

# An Implementation of Integrated Visualization & Endpoint Modelling

By

Michael W. Schmidt

A thesis presented to the Honors College of Middle Tennessee State University in partial fulfillment of the requirements for graduation from the University Honors College.

Fall 2018

# An Implementation of Integrated Visualization & Endpoint Modelling

Michael W. Schmidt

APPROVED:

---

Dr. Ryan Otter  
Dept. of Biology

---

Dr. Phillip Phillips  
University Honors College Associate Dean

# **Abstract**

This creative work investigated Integrated Visualization and Endpoint Modeling with the intent to construct a software prototype. Integrated Visual Endpoint Modeling (IVEM) is a retrospective analysis tool that aids in assessing environmental damage. Specifically, IVEM is used to assess particular regions of a contaminated site using field collected data, but data are not directly comparable to one another. At present, a software implementation of IVEM did not exist. This paper details one approach to software implementation of IVEM utilizing C++, PHP, and web technologies to deliver a proof of concept. The works presented here resulted in a functional IVEM web application.

## List of Figures, Appendices, Symbols and Equations

<b>Figure 1</b>	A high level diagram illustrating the IVEM implementation process flow.
<b>Figure 2</b>	An IVEM result from processed data.
<b>Appendix A</b>	Sample data utilized for program development and testing.
<b>Appendix B</b>	File format specification found on the hosted the IVEM implementation. This is the data format the application expects to receive.
<b>Appendix C</b>	Source code for the IVEM implementation.
$\pi$	Greek little Pi. A numerical constant which is approximately 3.14159
$e$	Euler's number. A numerical constant which is approximately 2.71828
$\theta$	Greek Theta. A variable typically used to represent an angle in radians. A circle has $2\pi$ radians.
$\sum_{i=1}^n i$	Summation. The current expression is the sum of $1 + 2 + 3 + \dots + n$
$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$	Arithmetic Mean.
$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$	Standard Deviation.
$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$	Standard Error of the mean.
$Area = \frac{ab \sin(C)}{2}$	Area of a triangle, Side-angle-side method. (Sides: $a$ , $b$ ; Angle: $C$ )

# **I. Introduction**

Environmental disasters will never go away, but retrospective testing methods let us assess damage and provide insight for clean-ups. One chief issue common to both man-made and natural disasters are the resources we choose to allocate to these affected areas. Accurately evaluating affected areas involves numerous tests from several different disciplines, and carried out across different regions (Smith 2013). To challenge matters, collected data does not always agree with similar testing methods. It is often hard to reach a consensus among scientists as to which areas are the most affected since each test describes the scenario differently. The panel scientists report their findings to are not typically composed of scientists which is also problematic.

Integrated Visualization and Endpoint Modelling (IVEM) is a method of standardizing, structuring, and presenting data developed by Dr. Ryan Otter. His approach to processing and visualizing data overcomes the issues faced by scientist and regulatory bodies described above. Even though IVEM solves these problems, creating an IVE model is a time consuming and error prone task. Several statistical values such as arithmetic mean, standard deviation, standard error, and the minimum/maximum values are need for each and every test and region. Trigonometry is also used for generating plots. All of Dr. Otter's IVE models are currently performed on a cumbersome Microsoft Excel spreadsheet.

Computers are able to solve a myriad of problems for their ability to retain large amounts of data and execute hundreds of billions of operations per second on modern hardware. Spreadsheet applications can be excellent tools for solving general

data-centric problems, but more complex problems may require a program tailored to a specific need. Computer science is an interdisciplinary field for solving computationally heavy problems in Mathematics, Biology, Physics, Chemistry, and other data driven fields.

A computer program to perform these burdensome tasks presents an excellent opportunity to apply computer science in the environmental toxicology field. IVE models require organizing data, creating statistics that describe the data, manipulation of that data, and generating plots. As of October 2015, no full implementation exists; only a complicated Microsoft Excel document which serves as a proof of concept. Creating a functional IVEM implementation applies computer science to accomplish a task in minutes rather than days. IVEM is an excellent retrospective analysis tool that should be made available beyond the currently time consuming process.

## **II. Objective**

The aim of this project was to develop a functional program that implements IVEM and test this program's results against data provided by Dr. Ryan Otter. In order to accomplish this task, several other smaller objectives first needed to be completed.

The program needs a method to store and handle user collected data appropriately, but in order to do so that data must first be organized. Therefore inputting, storing, and processing data needs to start with users providing well-formatted data. Providing guidelines on how to format data is a requirement in both large- (How to Choose a Data Format) and small- (Best practices for file formats) scale projects. These guidelines must provide a formal system on how this project can expect to receive data,

and the details the steps necessary for proper operation. On the application-side, data storage must consider the order and amount of data received. Adding/removing data are user-facing objectives the program should meet, but internally, statistics from this information will need to be calculated, and the data will also need to be sorted accordingly. Information will need to be passed both in and out of this internal arrangement too. This internally arranged structure chiefly needs to represent an area that underwent testing with methods to access and manipulate all of its relevant data.

Creating and calculating results from data is the end goal of this project, but without internal organization, the data won't lead to a direct solution. It is not enough to have data arranged at a site level. The data must undergo standardization by using every recorded measurement at the test level. This is a fundamental requirement for an IVEM implementation. A "master" table that contains every recorded measurement will need to be made to calculate critical parameters. There also needs to be mechanisms to know when processed data is no longer accurate, and How? and When? these changes should occur or get updated. Finally, this table will have to be created or updated alongside each site data object on creation.

This project also aims to create an informative, yet easy to read graph. Providing a legend for the plotted regions, appropriate labels for each test, and sorted tabular results are an essential requirements for a general plot. However, unlike a general plot, IVEM expects and addresses zone ambiguity, so the plot will seek easy identification of these zones, the amount of uncertainty, and its overall significance. It is not an aim of the

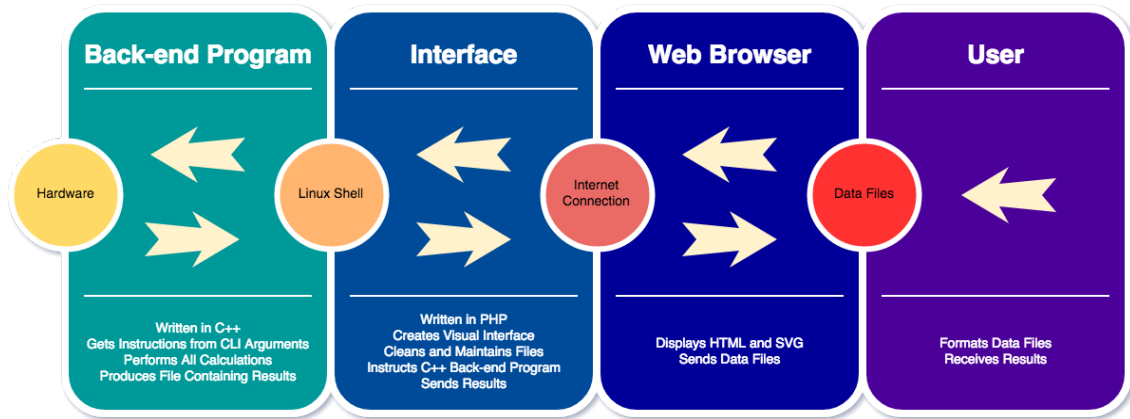
project to adjust the size of the graph, but it will provide a reasonably sized output for simplicity. Third-party tools might be used to satisfy this requirement.

Finally, general principles related to application development are important too. Development for cross-platform compatibility is a requisite not only graphical output formats, but benefits users on different platforms. A successful implementation also makes provision for expansion and experimental features, so development will also seek code modularity where it can be used most fittingly. Furthermore, in the application's creation opportunities for object oriented paradigms are hoped to be explored. One way to accomplish this goal is to keep "hard-coded" portions to a minimum. Another aim will explore C++ inheritance semantics. The project is intended to be a single use program and will not require anything outside of providing formatted data.

### **III. Methods & Materials**

The project was approached and designed as a web service with three distinct processing layers. The first layer consists of the core IVEM implementation as a C++ back-end program. This program is the "last man" in the process chain and performs all the calculations to generate the final result. The second layer is an interface script. This interface script was written in PHP and acts as an interface layer. On one side, the script executes the back-end program passing a numerical parameter on how to proceed. On the other side, the script provides the graphical interface to the user. The third layer, a web browser, allows the user to send data files to the application via the PHP script, receive the results, and obscures the underlying complexity of the program. See the diagram below.





**Figure 1.** The user formats and provides data files via a web browser. The data files are received by a server that supplied the interface. The interface executes a back-end program to perform the calculations and create a report. When the program terminates, the script renders the report to the user.

Using web technologies avoided many platform-specific issues and greatly simplified the project. In this paradigm, the user is expected to use an HTML 5 conforming web browser rather than a specific device. Building IVEM this way did add an additional requirement, but a trivial one that enabled the application to be truly platform agnostic. It also ensured a uniform experience. Furthermore, this requirement affords IVEM's to be accessible on mobile devices (i.e field tablets) which was not initially considered.

PHP is a scripting language for creating dynamic web content and an excellent candidate for managing the web interface for IVEM. Websites such as Facebook use PHP to fetch and present information on a user-by-user basis (MacKay, et. al. 2017). For this project, PHP responsibilities were rather elementary: It handled user files, ran the backend program, and renders a page with the results. As pointed out by MacKay (et al., 2017), PHP's syntax has the familiarity of C++ which reduced the time needed to

produce a working interface. Other technologies such as Scalable Vector Graphics (SVG) and Bootstrap were respectively used for the graphical plot and interface styling in conjunction with the PHP script.

All the code needed to facilitate the service was executed on a server. The PHP script allows a user to upload a file via the web browser and manages a counter to keep track of how many files it receives. Each file the user uploads represents an area of interest and the associated test data as stated in the guidelines. After receiving a file, each file is renamed consecutively starting at 0 (ex. “0.dat”, then “1.dat” ... and so on). When the script receives the command to process the data, it executes the C++ back-end program with a numeric parameter. For example, let’s say ‘3’ was passed as a parameter. This implies that there were three regions that were investigated and three files were uploaded by the user. The backend program now executes while the script waits for the program to complete. Because the script renamed all the user files consecutively, the backend program can expect the files 0.dat, 1.dat, and 2.dat to exist (no more than “3.dat”). The back-end program executes under this assumption, processes each file, and creates a result file before it terminates. Once the PHP script receives the termination signal from the back-end program, it checks for the existence of the result file and finishes the request by sending the results. A reset function deletes the user files, result file, and sets the file counter back to zero.

The backend program performs all the mathematical operations to produce a result. Those operations must be performed on data it retains in quickly accessible memory and not the files themselves; the program parse these files into an object which

represents a dataset. The eponymously named object, a **Dataset** object, retains this data and provides an interface to control the data indirectly for operations such as adding, removing, organizing, querying, and manipulating those values as well as generating statistics about the data. A **Site** object retains **Dataset** objects, and **Site** objects interface with each other for instructions on how to manipulate their respective datasets. The underlying container to hold data in a **Dataset** is the STL's list class, and the **Dataset** object directly operates on this list. The STL is a well vetted, open source, ISO specified collection of tools that developers should use for best practices. The STL list was selected for its flexibility in data storage.

As mentioned earlier, each file submitted represents a sampled region and its test data. The backend program's internal representation of these sampled regions are called **SampledSite** objects and are created using C++'s inheritance system (**Site** is the parent class). As each file is parsed, the back-end program instantiates these **SampledSite** objects. Each object has its own properties that only pertain to the region. When the first **SampledSite** object is created, another special object called a **MasterSite** is also created using the inheritance system. As additional sites are added, their data is also merged into the **MasterSite**'s datasets. The **MasterSite** object therefore contains all data from all tests conducted at every sampled region. Once all data is read in, the data is sorted and information is needed for all **SampledSite** objects to normalize their data. The **MasterSite** object provides information on how to do so. Each **SampledSite** object standardizes its datasets using the

MasterSite's information. Finally, with the data in a standardized form, results can be calculated and plotted.

The C++ back-end program needs to output a Scalable Vector Graphics image as part of its duties. Scalable Vector Graphics or SVG is a technology for developing high-quality images using vectors, lines, and shapes. Most, if not all, of SVG's shapes rely on numerical parameters. SVG's use a Cartesian plane coordinate system to render these shapes, but the representation of this plot is better suited in a polar coordinate system. For example, if  $n$  tests were conducted across different regions, then the angle for any endpoint would be some multiple of  $\frac{2\pi}{n}$  on a polar system. This computes a  $\theta$  value. The magnitude then becomes the test's mean or error values.

