

The method `lib.vectorAverage` takes two arguments, a vector of values v and a positive integer i. In the result vector r, element $r[j]$ is defined as the average of the i values, $v[j-i+1], \ldots, v[j]$. If `userArg1` is 1, the result vector is the same as the input vector. Larger values produce smoother curves.

6 Summary

One of the key features of the PHYSS device is that it is user-programmable, providing a great deal of flexibility in the data collection procedures. The data analysis console delivers similar flexibility to the analysis of the data acquired by the PHYSS. Most of this report has focused on the use of the default chart modes provided by the standard mode library. However, users can easily tailor the standard chart modes to special circumstances, or can develop their own libraries if the provided ones do not meet their needs. Please direct questions and suggestions to jon.turner@wustl.edu.

References

[1] 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. 2

[2] 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. 2

[3] Molloy, Derek. *Exploring Beaglebone: Tools and Techniques for Building with Embedded Linux*, Wiley, 2014. 4