The resulting values can be connected to form a polygon. The resulting area inside that polygon represents the degree of damage at that region. Each polygon generated shares a common origin from which it was derived from: the origin in the polar system. Using this point, the angle formed from the two neighboring endpoints ( $\theta$  from above), and the magnitude values as distances, the polygon can be split into triangles. The area for these triangular segments can be computed using the trigonometric side-angle-side formula:

$$Area = \frac{Test1 \cdot Test2 \cdot \sin(\theta)}{2}$$

If we let the  $E$  be the standardized values for  $k$  tests, and let  $n$  represent the number of endpoints or tests, then algorithmically calculating the area of the polygon is just the sum of these triangles:

$$A = \sum_{k=2}^n \frac{E_{k-1} \cdot E_k \cdot \sin(\frac{2\pi}{n})}{2}$$

This equation fails to calculate the area for a triangle between the  $n^{\text{th}}$  and  $1^{\text{st}}$  endpoint. The missing “slice” can be calculated and added to compute the complete area of the polygon:

$$Area = A + \frac{E_1 \cdot E_n \cdot \sin(\frac{2\pi}{n})}{2}$$

Using this approach, the area of all polygons can be computed iteratively. This process was applied for the mean as well as the upper and lower bound standard error polygons. Even though this method is an effective way to determine the area of a polygon, no actual polygon exists yet; the polygon is still a virtual abstraction of the data at this point.

Instead of using a 3rd party library to provide plotting, a less practical method was selected as an exercise. Text holding variables (called “strings”) were used to create an actual SVG image from scratch. SVG follows a format that’s similar to how HTML marks-up web pages; rather than defining a document pixel by pixel, an element or “tag” is used. Field/value pairs are used to describe the tag’s attributes. For this project `Line` and `Path` were the predominately used tags. A possible `line` tag could look like this:

```
<Line x1="1" y1="2" x2="10" y2="20" stroke="black" />
```

The fields `x1` and `y1` define the starting (x, y) coordinate for the line, and fields `x2` and `y2` defining the ending (x, y) coordinate. Needless to say, `stroke` defines the color of the line. Other field/value pairs exist and can be used to provide additional attributes. In this example, a black line runs from the point (1, 2) to (10, 20) on the canvas of the SVG. It uses the top-left corner as the origin in a Cartesian system.

A simple way to adapt IVEM's data is to convert it to the Cartesian plane coordinate system. These conversions are fairly easy and the back-end program can do these conversions on the fly. The conversions are:

$$x = r \cdot \cos(t \cdot \theta), \quad y = r \cdot \sin(t \cdot \theta)$$

Using this approach, an  $(x, y)$  coordinate can be created from the statistical result  $(r)$  and a number, say  $t$  for the test number, to get the amount for the angle needed. The implementation present here used partially defined strings as templates and exploited the numerical fields in both `line` and `path` tags to create the shapes needed. For example, the line for an endpoint needs to span out using two pieces of information, an angle and a magnitude from some reference point and direction. Since these rotation and magnitude values were readily from the area calculations,  $r$  and  $t \cdot \theta$ , this data was reused. The  $(x, y)$  coordinate was calculated using the method described above to plot the shapes onto an SVG canvas. `line` tags were used to plot endpoints and the `path` tag was used for the legend, mean, and error bounds. A similar method was used on `text` tags for labeling. Tabular data followed the same approach, but used a table and the area results directly.

## IV. Results

A case study using fictitious data was prepared and used to validate the IVEM implementation. Fabricated data was compiled by Dr. Ryan Otter as datasets that one might observe in the field (see Appendix A for the data used). This specifically included four contaminated zones and a fifth pre-disaster reference zone. Each zone included five different test sets:

- 7-ethoxy-resorufin *O*-deethylation (EROD), a liver enzyme commonly used as a pollutant marker found in pigs and fish (Zamaratskaia, Vladimir 2009). This test is measured in pmol/mg/min.
- A ground sediment test that measures the concentration of a contaminant in mg per kg of soil.
- A comet assay test that “...has applications in testing novel chemicals for genotoxicity, monitoring environmental contamination with genotoxins, human biomonitoring and molecular epidemiology...” (Collins 2004). This test measures DNA damage as a percentage.
- A gene test that measures the expression of a given gene in response to the environment (Choi and Sang 2007). The test is measured as a simple count of occurrences.
- A concentration test that includes a direct mg/kg concentration of the of a substance in fish.

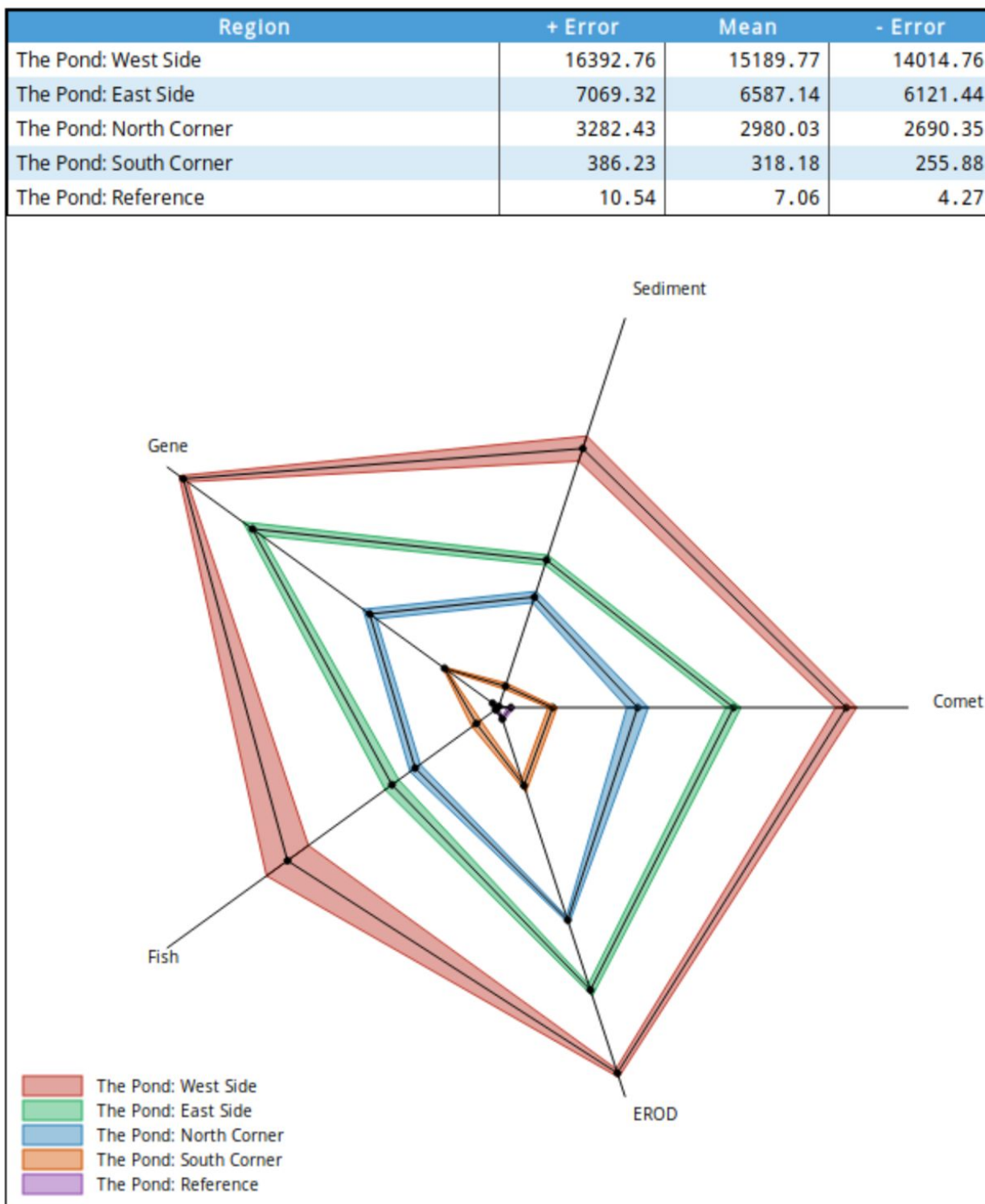
Each test measured a completely different feature one might wish to observe in a contaminated region. It is important to note that while the sediment and fish test share the same unit of measurement (mg/kg), they were designed to simulate measuring different compounds with different concentration ranges. The rest of the tests have different units of measurements, but all tests cannot be directly compared against each other.

Most datasets contained ten data points per region. The sediment test included twenty data points with the “South Corner” including an additional two points. The

South Corner also included an additional two points in its gene test. The purpose of these additional data tested IVEM against uneven sampling in both test-to-test and region-to-region scenarios. For example, the sediment test contains twenty points where other tests contained ten (test-to-test: The amount of data in each test differs), and the South Corner contained two additional data points in both gene and sediment samples while others sites did not (region-to-region: The amount of data in a region differs from the others). Notice that South Corner specifically contained an additional two data in the sediment test as well which caused both region-to-region and test-to-test amounts to differ simultaneously. Finally, some data were purposely manipulated to be artificially high or low to increase variation which might result from equipment errors, bad data collection, or by sheer chance.

During development, the arithmetic mean, standard deviation, and standard error from the mean were calculated using the back-end program and checked against precomputed values for each region. With each calculation of raw data measuring up correctly, the data and descriptive statistics were standardized. If everything performed accordingly in the implementation, the program should predict normalized values for both data and statistics. Using each dataset as the final testing case, the implementation did predict accurate results. Furthermore, the program correctly predicted the area of each polygon, including the standard error boundaries from manual calculations. Previously, no working model provided a visual output so testing this project presented an opportunity to see a true to life example. Figure 2 is the result from the data contained in appendix A.





**Figure 2.** The graphical output that resulted from the data in Appendix A. Each zone was designed to be progressively worse than the last. Notice the large variance in The Pond: West Side's fish test as a result from higher variation. The reference site (inter rmost site) barely registers a polygon due in the result due to background contaminations.

## V. Discussion

The application's functionality, interface, and graphical output are both intuitive and straightforward even for non-technical users. The resulting numerical data contains no unit of measurement, and can be thought of as a score where higher values signify worsening conditions. The numerical data is also sorted with respect to mean values and places the worst location at the top of the list. One does not need to know the statistics and math used to create the plot, but the result provides enough details to say that some amount of error can be expected both visually and numerically. Error boundaries are plotted with slightly darker colored lines surrounding the solid black line. This represents the mean measurement from the data with the standard error range filled-in between the colored boundary lines. The plot is designed such that someone can quickly gauge a test, set of tests, or a location as a whole and quickly determine its reliability despite the absence of numerical values. The color-filled error range is also semi transparent so overlapped regions are distinctly visible.

Each test ranged contained different value ranges and the best and worst test did not reach the outermost and innermost values on each endpoint. This was not expected, but after careful examination it was concluded this was an appropriate outcome. Each dataset was normalized using all data for a given test, but a dataset can only contain two extreme values because there can only be one maximum and minimum value in the entire collection. Even if the dataset contains a single entry, that entry is paradoxically both the minimum and the maximum value yet the test still contains a single extreme value.

Furthermore, if a test contains both the minimum and maximum value from the entire dataset, then it can only skew the mean and broaden the error boundary. Therefore plotted data will never reach values zero or one at the extremities of each endpoint. It was also not expected to see a sharp contrast between each site as well. Anecdotaly manipulating the data to overlap more and more still resulted with plots tending to “pull the zones apart.” This too was not expected, but again happens as data is normalized, the property mention above, and the result imposed onto each endpoint.

Moving away from a software prototype, other technologies such as the Python programming language would have dramatically simplified the development of IVEM into a more stable web application. An implementation in Python would merge both the web interface and IVEM program calculations into one cohesive application. Python has several hundred libraries (*Python Package Index*) that can add the functionality needed to replace the current prototype. For example, Python’s Flask library is “a micro-framework” that is better suited web framework than the current PHP script. It would combine both the web server, interface, and the app itself into the same code. SciPy and NumPy (“Sci” for science, “Num” for numbers, and “py” for Python) offers an extensive collection of science related functions ("SciPy.org" 2018), and array/matrix data structure with commonly used statistical calculations (“NumPy.” 2018). These libraries and methods were not known over the course of the projects development. However, the implementation did use C++’s Standard Template Library (STL) that follows best practices for code portability.

Future work to this implementation could remove some assumptions and limitations made at the model and code level. For example, the code made provisions for inverted data, but did not provide this functionality. In the case of inverted data, lower scores mark increasing damage (on the contrary, normal tests treat higher scores as increasing damage). Currently more testing and development needs to be conducted to remove this limitation. One assumption made is that all data scales linearly, but some tests such as the Richter Scale are non-linear. For example, a value of 3 is ten times the intensity of the value 2, and the value 2 is ten times the intensity of the value 1 in this non-linear scale. The impact of non-linear tests on IVEM is not known, and no work in this implementation exists for non-linear datasets. However, one could either scale the data before entering it, the application could correct it on the user's behalf, or it may simply be the case IVEM can handle non-linear datasets.

Appendix B shows the current file format with the letter "A" after each test name. This "A" designation is currently a fixed parameter and symbolizes that the dataset is in an ascending order (linear and non-inverted), however other letters might symbolize different data formats:

- **I:** Data is inverted and should be treated as such.
- **T:** TAB is the delimiter for each datum.
- **CSV:** The data follows a comma separated value format.
- **L<n>:** The data is in a logarithmic scale. An optional value  $n$  inside "<" and ">" could specify a custom base.
- **L<e>:** Data follows the natural logarithmic scale. Same as LN or  $\text{Log}_e$

At the beginning of this article, an IVEM implementation was explored using common programming tools to develop a proof of concept. This article has shown IVEM implemented as a web application with a graphical interface which is easy to understand. Likewise the output is easy to understand as well. This was a critical part of the model because ultimately IVEM is targeted toward non-scientific entities where a simple but expressive result is needed. To date no implementation of IVEM existed, but this project offers a complete and working example that one might expect using curated but realistic data. This study did not implement some functionality, but instead presented IVEM as a usable, multi-platform, and a highly accessible application. From this prototype, future work should consider programming environments more suitable for web development and frameworks better suited for plotting graphs. A more reliable application would consider Python as a framework to solidify this modeling tool. In order for this project to be taken more seriously, real data could be used and compared with opinions from the scientific community with graphs presented to policy makers.

## Appendix A

Raw data for each datasets that produced Figure 2 and associated arithmetic means, standard error, and standard deviation for each test. Standard deviation is based on the sample and not the standard deviation of the population.

### The Pond: West Side

<b>Fish</b> <b>(mg/kg)</b>	<b>Gene</b> <b>(count)</b>	<b>Comet</b> <b>(percent)</b>	<b>EROD</b> <b>(pmol/mg/min)</b>	<b>Sediment</b> <b>(mg/kg)</b>	
14	409	42	269	75	145
34	450	57	300	120	98
26	400	50	290	80	97
18	424	47	280	75	86
17	438	46	270	110	112
18	449	43	275	101	104
29	418	51	285	89	103
22	444	49	282	76	100
14	431	47	276	156	100
20	419	53	292	135	101

	<b>Fish</b>	<b>Gene</b>	<b>Comet</b>	<b>EROD</b>	<b>Sediment</b>
<b>Mean</b>	21.200	428.200	48.500	281.900	103.150
<b>Std. Dev.</b>	2.086	17.100	4.528	9.994	22.260
<b>Std. Err.</b>	6.596	5.407	1.432	3.160	4.977

### The Pond: East Side

<b>Fish</b> <b>(mg/kg)</b>	<b>Gene</b> <b>(count)</b>	<b>Comet</b> <b>(percent)</b>	<b>EROD</b> <b>(pmol/mg/min)</b>	<b>Sediment</b> <b>(mg/kg)</b>	
10	350	35	220	67	46
12	300	30	215	62	49
13	412	34	210	56	60
14	318	33	202	59	60
9	300	41	200	58	62
8	312	30	215	51	67
8	305	31	245	47	59
13	390	32	235	60	64
10	350	33	222	70	68
11	325	32	226	39	72

	<b>Fish</b>	<b>Gene</b>	<b>Comet</b>	<b>EROD</b>	<b>Sediment</b>
<b>Mean</b>	10.800	336.200	33.100	219.000	58.800
<b>Std. Dev.</b>	2.150	38.955	3.213	13.960	8.679
<b>Std. Err.</b>	0.680	12.319	1.016	4.415	1.941

**The Pond: North Corner**

<b>Fish</b> <b>(mg/kg)</b>	<b>Gene</b> <b>(count)</b>	<b>Comet</b> <b>(percent)</b>	<b>EROD</b> <b>(pmol/mg/min)</b>	<b>Sediment</b> <b>(mg/kg)</b>	
9	200	18	175	50	42
8	210	15	150	34	51
10	193	13	160	67	32
11	180	20	165	34	51
7	170	20	164	36	29
6	112	19	163	39	50
7	188	18	161	45	62
8	170	23	171	42	54
9	200	26	180	44	43
10	180	28	172	42	34

	<b>Fish</b>	<b>Gene</b>	<b>Comet</b>	<b>EROD</b>	<b>Sediment</b>
<b>Mean</b>	8.500	180.300	20.000	166.100	44.050
<b>Std. Dev.</b>	1.581	27.398	4.619	8.621	9.987
<b>Std. Err.</b>	0.500	8.664	1.461	2.726	2.233

**The Pond: South Corner**

<b>Fish</b> <b>(mg/kg)</b>	<b>Gene</b> <b>(count)</b>	<b>Comet</b> <b>(percent)</b>	<b>EROD</b> <b>(pmol/mg/min)</b>	<b>Sediment</b> <b>(mg/kg)</b>	
2	100	9	75	8	3
3	90	8	72	10	5
4	80	7	67	12	6
3	70	11	56	17	3
2	60	9	54	8	2
5	75	7	58	6	7
2	85	12	45	5	9
1	89	6	80	15	8
1	85	7	90	20	12
1	85	8	45	1	15
	75			3	5
	75				

	<b>Fish</b>	<b>Gene</b>	<b>Comet</b>	<b>EROD</b>	<b>Sediment</b>
<b>Mean</b>	2.400	80.750	8.400	64.200	8.600
<b>Std. Dev.</b>	1.350	10.550	1.897	15.083	5.205
<b>Std. Err.</b>	0.427	3.045	0.600	4.770	1.164

**The Pond: Reference**

<b>Fish (mg/kg)</b>	<b>Gene (count)</b>	<b>Comet (percent)</b>	<b>EROD (pmol/mg/min)</b>	<b>Sediment (mg/kg)</b>	
0.67	12	1	17	0.1	0.3
0.3	16	2	15	0.3	0.8
0.8	20	4	22	0.5	0.2
0.2	9	3	21	0.7	0.9
0.9	15	2	12	0.4	0.5
0.5	22	1	18	0.8	0.4
0.4	29	4	10	0.2	0.5
0.5	17	5	9	1.0	0.3
0.3	16	3	9	0.45	0.8
0.5	13	2	5	0.67	0.9

	<b>Fish</b>	<b>Gene</b>	<b>Comet</b>	<b>EROD</b>	<b>Sediment</b>
<b>Mean</b>	0.507	16.900	2.700	13.800	0.567
<b>Std. Dev.</b>	0.226	5.666	1.337	5.673	0.268
<b>Std. Err.</b>	0.071	1.792	0.423	1.794	0.060



## Appendix B

### Integrated Visualization & Endpoint Modeling File Specification

The Integrated Visualization & Endpoint Modeling (IVEM) uses plain text files for processing data. Text files with encodings (such as .DOC, .RTF, .ODT) will cause an error in processing and produce incorrect results or cause the program to crash. You can create input files on any operating system from commonly used programs including:

- **Windows machines:** Notepad, Notepad++.
- **macOS:** TextEdit may be used if you remove encode by pressing SHIFT + COMMAND + T or selecting Format > Make Plain Text from the global menu bar.
- **Cross Platform:** The Atom editor is recommend on any operating system if the file is saved with a '.txt' extension.

When creating input files, the first line represents the name or a region, site, or area of interest and must be unique. Each file represents a site or region with test data following afterwards. Test data must have a name, followed by capital letter 'A' on the next line. The unit of measurement follows the next line. The name will helps the user identify the test on the plot, the 'A' is reserved for future use, and the unit of measurement may be:

- **Single unit:** Specified directly such as Percent, Count, '%', or alpha.
- **Compound units:** No more than three units using a forward slash '/' as a delimiter. For example mg/kg, or pmol/mg/kg.

The order in which test data is listed in the file does not matter – the application will resolve tests automatically. The letter 'A' must be placed at the end of the unit of measurement for all dataset if they are to be compared. All test data enter below unit of measurement is delimited by a new line. Data may integers or rational numbers. The word 'END' is placed at the end of the data to signal the end of the dataset. The line after END must be blank before specifying a new test. Here is an example for testing two location using two test:

Filename: "ID 0123-4a.txt"

Cannonville Lake, South  
EROD  
A  
pmol/mg/kg  
13.2  
12.3  
. . .  
15.2  
15  
END

Fish Contamination  
A  
Percent  
3  
4.2  
. . .  
3.8  
12.3333  
END

Filename: "123abc.dat"

Cannonville Lake, East  
Fish Contamination  
A  
Percent  
5.2  
4.2  
. . .  
1.3  
4.5  
END

EROD  
A  
pmol/mg/kg  
23.2  
14.2  
. . .  
4  
17.8  
END

The file name and extension have no restrictions. You must supply three or more tests in order to

## Appendix C

```
1 <<<<<< colors.dat
2 #27ae60
3 #2980b9
4 #8e44ad
5 #c0392b
6 #d35400
7 #f39c12
8 <<<<<< END
```

```
9 <<<<<< dataset-summary.cpp
10 // #include <boost/function.hpp>
```

```
11 #include "dataset-summary.h"
12 #include "dataset.h"
```

```
13 DatasetSummary::DatasetSummary(Dataset * dset)
14     : m_dsref(dset) {
15     d_init();
16     m_dsref = (dset) ? dset : nullptr;
17 }
```

```
18 DatasetSummary::DatasetSummary(sDatasetPtr dset) {
19     d_init();
20     m_dsref = (dset.get()) ? dset.get() : nullptr;
21 }
```

```

22 void DatasetSummary::d_init() {
23     m_testname = &Dataset::name;
24     m_testid   = &Dataset::id;
25     m_testunit = &Dataset::unit;
26
27     m_inverted = &Dataset::inverted;
28     m_n        = &Dataset::size;
29
30     m_mean      = &Dataset::mean;
31     m_stddev_p  = &Dataset::stddev_p;
32     m_stddev_s  = &Dataset::stddev_s;
33     m_stdevr    = &Dataset::std_error;
34
35     m_min       = &Dataset::minimum;
36     m_q1        = &Dataset::q1;
37     m_median    = &Dataset::median;
38     m_q3        = &Dataset::q3;
39     m_max       = &Dataset::maximum;
40
41     m_iqr       = &Dataset::iqr;
42     m_range     = &Dataset::range;
43 }
44
45 TestSummary::TestSummary(Dataset* dset, SampledSite* site)
46 : DatasetSummary(dset) {
47     t_init();
48 }

```

```
43         m_siteref = (site) ? site : nullptr;
44     }
```

```
45     TestSummary::TestSummary(sDatasetPtr dset, SampledSite* site)
46     : DatasetSummary(dset) {
47         t_init();
48         m_siteref = (site) ? site : nullptr;
49     }
```

```
50     void TestSummary::t_init() {
51         m_n_upbound      = &Dataset::n_upper_bound;
52         m_n_mean         = &Dataset::norm_mean;
53         m_n_lwbound      = &Dataset::n_lower_bound;
54         m_standardized   = &Dataset::standardized;
55         m_normalize      = &Dataset::normalize;
56     }
57     <<<<<< END
```

```
58     <<<<<< dataset-summary.h
59     #ifndef DATASUMMARIES_H
60     #define DATASUMMARIES_H
61     /*
62     #include <string>
63     #include <memory>
```

```

64 #include <boost/function.hpp>

65 #include "dataset.h"
66 #include "sitesample.h"

67 class Site;
68 class DatasetSummary;

69 typedef std::shared_ptr<Dataset> sDatasetPtr;
70 typedef DatasetSummary * DatasetSummaryPtr;

71 using std::string;
72 using boost::function;

73 // Standard info thats in every dataset should, to be handed off to a site.
74 // this acts as a thin layer between accessors up the class structure.
75 class DatasetSummary {
76 public:
77     DatasetSummary() {};
78     DatasetSummary(Dataset * dset);
79     DatasetSummary(sDatasetPtr dset);

80     string test_name() const { return (m_dsref) ? m_testname(m_dsref) : ""; }
81     int test_id() const { return (m_dsref) ? m_testid(m_dsref) : -1; }
82     string test_unit() const { return (m_dsref) ? m_testunit(m_dsref) : ""; }

83     bool inverted() const { return (m_dsref) ? m_inverted(m_dsref) : false; }
84     int n() const { return (m_dsref) ? m_n(m_dsref) : 0; }

```

```

85     double mean() const           { return (m_dsref) ? m_mean(m_dsref)           : 0; }
86     double std_dev_s() const      { return (m_dsref) ? m_stddev_s(m_dsref)        : -1; }
87     double std_dev_p() const      { return (m_dsref) ? m_stddev_p(m_dsref)        : -1; }
88     double std_err() const        { return (m_dsref) ? m_stder(m_dsref)           : -1; }

89     double minimum() const        { return (m_dsref) ? m_min(m_dsref)           : 0; }
90     double q1() const             { return (m_dsref) ? m_q1(m_dsref)             : 0; }
91     double median() const         { return (m_dsref) ? m_median(m_dsref)         : 0; }
92     double q3() const             { return (m_dsref) ? m_q3(m_dsref)             : 0; }
93     double maximum() const        { return (m_dsref) ? m_max(m_dsref)            : 0; }
94     double iqr() const            { return (m_dsref) ? m_iqr(m_dsref)            : 0; }

95     double range() const          { return (m_dsref) ? m_range(m_dsref)          : 0; }

96     protected:
97     Dataset* m_dsref;

98     // Dataset hooks
99     function<string(Dataset*)> m_testname;
100    function<int(Dataset*)> m_testid;
101    function<string(Dataset*)> m_testunit;

102    function<bool(Dataset*)> m_inverted;
103    function<int(Dataset*)> m_n;

104    function<double(Dataset*)> m_mean;
105    function<double(Dataset*)> m_stddev_s;

```

```

106 function<double(Dataset*)>> m_stddev_p;
107 function<double(Dataset*)>> m_stder;
108 function<double(Dataset*)>> m_min;
109 function<double(Dataset*)>> m_q1;
110 function<double(Dataset*)>> m_median;
111 function<double(Dataset*)>> m_q3;
112 function<double(Dataset*)>> m_max;
113 function<double(Dataset*)>> m_iqr;
114 function<double(Dataset*)>> m_range;
115 private:
116     // Helper Methods
117     void d_init();
118 };
119 // extend the DatasetSummary class to include site specific data.
120 class TestSummary
121     : public DatasetSummary {
122 public:
123     TestSummary(Dataset*, SampledSite*);
124     TestSummary(sDatasetPtr, SampledSite*);
125     string site_name() const { return (m_siteref) ? m_sitename(m_siteref) : ""; }
126     int site_id() const { return (m_siteref) ? m_siteid(m_siteref) : -1; }

```



```

127 double upper_bound() const { return (m_dsref) ? m_n_upbound(m_dsref) : 0; }
128 double norm_mean() const { return (m_dsref) ? m_n_mean(m_dsref) : 0; }
129 double lower_bound() const { return (m_dsref) ? m_n_lwbound(m_dsref) : 0; }

130 bool standardized() const { return (m_dsref) ? m_standardized(m_dsref) : false; }
131 void normalize() const { if (m_dsref) m_normalize(m_dsref); }

132 protected:

133 private:
134 void t_init();
135 Site* m_siteref;

136 // Test & Site Specific hooks
137 function<double(Dataset*)> m_n_upbound;
138 function<double(Dataset*)> m_n_mean;
139 function<double(Dataset*)> m_n_lwbound;
140 function<bool(Dataset*)> m_standardized;
141 function<void(Dataset*)> m_normalize;

142 // Site Specific
143 function<string(Site*)> m_sitename;
144 function<int(Site*)> m_siteid;

145 function<double(Site*)> m_lwe_outcome;
146 function<double(Site*)> m_est_outcome;
147 function<double(Site*)> m_upe_outcome;

```

```
148     };
149     */
150 #endif // !DATASUMMARIES_H
151 <<<<<< END
```

```
152 <<<<<< dataset.cpp
153 // Preprocessor System Directives
154 #include <iostream>
155 #include <cmath>
156 #include <iomanip>
157 #include <string>
```

```
158 // User Defined Directives
159 #include "dataset.h"
160 #include "types.h"
```

```
161 // Using directives
162 using std::cout;
163 using std::endl;
164 using std::cin;
165 using std::setw;
166 using std::setprecision;
167 using std::abs;
```

```
168 // Implementation
```

```
169 // Constructors & Destructor
```

```

170     Dataset::Dataset() {
171         m_size = 0;
172         m_unitcnt = 0;
173
174         init_stats();
175         stale_stats();
176
177         m_inverted = false;
178         m_normtype = SELF;
179     }
180
181     Dataset::Dataset(bool descending) {
182         m_size = 0;
183         m_unitcnt = 0;
184
185         init_stats();
186         stale_stats();
187
188         m_inverted = descending;
189         m_normtype = SELF;
190     }
191
192     Dataset::Dataset(const Dataset &to_copy) {
193         m_data = to_copy.m_data;
194         m_size = to_copy.m_data.size();
195         m_inverted = to_copy.m_inverted;

```

```
190     m_name = to_copy.m_name;
191     unit_setter(to_copy.unit());
192
192     init_stats();
193     stale_stats();
194
194     m_inverted = to_copy.m_inverted;
195     m_normtype = to_copy.m_normtype;
196     //calc_stats();
197 }
```

```
198     Dataset::Dataset(vector<double> data, string name, string unit, bool inverted) {
199         vector<double>::const_iterator iter;
200         DataNode node;
```

```
201         name_setter(name);
202         unit_setter(unit);
```

```
203         init_stats();
```

```
204         m_inverted = inverted;
205         m_normtype = SELF;
```

```
206         node.norm_value = 0;
207         for (double d : data) {
208             node.value = d;
```

```
209         m_data.push_back(node);
210     }

211     m_size = m_data.size();
212     stale_stats();
213     sortdata();
214     normalize();
215     rank();
216 }

217 Dataset::~Dataset() {
218 }

219 // Public Observers
220 string Dataset::unit() const {
221     string ret_string = m_units[0];

222     if (m_unitcnt > 1) {
223         ret_string += "/";
224         ret_string += m_units[1];
225     }

226     if (m_unitcnt > 2) {
227         ret_string += "/";
228         ret_string += m_units[2];
```

```

229     }
230     return ret_string;
231 }

232 // Main Interface
233 void Dataset::print() {
234     cout << "Dataset: " << Dataset::name();

235     if (m_normtype == EXTERNAL)
236         cout << " (STANDARDIZED)";

237     cout << endl;

238     cout << "Values |    Unit |    Rank |    N. Value |" << endl;

239     for(DataNode data : m_data ) {
240         cout << setw(6) << data.value;
241         cout << " | " << setw(8) << unit() << " | ";
242         cout << setw(6) << data.rank << " | ";
243         cout << setw(9) << data.norm_value << " | " << endl;
244     }
245     cout << endl;

246     cout << setw(12) << "Mean: ";
247     cout << setprecision(6) << setw(7) << mean();
248     cout << setw(14) << "St Dev: ";
249     cout << setprecision(6) << stddev_s();
250     cout << endl;

```

```

251     cout << setw(12) << "N. L Err: ";
252     cout << setprecision(6) << setw(7) << Dataset::n_lower_bound();
253     cout << setw(14) << "St Err: ";
254     cout << setprecision(6) << Dataset::std_error();
255     cout << endl;

256     cout << setw(12) << "Nm. Mean: ";
257     cout << setprecision(6) << Dataset::norm_mean();
258     cout << endl;

259     cout << setw(12) << "N. U Err: ";
260     cout << setprecision(6) << Dataset::n_upper_bound();
261     cout << endl;
262     cout << setw(12) << "Range: " << Dataset::range() << endl;
263     cout << "~~~~~" << endl;
264 }

```

```

265     double Dataset::basis() {
266         if (stale_basis) {
267             Dataset::c_basis();
268             stale_basis = false;
269         }
270         return m_basis;
271     }

```

```
272 double Dataset::mean() const {
273     if (stale_mean) {
274         Dataset::c_mean();
275         stale_mean = false;
276     }
277     return m_mean;
278 }
```

```
279 double Dataset::median() const {
280     if (stale_median) {
281         Dataset::c_median();
282         stale_median = false;
283     }
284     return m_median;
285 }
```

```
286 double Dataset::iqr() const {
287     return abs(q3() - q1());
288 }
```

```
289 double Dataset::q1() const {
290     if (stale_q1) {
291         Dataset::c_q1();
292         stale_q1 = false;
293     }
294 }
```



```
294         return m_q1;
295     }
```

```
296     double Dataset::q3() const {
297         if (stale_q3) {
298             Dataset::c_q3();
299             stale_q3 = false;
300         }
301         return m_q3;
302     }
```

```
303     double Dataset::minimum() const {
304         if (m_inverted) {
305             if (stale_max) {
306                 c_max();
307                 return m_intmax;
308             }
309             } else if (stale_min)
310                 c_min();
```

```
311         return (m_inverted) ? m_intmax : m_intmin;
312     }
```

```
313     double Dataset::maximum() const {
314         if (m_inverted) {
```

```
315         if (stale_min) {
316             c_max();
317             c_min();
318             return m_intmax;
319         }
320     } else if (stale_max)
321         c_max();
```

```
322     return (m_inverted) ? m_intmin : m_intmax;
323 }
```

```
324 double Dataset::stddev_p() const {
325     if (stale_stddev_p) {
326         Dataset::c_stddev_p();
327         stale_stddev_p = false;
328     }
329     return m_stddev_p;
330 }
```

```
331 double Dataset::stddev_s() const {
332     if (stale_stddev_s) {
333         Dataset::c_stddev_s();
334         stale_stddev_s = false;
335     }
336     return m_stddev_s;
337 }
```

```
338 double Dataset::std_error() const {
339     if (stale_stdererror) {
340         Dataset::c_stdererror();
341         stale_stdererror = false;
342     }
343     return m_stdererror;
344 }
```

```
345 double Dataset::sum() const {
346     if (stale_sum) {
347         Dataset::c_sum();
348         stale_sum = false;
349     }
350     return m_sum;
351 }
```

```
352 double Dataset::up_mean_err() {
353     if (m_inverted)
354         return mean() - std_error();
355     else
356         return mean() + std_error();
357 }
```

```
358 double Dataset::lw_mean_err() {
359     if (m_inverted)
360         return mean() + Dataset::std_error();
361     else
362         return mean() - Dataset::std_error();
363 }
```

```
364 // Public normalized/standardized data access
365 double Dataset::norm_mean() {
366     double value = mean() - m_normval;
367     return abs(value / m_normbasis);
368 }
```

```
369 double Dataset::norm_median() {
370     double value = median() - m_normval;
371     return abs(value / m_normbasis);
372 }
```

```
373 double Dataset::norm_q1() {
374     double value = q1() - m_normval;
375     return abs(value / m_normbasis);
376 }
```

```
377 double Dataset::norm_q3() {
```

```
378     double value = q3() - m_normval;
379     return abs(value / m_normbasis);
380 }
```

```
381 double Dataset::n_upper_bound() {
382     double value = up_mean_err() - m_normval;
383     return abs(value / m_normbasis);
384 }
```

```
385 double Dataset::n_lower_bound() {
386     double value = lw_mean_err() - m_normval;
387     return abs(value / m_normbasis);
388 }
```

```
389 // Public Mutators
390 void Dataset::normalize() {
391     m_normtype = SELF;
392     norm_helper();
393 }
```

```
394 void Dataset::standardize(double ref_value, double ref_basis) {
395     m_normtype = EXTERNAL;
396     m_normval = ref_value;
397     m_normbasis = ref_basis;
398     norm_helper();
}
```

```
399     }
```

```
400     void Dataset::insert(double to_insert) {  
401         DataNode ins;
```

```
402         ins.value = to_insert;  
403         ins.norm_value = 0;
```

```
404         // TODO: Insert in place to avoid whole set sorting:  $O(N)$  vs  $O(n \log n)$   
405         m_data.push_back(ins);  
406         sortdata();  
407         m_size++;  
408         rank();  
409         stale_stats();  
410         norm_helper();  
411     }
```

```
412     void Dataset::merge(const Dataset &_merge) {  
413         list<DataNode>::_const_iterator iter = _merge.m_data.begin();  
414         for (; iter != _merge.m_data.end(); iter++)  
415             this->m_data.push_back(*iter);
```

```
416         m_data.sort([](const DataNode & a, const DataNode & b) { return a.value < b.value; });  
417         m_size = m_data.size();  
418         stale_stats();  
419         rank();
```

```
420     norm_helper();
421 }
```

```
422 void Dataset::remove(int pos) {
423     // NYI
424 }
```

```
425 void Dataset::rank() {
426     list<DataNode>::iterator iter = m_data.begin();
427     int rank = 1;
428     for (; iter != m_data.end(); iter++, rank++)
429         iter->rank = rank;
430 }
```

```
431 void Dataset::unit_setter(string unit) {
432     char reader;
433     m_unitcnt = 1;
```

```
434     for (size_t i = 0; i < unit.size(); i++) {
435         reader = unit.at(i);
```

```
436         if (reader == ' ')
437             continue;
438         else if (reader == '/') {
439             m_unitcnt++;
```

```
440         continue;
441     }
442     if (m_unitcnt > 3)
443         break;
444     m_units[m_unitcnt - 1] += reader;
445 }
446 }

447 void Dataset::calc_stats() {
448     stale_stats();
449     c_basis();
450     c_sum();
451     c_mean();
452     c_median();
453     c_q1();
454     c_q3();
455     c_stddev_s();
456     c_stddev_p();
457     c_stderr();
458 }

459 // private
460 void Dataset::c_mean() const {
461     if (m_size != 0)
```



```

462         m_mean = sum() / m_size;
463     else
464         m_mean = std::numeric_limits<double>::quiet_Nan();
465     }

466     void Dataset::c_stddev_s() const {
467         list<DataNode>::const_iterator iter = m_data.begin();

468         double* nums = new double[m_size];
469         double tempsum = 0;
470         c_mean();

471         for (int i = 0; iter != m_data.end(); iter++, i++)
472             nums[i] = pow(iter->value - m_mean, 2);

473         for (int i = 0; i < m_size; i++)
474             tempsum += nums[i];

475         tempsum /= (m_size - 1);
476         m_stddev_s = sqrt(tempsum);
477         delete[] nums;
478     }

479     void Dataset::c_stddev_p() const {
480         list<DataNode>::const_iterator iter = m_data.begin();

```

```

481 double* nums = new double[m_size];
482 double tempsum = 0;
483 c_mean();
484
485     for (int i = 0; iter != m_data.end(); iter++, i++)
486         nums[i] = pow(iter->value - m_mean, 2);
487
488     for (int i = 0; i < m_size; i++)
489         tempsum += nums[i];
490
491     tempsum /= (m_size);
492     m_stddev_p = sqrt(tempsum);
493     delete[] nums;
494 }
495
496 void Dataset::c_stdevr() const {
497     m_stdevr = stddev_s() / (sqrt(m_size));
498 }
499
500 // m_min will hold the absolute minimum in the set
501 void Dataset::c_min() const {
502     if (m_inverted)
503         m_intmin = m_data.rbegin()->value;
504     else
505         m_intmin = m_data.begin()->value;
506     stale_min = false;
507 }

```

```
502     }
```

```
503     void Dataset::c_q1() const {
504         bool plus_one_form = false;
505         int n, midpoint;
506         double temp_q1;
507         list<DataNode>::const_iterator iter = m_data.begin();
```

```
508         if (m_size % 2) {
509             if (((m_size - 3) % 4)) {
510                 n = (m_size - 1) / 4;
511                 n--;
512                 plus_one_form = true;
513             }
514             else
515                 n = (m_size - 3) / 4;
516
517             // odd length total set
518             // of 4n+1
```

```
516         if (plus_one_form) {
517             for (int k = 0; k < n; k++)
518                 iter++;
```

```
519         temp_q1 = (iter->value * .25);
520         iter++;
521         m_q1 = temp_q1 + ((iter->value) * .75);
522     } else {
523         for (int k = 0; k < n; k++)
524             iter++;
```

```

525 temp_q1 = (iter->value * .75);
526 iter++;
527 m_q1 = temp_q1 + ((iter->value) * .25);
528 }
529 } else {
530     midpoint = m_size / 2;
531     if (midpoint % 2) {
532         midpoint /= 2;
533     }
534     for (int k = 0; k < midpoint; k++)
535         iter++;
536     m_q1 = iter->value;
537 } else {
538     midpoint /= 2;
539     midpoint--;
540     for (int k = 0; k < midpoint; k++)
541         iter++;
542     temp_q1 = iter->value;
543     iter++;
544     temp_q1 += iter->value;
545     m_q1 = (temp_q1 /= 2);
546 }
547 }
    stale_q1 = false;

```

// even length  
// odd n on lower half

```
548     }
```

```
549     void Dataset::c_median() const {  
550         list<DataNode>::const_iterator iter = m_data.begin();  
551         int midpoint = (m_size / 2);  
552         bool is_odd = (m_size % 2 == 1) ? true : false;
```

```
553         if (!is_odd)  
554             midpoint--;
```

```
555         for (int i = 0; i < midpoint; i++)  
556             iter++;
```

```
557         if (is_odd) {  
558             m_median = iter->value;
```

```
559         }  
560         else {
```

```
561             m_median = iter->value;  
562             iter++;
```

```
563             m_median = (iter->value + m_median) / 2;  
564         }
```

```
565     }
```

```
566     void Dataset::c_q3() const {
```

```
567         bool plus_one_form = false;
```

```
568         int n, midpoint;
```

```

569 double temp_q3;
570 list<DataNode>::const_reverse_iterator iter = m_data.rbegin();

571 if (m_size % 2) {           // odd length total set
572     if ((m_size - 3) % 4) { // of the form 4n+1? (then mod 4 == true)
573         n = (m_size - 3) / 4;           // swapped
574         plus_one_form = true;
575     }
576     else
577         n = (m_size - 1) / 4;           // 4n+3 form

578     if (plus_one_form) {
579         for (int k = 0; k < n; k++)
580             iter++;

581         temp_q3 = (iter->value * .25);           // Apply weights
582         iter++;
583         m_q3 = temp_q3 + ((iter->value) * .75); // Set Q3
584     } else {
585         for (int k = 0; k < n; k++)
586             iter++;

587         temp_q3 = (iter->value * .75);
588         iter++;
589         m_q3 = temp_q3 + ((iter->value) * .25);
590     }
591 } else {
592     midpoint = m_size / 2;           // even length

```

```

593         if (midpoint % 2) {
594             midpoint /= 2;
595
596             for (int k = 0; k < midpoint; k++)
597                 iter++;
598
599             m_q3 = iter->value;
600         } else {
601             midpoint /= 2;
602             midpoint--;
603
604             // even n on lower half
605
606             for (int k = 0; k < midpoint; k++)
607                 iter++;
608
609             temp_q3 = iter->value;
610             iter++;
611             temp_q3 += iter->value;
612             m_q3 = (temp_q3 /= 2);
613         }
614
615         stale_q3 = false;
616     }
617
618     // m_max will hold the absolute maximum in the set
619     void Dataset::c_max() const {
620         if (m_inverted)
621             m_intmax = (m_data.begin()->value;
622         else

```

```
616         m_intmax = (m_data.rbegin()->value;
617         stale_max = false;
618     }
```

```
619     void Dataset::c_sum() const {
620         m_sum = 0;
621         list<DataNode>::const_iterator iter = m_data.begin();
622         for (; iter != m_data.end(); iter++)
623             m_sum += iter->value;
624         stale_sum = false;
625     }
```

```
626     void Dataset::init_stats() {
627         m_normval = 0;
628         m_normbasis = 0;
```

```
629         m_basis = 0;
630         m_iqr = 0;
631         m_mean = 0;
632         m_median = 0;
633         m_q1 = 0;
634         m_q3 = 0;
635         m_intmin = 0;
636         m_intmax = 0;
637         m_stddev_s = 0;
638         m_stddev_p = 0;
```



```
639     m_stder = 0;
640     m_sum = 0;
641 }
```

```
642 void Dataset::stale_stats() {
643     stale_basis = true;
644     stale_mean = true;
645     stale_median = true;
646     stale_max = true;
647     stale_min = true;
648     stale_q1 = true;
649     stale_q3 = true;
650     stale_stddev_p = true;
651     stale_stddev_s = true;
652     stale_stder = true;
653     stale_sum = true;
654 }
```

```
655 void Dataset::sortdata() {
656     if (m_inverted)
657         m_data.sort([](const DataNode & lhs, const DataNode & rhs) { return lhs.value >
658             rhs.value; });
659     else
660         m_data.sort([](const DataNode & lhs, const DataNode & rhs) { return lhs.value <
661             rhs.value; });
662 }
```

```

663 void Dataset::norm_helper() {
664     if (m_normtype == SELF) {
665         m_normbasis = basis();
666         if (m_inverted) {
667             c_max();
668             m_normval = m_intmax;
669         }
670         else {
671             c_min();
672             m_normval = m_intmin;
673         }
674     }
675     if (m_inverted) {
676         for (DataNode d : m_data)
677             d.norm_value = abs((d.value - m_normval) / m_normbasis);
678     }
679     else {
680         for (DataNode d : m_data)
681             d.norm_value = (d.value - m_normval) / m_normbasis;
682     }
683 }
684 void Dataset::c_basis() {
685     m_basis = Dataset::range() / 100;

```

```

686         if (m_basis == 0)
687             m_basis = 1;

                                     // Don't divide by zero

688         stale_basis = false;
689     }
690 <<<<<< END

<<<<<< dataset.h
691 #ifndef DATALIST_H
692 #define DATALIST_H
693
694     // External Preprocessor Directives
695     #include <string>
696     #include <memory>
697     #include <list>
698     #include <vector>
699
700     // Internal Preprocessor Directives
701     #include "types.h"
702
703     // Using Statements
704     using std::shared_ptr;
705     using std::list;
706     using std::vector;
707     using std::string;

```

```

706 // Forward Declarations and typedef's
707 class Dataset;
708 struct DataNode;
709 typedef std::shared_ptr<Dataset> sDatasetPtr;
710 typedef Dataset* DatasetPtr;

711 class Dataset {
712 public:
713     // Constructors, Destructor
714     Dataset();
715     Dataset(bool descending);
716     Dataset(const Dataset &to_copy);
717     Dataset(sDatasetPtr to_copy) : Dataset(*to_copy) {};
718     Dataset(vector<double> data, string name, string unit, bool descending = true);
719     ~Dataset();

720     // Strict Observers
721     bool empty() const { return (m_size == 0); } // Returns if list is empty
722     int size() const { return m_size; } // Returns num of elements in list

723     bool inverted() const { return m_inverted; }
724     bool standardized() const { return (m_normtype == EXTERNAL); } //
725     string name() const { return m_name; } // Returns name of dataset
726     int id() const { return m_id; }
727     string unit() const; // Returns unit of dataset

728     // Amendable Observers
729     double up_mean_err(); // returns mean + std error

```

```

730 double mean() const;           // returns mean
731 double lw_mean_err();          // returns mean - std error

732 double stddev_p() const;       // returns std dev of a population
733 double stddev_s() const;
734 double std_error() const;      // returns std error of the mean

735 double minimum() const;        // returns smallest value
736 double q1() const;             // returns first quartile
737 double median() const;         // returns median
738 double q3() const;             // returns third quartile
739 double maximum() const;        // returns highest value
740 double iqr() const;            // returns Q3 - Q1

741 // Observer Mutators
742 void print();                  // Modifies stat values
743 double basis();                // returns basis_unit: Range / 100

744 double range() { return abs(maximum() - minimum()); } // returns max - min

745 double n_upper_bound();
746 double norm_mean();
747 double n_lower_bound();

748 double norm_q1();
749 double norm_median();
750 double norm_q3();
751 double norm_iqr() { return abs(norm_q3() - norm_q1()); }

```

```

752     double sum() const;                                // returns the sum of the values

753     // Mutators
754     void name_setter(string name) { m_name = name; }    // Set the name of the dataset
755     void id_setter(int _id) { m_id = _id; }
756     void unit_setter(string units);                      // Set the unit of the dataset

757     void insert(double to_insert);
758     void insert(const DataNode &to_insert) { insert(to_insert.value); }
759     void merge(const Dataset &merge);
760     void merge(sDatasetPtr _merge) { merge(*_merge); } // Merge & Sort a DataSet by shared_ptr
761     void remove(int position);                          // Remove data (*NYI*)

762     void normalize();
763     void standardize(double ref_value, double ref_basis);

764     void calc_stats();                                  // Refresh all internal stats

765     private:
766         mutable bool stale_mean;
767         mutable bool stale_stddev_p;
768         mutable bool stale_stddev_s;
769         mutable bool stale_stdeverror;

770         mutable bool stale_min;
771         mutable bool stale_q1;
772         mutable bool stale_median;

```

```

773 mutable bool stale_q3;
774 mutable bool stale_max;

775 bool stale_basis;

776 mutable bool stale_sum;
777 bool m_inverted;

778 int m_size;
779 int m_unitcnt;
780 unsigned int m_id;

// Number of items in this list

781 double m_basis;
782 double m_normval;
783 double m_normbasis;

// Internal observer data references
// value to norm. / stdz with
// basis value to norm/stdz with

784 mutable double m_intmin;
785 mutable double m_intmax;

786 mutable double m_mean;
787 mutable double m_stddev_p;
788 mutable double m_stddev_s;
789 mutable double m_stder;

790 mutable double m_q1;
791 mutable double m_median;
792 mutable double m_q3;
793 double m_iqr;

```

```

794     mutable double m_sum;

795     string m_name;           // The name or type of test conducted
796     string m_units[3];      // Stores compound units (mg/L/hr)

797     NORM_T m_normtype;

798     list<DataNode> m_data;    // The dataset

799     // Helper Methods
800     void c_mean() const;
801     void c_stddev_s() const;
802     void c_stddev_p() const;
803     void c_stdevr() const;

804     void c_min() const;
805     void c_q1() const;
806     void c_median() const;
807     void c_q3() const;
808     void c_max() const;

809     void rank();
810     void c_basis();          // Calcs basis_unit or uses basis_override if normalization_override

811     void c_sum() const;
812     void stale_stats();
813     void init_stats();       // Flag resetter

```



```
814     void norm_helper();
815     void sortdata();
816 };
817 #endif
818 <<<<<< END
```

```
819 <<<<<< ETP-main.cpp
820 // Preprocessor Directives
821 #include <iostream>
822 #include <vector>
823 #include <list>
824 #include <iomanip>
825 #include <fstream>
826 #include <cassert>
827 #include <string>
828 #include <time.h>
```

```
829 // User Defined Directives
830 #include "types.h"
831 #include "dataset-summary.h"
832 #include "dataset.h"
833 #include "site.h"
834 #include "sitesample.h"
835 #include "sitemaster.h"
836 #include "integrator.h"
837 #include "plotter.h"
838 #include "box-whisker.h"
```

```
839 // Using statements
840 using std::cout;
841 using std::string;
842 using std::vector;
843 using std::list;
844 using std::cin;
845 using std::endl;
846 using std::setprecision;
847 // Function Prototypes

848 // Main
849 int main() {

850     clock_t begin, end;
851     double time_spent;

852     vector<string> paths;
853     paths.push_back("site1.dat");
854     paths.push_back("site2.dat");
855     paths.push_back("site3.dat");
856     //paths.push_back("site4.dat");
857     //paths.push_back("site5.dat");
858     //paths.push_back("site3.dat");

859     begin = clock();
860     Integrator inte(paths);
861     inte.configure();
```

```

862     inte.show_results();
863     end = clock();

864     //string test = string();
865     //BoxWhisker::box_whisker(test, 20, 40, 70, 85, 88, 300);
866     //cout << test;

867     time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
868     cout << "done in: " << time_spent << " seconds" << endl;
869     return 0;
870 }
871 <<<<<< END

872 <<<<<< HTMLTable.cpp
873 #include "HTMLTable.h"
874 /*
875 using std::to_string;

876 HTMLTable::HTMLTable(int rows, int cols, string id)
877     : m_rows(rows), m_cols(cols) {

878     m_table.reserve(rows + 1);
879     m_headerset = false;

880     m_buffer = "<table id='" + id + "'>\n";
881     for (int i = 1; i <= cols; i++) {
882         m_buffer += "<colgroup class='" + id + "-" + to_string(i) + "'>< / colgroup>\n";

```

```

883         }
884     };
885     HTMLTable::~HTMLTable() {
886     }
887     void HTMLTable::set_header(const list<string> & headings) {
888         // _html += "<thead>\n";
889         // _html += "<tr class='header'>\n";
890         // _html += "<th class='header-label'><span>Region</span></th>\n";
891         // _html += "<th class='header-label'><span>+ Error</span></th>\n";
892         // _html += "<th class='header-label'><span>Mean</span></th>\n";
893         // _html += "<th class='header-label'><span>- Error</span></th>\n";
894         // _html += "</tr>\n";
895         // _html += "</thead>\n";
896
897         m_table[0] = headings;
898     }
899
900     void HTMLTable::set_row(int row, const list<string> & entries) {
901         if (row > m_rows || row == 0)

```

```
900         return;
901     else
902         m_table[row] = entries;
903 }
```

```
904 void HTMLTable::string_entry(string entry, string & out) {
905     out.clear();
906     out += "<td class='site-name'>";
907     out += entry;
908     out += "</td>\n";
909 }
910 */
911 <<<<<< END
```

```
912 <<<<<< HTMLTable.h
913 #ifndef HTML_TABLE_H
914 #define HTML_TABLE_H
915 /*
916 #include <string>
917 #include <vector>
918 #include <list>
```

```
919 using std::string;
920 using std::vector;
921 using std::list;
```

```
922 class HTMLTable {
923     public:
924         HTMLTable() {};
925         HTMLTable(int rows, int cols, string id);
926
927         ~HTMLTable();
928
929         void set_header(const list<string> & headings);
930         void set_row(int row, const list<string> & entries);
931
932         static void string_entry(string entry, string & out);
933         template <typename T> static void number_entry(T num, string & out) {
934             string temp = to_string(num);
935             std::size_t pos;
936
937             out.clear();
938             out += "<td class='num-result'>";
939             pos = temp.find('.');
940             if (pos != string::npos) {
941                 pos += 3;
942                 temp = temp.substr(0, pos);
943             }
944             out += temp;
945             out += "</td>\n";
946         }
947     private:
```

```

944     bool m_headerSet;
945     int m_rows;
946     int m_cols;

947     string m_buffer;
948     vector<list<string>> m_table;
949 };

950 */
951 #endif
952 <<<<<< END

953 <<<<<< integrator.cpp
954 #define _USE_MATH_DEFINES

955 // System Directives
956 #include<math.h>
957 #include<iomanip>

958 // User Defined Directives
959 #include "integrator.h"
960 #include "plotter.h"
961 #include "sitesample.h"
962 #include "sitemaster.h"

963 // Using Statements
964 using std::cout;

```

```

965 using std::endl;
966 using std::sin;
967 using std::setw;
968 using std::setprecision;

969 // Take a vector of file paths, create the integrator object that
970 // holds all the sites, the datasets of those sites, and creates a
971 // master listing from those sites.
972 Integrator::Integrator(const vector<string> &paths) {
973     vector<s_SampledSitePtr>::const_iterator sites_iter;
974     shared_ptr<SampledSite> sampled_site;

975     m_sitecount = 0;
976     stale_conf = true;

977     for (const string path : paths) {
978         sampled_site = s_SampledSitePtr(new SampledSite(path));
979         if (find_site(sampled_site->name()) == -1) {
980             m_sitenames.push_back(sampled_site->name());
981             m_sites.push_back(sampled_site);

982             m_sitecount++;
983         }
984         sampled_site.reset();
985     }
986     // Constructor MASTER reference.
987     sites_iter = m_sites.begin();
988     m_master = s_MasterSitePtr(new MasterSite(**sites_iter));

```



```
989     sites_iter++;
990     while (sites_iter != m_sites.end()) {
991         m_master->add_data(**sites_iter);
992         sites_iter++;
993     }
994     m_usable_ids.reserve(m_master->test_count());
995
996     assign_ids();
997 }
998
999     Integrator::~Integrator() {
1000         m_sites.clear();
1001         m_sitenames.clear();
1002
1003         m_master.reset();
1004     }
1005
1006     void Integrator::print_sites() const {
1007         for (const s_SampledSitePtr site : m_sites)
1008             site->print_tests();
1009     }
1010
1011     void Integrator::print_site(string site_name) const {
1012         for (const s_SampledSitePtr site : m_sites) {
1013             if (site->name() == site_name) {
```

```

1009         site->print_tests();
1010         break;
1011     }
1012 }
1013
1014 int Integrator::get_testcount() {
1015     if (stale_conf)
1016         configure();
1017     return m_master->test_count();
1018 }
1019 // Mutators
1020 // standardize every test in a site
1021 void Integrator::standardize(string site_name) {
1022     for (const DatasetPtr test : m_master->getMasterSets())
1023         standardize(site_name, test->name());
1024 }
1025 void Integrator::standardize(string site_name, string name_setter) {
1026     vector<s_SampledSitePtr>::iterator site_iter = m_sites.begin();
1027     DatasetPtr refer = m_master->get_test(name_setter);
1028     double r_value = refer->minimum();

```

```
1029     double r_basis = refer->basis();
1030
1031     while (site_iter != m_sites.end()) {
1032         if ((*site_iter)->name() == site_name) {
1033             (*site_iter)->standardize(name_setter, r_value, r_basis);
1034             break;
1035         }
1036         site_iter++;
1037     }
1038 }
```

```
1038     void Integrator::normalize(string site_name, string name_setter) {
1039         vector<s_SampledSitePtr>::iterator site_iter = m_sites.begin();
1040         while (site_iter != m_sites.end()) {
1041             if ((*site_iter)->name() == site_name) {
1042                 (*site_iter)->normalize(name_setter);
1043                 break;
1044             }
1045             site_iter++;
1046         }
1047     }
1048 }
```

```
1048     void Integrator::normalize(string site_name) {
1049         vector<s_SampledSitePtr>::iterator iter;
```

```
1050     for (iter = m_sites.begin(); iter != m_sites.end(); iter++) {
1051         if ((*iter)->name() == site_name)
1052             (*iter)->normalize();
1053     }
1054 }
```

```
1055 void Integrator::add_site(string path) {
1056     s_SampledSitePtr new_sampled(new SampledSite(path));
1057     add_site(new_sampled);
1058 }
```

```
1059 void Integrator::add_site(s_SampledSitePtr site) {
1060     if (Integrator::find_site(site->name()) == -1) {
1061         m_sites.push_back(site);
1062         m_sitenames.push_back(site->name());
1063         m_master->add_data(site);
1064         // if sites are standardized, they aren't now !!
1065         // THATS A BIG PROBLEM!
1066         // TODO: keep a list of names and only restandardize those sites.
```

```
1067         // OR... you can handle this dynamically...
1068         m_sitecount++;
1069     }
1070     stale_conf = true;
1071 }
```

```
1072 void Integrator::remove_site(string name) {
1073     int index = find_site(name);
1074     if (index == -1)
1075         return;
1076     else {
1077         m_sites[index].reset();
1078         m_sitecount--;
1079     }
1080     stale_conf = true;
1081 }
```

```
1082 void Integrator::configure() {
1083     vector<s_SampledSitePtr>::iterator site_iter = m_sites.begin();
1084     vector<DatasetPtr>::iterator ref_iter;
1085     vector<vector<TestResults>>::iterator iter;
1086     int unique_count = m_master->test_count();
1087     int test_id;
1088     double r_val;
1089     double r_basis;
1090     if (!m_tesres_adj.empty())
1091         m_tesres_adj.clear();
```

```

1092     determine_coverage();

1093     while (site_iter != m_sites.end()) {
1094         for (ref_iter=m_master->m_tests.begin(); ref_iter!=m_master->m_tests.end(); ref_iter++) {
1095             test_id = (*ref_iter)->id();
1096             if (is_usable(test_id)) {
1097                 m_master->std_refrc(test_id, r_val, r_basis);
1098                 (*site_iter)->standardize(test_id, r_val, r_basis);
1099             }
1100         }
1101         if ((*site_iter)->standardize_cnt() > 0) {
1102             (*site_iter)->generate_payloads();
1103             m_tesres_adj.push_back((*site_iter)->get_payloads());
1104         }
1105         site_iter++;
1106     }

1107     for (iter = m_tesres_adj.begin(); iter != m_tesres_adj.end(); iter++)
1108         calculate_site(*iter);

1109     stale_conf = false;
1110 }

1111 void Integrator::show_results() {
1112     if (stale_conf)
1113         configure();

```

```

1114     m_plotter = new Plotter(&m_tesres_adj,
1115                             &m_results,
1116                             m_theta,
1117                             m_usable_ids.size(),
1118                             &(*m_master));
1119
1119     m_plotter->plot_results();
1120     delete m_plotter;
1121     m_plotter = nullptr;
1122 }
1123
1123 // Private
1124 void Integrator::assign_ids() {
1125     vector<s_SampledSitePtr>::iterator sites;
1126     int test_count = m_master->test_count();
1127     int test_id = 0;
1128     string name;
1129
1129     for (int i = 0; i < test_count; i++) {
1130         m_master->id_enumerator(i, name);
1131         //get name based on id
1131         for (sites = m_sites.begin(); sites != m_sites.end(); sites++) {
1132             (*sites)->id_setter(test_id);
1133             (*sites)->id_enumerator(i, name);
1134         }
1135         test_id++;
1136     }
1137 }

```

```
1138 void Integrator::set_theta(int points) {
1139     m_theta = (2 * M_PI) / points;
1140     m_sine_theta = sin(m_theta);
1141 }
```

```
1142 void Integrator::calculate_site(vector<TestResults> &test_results) {
1143     double u_area = 0;
1144     double m_area = 0;
1145     double l_area = 0;
1146     double str_u, str_m, str_l;
1147     double lag_u, lag_m, lag_l;
1148     double lead_s;
```

```
1149     string site_name;
1150     int site_id;
```

```
1151     ResultsPtr site_result;
1152     vector<TestResults>::const_iterator iter = test_results.begin();
1153     site_name = iter->site_name();
1154     site_id = iter->site_id();
1155     lag_u = str_u = iter->upper_error();
1156     lag_m = str_m = iter->mean();
1157     lag_l = str_l = iter->lower_error();
1158     iter++;
```



```

1159 // Partial area calculation for the polygon
1160 while (iter != test_results.end()) {
1161     lead_s = iter->upper_error();
1162     u_area += .5 * lag_u * lead_s * m_sine_theta;
1163     lag_u = lead_s;
1164
1165     lead_s = iter->mean();
1166     m_area += .5 * lag_m * lead_s * m_sine_theta;
1167     lag_m = lead_s;
1168
1169     lead_s = iter->lower_error();
1170     l_area += .5 * lag_l * lead_s * m_sine_theta;
1171     lag_l = lead_s;
1172     iter++;
1173 }
1174 u_area += .5 * lag_u * str_u * m_sine_theta;
1175 m_area += .5 * lag_m * str_m * m_sine_theta;
1176 l_area += .5 * lag_l * str_l * m_sine_theta;
1177
1178 // That 0 needs to be fixed
1179 site_result = new Results(site_name, site_id, u_area, m_area, l_area);
1180 m_results.push_back(*site_result);
1181 delete site_result;
1182 site_result = nullptr;
1183 }

```

```
1181 void Integrator::determine_coverage() {
1182     vector<int>::iterator id_iter;    // an iterator to walk through the master site
1183     vector<DatasetPtr>::const_iterator master_iter = (m_master->getMasterSets()).begin();
1184     vector<s_SampledSitePtr>::const_iterator site_iter = m_sites.begin();
```

```
1185     while (master_iter != m_master->getMasterSets().end()) {
1186         m_usable_ids.push_back((*master_iter->id()));
1187         master_iter++;
1188     }
```

```
1189     while (site_iter != m_sites.end()) {
1190         id_iter = m_usable_ids.begin();
1191         while (id_iter != m_usable_ids.end()) {
1192             if (!(*site_iter->test_present(*id_iter))
1193                 id_iter = m_usable_ids.erase(id_iter);
1194             else
1195                 id_iter++;
1196             }
1197         site_iter++;
1198     }
1199     set_theta(m_usable_ids.size());
1200 }
```

```
1201 bool Integrator::is_usable(int _id) const {
1202     vector<int>::const_iterator iter;
1203     for (iter = m_usable_ids.begin(); iter != m_usable_ids.end(); iter++)
1204         if (_id == *iter)
```

```
1205         return true;
1206     return false;
1207 }
```

```
1208 int Integrator::find_site(string name) const {
1209     list<string>::const_iterator iter = m_sitenames.begin();
1210     int i = 0;
1211     while (iter != m_sitenames.end()) {
1212         if (*iter == name)
1213             return i;
1214         i++;
1215         iter++;
1216     }
1217     return -1;
1218 }
```

```
1219 /*
1220 void Integrator::maximize_accuracy() {
1221     } */
1222 <<<<<< END
```

```
1223 <<<<<< integrator.h
1224 #ifndef INTEGRATOR_H
1225 #define INTEGRATOR_H
```

```
1226 #define _USE_MATH_DEFINES

1227 // System Directives
1228 #include <math.h>
1229 #include <iostream>
1230 #include <memory>
1231 #include <algorithm>
1232 #include <set>

1233 // User Defined Directives
1234 #include "site.h"
1235 #include "sitemaster.h"
1236 #include "sitesample.h"
1237 #include "types.h"

1238 // Using Statements
1239 using std::set;
1240 using std::string;
1241 using std::shared_ptr;

1242 class Integrator;
1243 class Plotter;

1244 typedef Integrator * IntegratorPtr;
1245 typedef shared_ptr<Integrator> s_IntegratorPtr;
1246 typedef Plotter * PlotterPtr;

1247 class Integrator {
```

```
1248 public:
1249     // Constructors & Destructor
1250     Integrator() {};
1251     Integrator(const vector<string> &paths);
1252     ~Integrator();

1253     // Observers
1254     void print_sites() const;
1255     void print_site(string site_name) const;
1256     void print_master() const { m_master->print_tests(); }
1257     void show_results();
1258     //int get_sitecount() const { return m_sitecount; }

1259     // Observer/Mutator
1260     int get_testcount();           // will configure() on demand.

1261     // Mutators
1262     void standardize(string site_name);
1263     void standardize(string site_name, string name_setter);
1264     void normalize(string site_name, string name_setter);
1265     void normalize(string site_name);
1266     void add_site(string path);
1267     void add_site(s_SampledSitePtr site);
1268     void remove_site(string name);

1269     void configure();
1270 private:
```

```

1271     bool stale_conf;

1272     int m_sitecount;
1273     double m_theta;
1274     double m_sine_theta;

1275     // Site Data
1276     s_MasterSitePtr m_master;
1277     vector<s_SampledSitePtr> m_sites;

1278     // Names & Ids
1279     list<string> m_sitenames;
1280     vector<int> m_usable_ids;

1281     // Result Data
1282     list<Results> m_results;
1283     vector<vector<TestResults>> m_tesres_adj;
1284     vector<vector<VerboseResultsPtr>> m_verres_adj;

1285     PlotterPtr m_plotter;

1286     // Helper Methods
1287     void assign_ids();
1288     void set_theta(int points);
1289     void calculate_site(vector<TestResults> & test_results);
1290     void determine_coverage();
1291     bool is_usable(int ID) const;

    // assign ids to sites and tests
    // calculate theta based on how many data points
    // Use all (the most) sites

```

```
1292         int find_site(string name) const;          // search the vector of site names, -1 if not found
1293     };
1294     #endif
1295     <<<<<< END
```

```
1296     <<<<<< plotter.cpp
1297     #include <stdio.h>      /* defines FILENAME_MAX */
1298     #include <string>
1299     #include <cassert>
1300     #include <fstream>
1301     #include <algorithm>
1302     // #include <windows.h>
```

```
1303     #include "plotter.h"
1304     #include "integrator.h"
1305     #include "box-whisker.h"
1306     #include "svg.h"
1307     #include "types.h"
```

```
1308     using std::cerr;
1309     using std::endl;
1310     using std::to_string;
1311     using std::ifstream;
1312     using std::ofstream;
1313     using std::sort;
```

```

1314     Plotter::Plotter(vector<vector<TestResults>> * tesres_adjlist,
1315                      list<Results> * results,
1316                      double theta,
1317                      int endpoints_cnt,
1318                      const MasterSitePtr master)
1319                      : m_ORIGIN_X(300),
1320                      m_ORIGIN_Y(300) {
1321
1322         m_endpoints = endpoints_cnt;
1323
1324         m_theta = theta;
1325         m_radius = 250;
1326
1327         m_adj_list = tesres_adjlist;
1328         m_results = results;
1329         // verbose results here
1330
1331         // sort results
1332         m_results->sort([](const Results & lhs, const Results & rhs) { return lhs.mean() >
1333                                rhs.mean(); });
1334
1335         import_colors();
1336     }
1337
1338     void Plotter::plot_results(string out_filename) {
1339         const char * out_filename_cstr = out_filename.c_str();
1340
1341         vector<vector<TestResults>>::const_iterator sites = (*m_adj_list).begin();

```



```
1335     vector<string>::const_iterator color = m_colors.begin();
1336     string buffer = string();
1337     ofstream out_file;

1338     out_file.open(out_filename_cstr);
1339     if (!out_file.good()) {
1340         cerr << "Could not create " << out_filename << ". " << endl;
1341         return;
1342     }

1343     html_header(buffer);

1344     buffer += "<div class='wrapper'>\n";
1345     // START OF RESULTS TABLE
1346     result_table_hdr(buffer);
1347     result_table_rows(buffer);
1348     result_table_ftr(buffer);
1349     out_file << buffer;
1350     buffer.clear();
1351     // END OF RESULT TABLE

1352     // START OF SVG
1353     SVG::header(buffer, 600, 600);
1354     for (const vector<TestResults> & site : *m_adj_list) {
1355         if (color == m_colors.end())
1356             color = m_colors.begin();
1357         draw_site(buffer, *color, site);
```

```

1358     draw_legend(buffer, *color, site[0].site_name());
1359     color++;
1360 }
1361     draw_axis(buffer);
1362     SVG::footer(buffer);

1363     buffer += "</div>\n";

1364     out_file << buffer;
1365     buffer.clear();
1366     // END OF SVG

1367     // END OF HTML
1368     out_file << buffer;
1369     buffer.clear();
1370     out_file.close();

1371     //ShellExecute(NULL, "open", out_filename_cstr,
1372     //     NULL, NULL, SW_SHOWNORMAL);
1373 }

1374 // Helpers: Output Formatting
1375 void Plotter::html_header(string &_html) {
1376     // Create necessary HTML specs
1377     _html += "<!DOCTYPE html>\n";
1378     _html += "<html>\n";
1379     _html += "<head>\n";

```

```

1380 // styles and fonts
1381 _html += "<link rel=\"stylesheet\" type=\"text/css\" href=\"styles.css\">\n";
1382 _html += "<link href='https://fonts.googleapis.com/css?family=Droid+Sans+Mono'
1383 rel='stylesheet' type='text/css'>\n";
1384 _html += "<link href='https://fonts.googleapis.com/css?family=Open+Sans' rel='stylesheet'
1385 type='text/css'>\n";
1386 _html += "</head>\n";
1387 _html += "<body>\n";
1388 }

```

```

1389 void Plotter::result_table_hdr(string &_html) {
1390     _html += "<div class='table-wrap'>";
1391     _html += "<table id='results-table'>\n";
1392     _html += "<colgroup class='site-col'></colgroup>\n";
1393     _html += "<colgroup class='lower-col'></colgroup>\n";
1394     _html += "<colgroup class='mean-col' id='important'></colgroup>\n";
1395     _html += "<colgroup class='upper-col'></colgroup>\n";
1396     _html += "<thead>\n";
1397     _html += "<tr class='header'>\n";
1398     _html += "<th class='header-label'><span>Region</span></th>\n";
1399     _html += "<th class='header-label'><span>+ Error</span></th>\n";
1400     _html += "<th class='header-label'><span>Mean</span></th>\n";
1401     _html += "<th class='header-label'><span>- Error</span></th>\n";
1402     _html += "</tr>\n";
1403     _html += "</thead>\n";

```

```
1404     }
```

```
1405     void Plotter::result_table_rows(string & _html) {  
1406         list<Results>::const_iterator results = m_results->begin();
```

```
1407         string temp;  
1408         int pos;
```

```
1409         _html += "<tbody>\n";  
1410         while (results != m_results->end()) {  
1411             _html += "<tr>\n";  
1412             _html += "<td class='site-name'>";  
1413             _html += results->site_name();  
1414             _html += "</td>\n";
```

```
1415             _html += "<td class='num-result'>";  
1416             temp = to_string(results->upper_error());  
1417             pos = temp.find('.');  
1418             pos += 3;  
1419             temp = temp.substr(0, pos);  
1420             _html += temp;  
1421             temp.clear();  
1422             _html += "</td>\n";
```

```
1423             _html += "<td class='num-result'>";  
1424             temp = to_string(results->mean());  
1425             pos = temp.find('.');  
1426             temp = temp.substr(0, pos);  
1427             _html += temp;  
1428             temp.clear();  
1429             _html += "</td>\n";
```

```

1426 pos += 3;
1427 temp = temp.substr(0, pos);
1428 _html += temp;
1429 temp.clear();
1430 _html += "</td>\n";

1431 _html += "<td class='num-result'>";
1432 temp = to_string(results->lower_error());
1433 pos = temp.find('.');
1434 pos += 3;
1435 temp = temp.substr(0, pos);
1436 _html += temp;
1437 temp.clear();
1438 _html += "</td>\n";

1439 _html += "</tr>\n";
1440 results++;
1441 }
1442 _html += "</tbody>\n";
1443 }

1444 void Plotter::result_table_ftr(string & _html) {
1445     _html += "</table>\n";
1446     _html += "</div>\n";
1447 }

```

```

1448 void Plotter::html_footer(string & _html) {
1449     _html += "</body>\n";
1450     _html += "</html>\n";
1451 }

1452 // Helpers: Macro SVG generator methods
1453 void Plotter::draw_axis(string & buffer) {
1454     double end_x;
1455     double end_y;
1456     double theta = 0;

1457     vector<TestResults> & tests = m_adj_list[0][0];
1458     vector<TestResults>::const_iterator label_iter = tests.begin();

1459     for (int i = 0; i < m_endpoints; i++) {
1460         ptoc(m_ORIGIN_X, m_ORIGIN_Y, m_radius, theta, end_x, end_y);
1461         SVG::line(buffer, m_ORIGIN_X, m_ORIGIN_Y, end_x, end_y, "stroke='#000000'
1462             stroke-width='1'");
1463         theta += m_theta;
1464     }

1465     theta = 0;
1466     for (int i = 0; i < m_endpoints; i++) {
1467         ptoc(m_ORIGIN_X, m_ORIGIN_Y, m_radius + 15, theta, end_x, end_y);
1468         SVG::text(buffer, label_iter->test_name(), end_x, end_y, "fill='#000'");

```

```
1469         label_iter++;
1470         theta += m_theta;
1471     }
1472 }
```

```
1473 void Plotter::draw_section(string & buffer,
1474                             const string & color,
1475                             int offset,
1476                             const TestResults & curr,
1477                             const TestResults & next) {
```

```
1478     vector<double> xs;
1479     vector<double> ys;
1480     double theta = offset * m_theta;
1481     double end_x;
1482     double end_y;
```

```
1483     double mean_x1, mean_y1;
1484     double mean_x2, mean_y2;
```

```
1485     string extra;
```

```
1486     extra = "fill='";
1487     extra += color;
1488     extra += "' fill-opacity='0.45' stroke='";
1489     extra += color;
1490     extra += "' stroke-width='1'";
```

```

1491     ptoc(m_ORIGIN_X, m_ORIGIN_Y, curr.mean() * 2.5, theta, mean_x1, mean_y1);
1492     ptoc(m_ORIGIN_X, m_ORIGIN_Y, next.mean() * 2.5, theta + m_theta, mean_x2, mean_y2);

1493     // Get first corner
1494     ptoc(m_ORIGIN_X, m_ORIGIN_Y, curr.upper_error() * 2.5, theta, end_x, end_y);
1495     xs.push_back(end_x);
1496     ys.push_back(end_y);

1497     // Get second corner
1498     ptoc(m_ORIGIN_X, m_ORIGIN_Y, next.upper_error() * 2.5, theta + m_theta, end_x, end_y);
1499     xs.push_back(end_x);
1500     ys.push_back(end_y);

1501     // Get third corner
1502     ptoc(m_ORIGIN_X, m_ORIGIN_Y, next.lower_error() * 2.5, theta + m_theta, end_x, end_y);
1503     xs.push_back(end_x);
1504     ys.push_back(end_y);

1505     // Get fourth corner
1506     ptoc(m_ORIGIN_X, m_ORIGIN_Y, curr.lower_error() * 2.5, theta, end_x, end_y);
1507     xs.push_back(end_x);
1508     ys.push_back(end_y);

1509     SVG::path(buffer, xs, ys, extra);
1510     draw_mean(buffer, mean_x1, mean_y1, mean_x2, mean_y2);
1511 }

```



```

1512 // Really F'n optimized...
1513 void Plotter::draw_site(string & buffer,
1514     const string & color,
1515     const vector<TestResults> & tests) {
1516     vector<TestResults>::const_iterator test_iter = tests.begin();
1517     const TestResults & start = *test_iter;
1518     const TestResults * lag = &start;
1519     const TestResults * curr = &tests[0];
1520     int offset;
1521     int test_cnt = tests.size();
1522     for (offset = 1; offset < test_cnt; offset++) {
1523         curr = &tests[offset];
1524         draw_section(buffer, color, offset - 1, *lag, *curr);
1525         lag = curr;
1526     }
1527     draw_section(buffer, color, offset - 1, *curr, start);
1528 }
1529 void Plotter::draw_Legend(string & buffer,
1530     const string & color,
1531     string site_name) {
1532     static int placeholder = m_results->size();

```

```

1533     int start = 600 - (placeholder * 15);

1534     string extra = "stroke='" + color + "' ";
1535     extra += "fill='" + color + "' ";
1536     extra += "fill-opacity='0.45' ";

1537     SVG::rectangle(buffer, 10, start, 36, 12, extra);
1538     SVG::text(buffer, site_name, 55, start + 10);

1539     placeholder -= 1;
1540 }

1541 void Plotter::draw_mean(string & buffer,
1542     double x1,
1543     double y1,
1544     double x2,
1545     double y2) {
1546     SVG::point(buffer, x1, y1, 2, "stroke='#000000'");
1547     SVG::line(buffer, x1, y1, x2, y2, "stroke='#000000' stroke-width='1'");
1548 }

1549 // Helpers: General plotter methods
1550 void Plotter::import_colors() {
1551     ifstream colors_file;
1552     string color = "colors.dat";

```

```

1553     colors_file.open(color.c_str());
1554     colors_file >> color;

1555         while (colors_file) {
1556             m_colors.push_back(color);
1557             colors_file >> color;
1558         }
1559     }

1560 void Plotter::ptoc(double start_x, double start_y, double radius, double angle, double & end_x,
1561 double & end_y) {
1562     end_x = (start_x + (radius * cos(angle)));
1563     end_y = (start_y + (radius * sin(angle)));
1564 }
1565 <<<<<< END

<<<<<< plotter.h
1566 #ifndef PLOTTER_H
1567 #define PLOTTER_H
1568

1569 #ifndef _USE_MATH_DEFINES
1570 #define _USE_MATH_DEFINES
1571 #endif // !_USE_MATH_DEFINES

1572 // System Directions

```

```

1573 #include <string>
1574 #include <vector>
1575 #include "math.h"

1576 // Other Directives
1577 #include "types.h"
1578 #include "sitemaster.h"
1579 #include "integrator.h"
1580 #include "svg.h"

1581 // Typedefs
1582 class Plotter;
1583 typedef Plotter* PlotterPtr;

1584 // Using Statements
1585 using std::vector;
1586 using std::cos;
1587 using std::sin;

1588 class Plotter {
1589 public:
1590     Plotter() : m_ORIGIN_X(0), m_ORIGIN_Y(0) {};
1591     Plotter(vector<vector<TestResults>> * tesres_adjlist,
1592            list<Results> * show_results,
1593            double theta,
1594            int test_cnt,
1595            const MasterSitePtr master);

```

```
1596     ~Plotter() {};  
  
1597     void plot_results(string out_filename="output.html");  
  
1598     private:  
1599         int m_endpoints;  
  
1600         const double m_ORIGIN_X;  
1601         const double m_ORIGIN_Y;  
  
1602         double m_theta;  
1603         double m_radius;  
  
1604         vector<vector<TestResults>> * m_adj_list;  
1605         list<Results> * m_results;  
1606         vector<string> m_colors;  
  
1607         // Helpers: Output Formatting  
1608         void html_header(string &_html);  
1609         void result_table_hdr(string &_html);  
1610         void result_table_rows(string &_html);  
1611         void result_table_ftr(string &_html);  
1612         void html_footer(string &_html);  
  
1613         // Helpers: SVG 'plotter' methods  
1614         void draw_axis(string & output);  
1615         void draw_section(string & output,
```

```

1616     const string & color,
1617     int offset,
1618     const TestResults & curr,
1619     const TestResults & next);
1620     void draw_site(string & out,
1621     const string & color,
1622     const vector<TestResults> & tests);
1623     void draw_legend(string & buffer,
1624     const string & color,
1625     string site_name);
1626     void draw_mean(string & output,
1627     double x1,
1628     double y1,
1629     double x2,
1630     double y2);

1631     // Helpers: General plotter methods
1632     void import_colors();
1633     void ptoc(double centerX,
1634     double centerY,
1635     double radius,
1636     double angle,
1637     double & end_x,
1638     double & end_y);
1639     };
1640     #endif
1641     <<<<<< END

```

```
1642 <<<<< site.cpp
1643 // System Directives
1644 #include <iostream>
1645 #include <fstream>
1646 #include <cassert>
1647 #include <string>
1648 #include <memory>
1649 #include <algorithm>
```

```
1650 // User Defined Directives
1651 #include "types.h"
1652 #include "site.h"
1653 using std::cout;
1654 using std::endl;
1655 using std::swap;
1656 using std::string;
1657 using std::vector;
1658 using std::ifstream;
1659 using std::shared_ptr;
1660 using std::sort;
```

```
1661 int Site::get_n(string test_name) {
1662     DatasetPtr temp = get_test(test_name);
1663     return temp ? temp->size() : -1;
1664 }
```

```
1665 // Inherited Methods
```

```
1666 void Site::add_test(const Dataset& add_data) {
1667     DatasetPtr temp = new Dataset(add_data);
1668     m_tests.push_back(temp);
1669     sort_tests();
1670 }
```

```
1670     delete temp;
1671     temp = nullptr;
1672 }
```

```
1673 void Site::add_test(DatasetPtr add_data) {
1674     DatasetPtr new_set = new Dataset(add_data);
1675     m_tests.push_back(new_set);
1676     sort_tests();
1677 }
```

```
1678 void Site::merge_test(const Dataset &add_data) {
1679     shared_ptr<Dataset> temp(new Dataset(add_data));
1680     vector<DatasetPtr>::iterator iter;
1681     for (iter = m_tests.begin(); iter != m_tests.end(); ++iter)
1682         if ((*iter)->name() == temp->name())
1683             (*iter)->merge(temp);
1684 }
```

```
1685 void Site::merge_test(DatasetPtr add_data) {
1686     vector<DatasetPtr>::iterator iter;
```



```

1687         for (iter = m_tests.begin(); iter != m_tests.end(); ++iter)
1688             if ((*iter)->name() == add_data->name())
1689                 (*iter)->merge(add_data);
1690     }

```

```

1691     DatasetPtr Site::get_test(string test_name) const {
1692         for (DatasetPtr dataset : m_tests)
1693             if (dataset->name() == test_name)
1694                 return dataset;
1695         return nullptr;
1696     }

```

```

1697     bool Site::collision(DatasetPtr dataset) const {
1698         vector<DatasetPtr>::const_iterator iter;
1699         for (iter = m_tests.begin(); iter != m_tests.end(); ++iter)
1700             if ((*iter)->name() == dataset->name())
1701                 return true;
1702         return false;
1703     }

```

```

1704     void Site::sort_tests() {
1705         bool swapped = true;
1706         int start = 0;
1707         while (swapped) {
1708             swapped = false;

```

// uses bubble sort

```

1709         start++;

1710         for (size_t i = 0; i < m_tests.size() - start; i++) {
1711             if (m_tests[i]->name() > m_tests[i + 1]->name()) {
1712                 swap(m_tests[i], m_tests[i + 1]);
1713                 swapped = true;
1714             }
1715         }
1716     }
1717 }
1718 <<<<<< END

1719 <<<<<< site.h
1720 #ifndef SITECORE_H
1721 #define SITECORE_H

1722 // System Directives
1723 #include <iostream>
1724 #include <vector>
1725 #include <list>
1726 #include <memory>

1727 // User Defined Directives
1728 #include "dataset.h"
1729 #include "types.h"

1730 // Using Statements

```

```

1731 using std::vector;
1732 using std::list;
1733 using std::shared_ptr;

1734 // Forward Declarations
1735 class Site;

1736 // Typedef and enumerations
1737 typedef shared_ptr<Dataset> sDatasetPtr;
1738 typedef shared_ptr<Site> s_SiteCorePtr;

1739 class Site {
1740 public:
1741     Site() {};
1742     ~Site() {};

                                     // Constructors & Destructor

1743     virtual void print_tests() = 0;
                                     // Observers
1744     virtual void describe() = 0;
1745     int get_n(string test_name);
1746 protected:
1747     vector<DatasetPtr> m_tests;

                                     // Datasets

1748     // To remove
1749     list<VerboseResultsPtr> m_descriptions;

                                     // Descriptive stats of those sets

1750     // Operations
1751     void add_test(const Dataset &add_data);
                                     // Single, unique DataSet object by reference
1752     void add_test(DatasetPtr add_data);
                                     // Single, unique DataSet object from shared ptr

```

```
1753 void merge_test(const Dataset &add_data); // Merge to an existing DataSet, by reference
1754 void merge_test(DatasetPtr add_data); // Merge to an existing DataSet, by shared ptr
1755 virtual void remove_test(string testname) = 0;

1756 // Search
1757 DatasetPtr get_test(string test_name) const;
1758 bool collision(DatasetPtr test) const;

1759 void sort_tests();
1760 };
1761 #endif
1762 <<<<<< END
```

```
1763 <<<<<< sitemaster.cpp
1764 // System Directives
1765 #include <iostream>
1766 #include <iomanip>
1767 #include <fstream>
1768 #include <cassert>
1769 #include <string>
1770 #include <memory>
```

```
1771 // User Defined Directives
1772 #include "types.h"
1773 #include "sitemaster.h"
```

```
1774 // using statements
```

```
1775 using std::cout;
1776 using std::endl;
1777 using std::setw;
1778 using std::ifstream;
1779 using std::string;
1780 using std::shared_ptr;
1781 using std::vector;
```

```
1782 // Constructors & Destructor
1783 MasterSite::MasterSite(const MasterSite &copy_in) : stale_descript(true) {
1784     vector<DatasetPtr>::const_iterator cpy_iter = copy_in.m_tests.begin();
1785     vector<DatasetPtr>::iterator msite_it;
```

```
1786     while (cpy_iter != copy_in.m_tests.end()) {
1787         (collision((*cpy_iter))) ? merge_test(*cpy_iter) : add_test(*cpy_iter);
1788         cpy_iter++;
1789     }
1790     sort_tests();
1791     id_tests();
1792 }
```

```
1793 MasterSite::MasterSite(const SampledSite& copy_in) : stale_descript(true) {
1794     vector<DatasetPtr>::const_iterator cpy_iter = copy_in.m_tests.begin();
```

```
1795     while (cpy_iter != copy_in.m_tests.end()) {
1796         (collision((*cpy_iter))) ? merge_test(*cpy_iter) : add_test(*cpy_iter);
```

```

1797         cpy_iter++;
1798     }
1799     sort_tests();
1800     id_tests();
1801 }

1802 MasterSite::MasterSite(string path) : stale_descript(true) {
1803     bool desc;
1804     char order;

1805     string name_setter;
1806     string test_unit;
1807     string s_reader;

1808     ifstream InputFile;
1809     vector<double> data;

1810     // Procedure
1811     InputFile.open(path.c_str());
1812     getline(InputFile, s_reader);
1813     assert(InputFile && "Could not be read.");
1814     while (getline(InputFile, name_setter)) {
1815         InputFile >> order;
1816         toupper(order);
1817         InputFile.get();

1818         desc = (order == 'A') ? false : true;

                                // Clear site name
                                // set name
                                // ordering
                                // clear \n

```

```

1819     getline(InputFile, test_unit);           // unit
1820     getline(InputFile, s_reader);           // priming data read

1821     while (s_reader.at(0) < 58) {
1822         data.push_back(stod(s_reader));
1823         getline(InputFile, s_reader);
1824     }                                         // Got all data

1825     getline(InputFile, s_reader);           // Clear "END"

1826     DatasetPtr temp = new Dataset(data, name_setter, test_unit, desc);

1827     if (!collision(temp)) {                 // don't add datasets with same name
1828         m_tests.push_back(temp);
1829     }
1830     data.clear();
1831 }

1832     InputFile.close();
1833     sort_tests();                           // Sort all the Datasets names for this
1834     site
1835     id_tests();
1836 }

1837     MasterSite::MasterSite(vector<string> paths) : stale_descript(true) {
1838         for (string path : paths) {

```

```
1839     shared_ptr<SampledSite> temp(new SampledSite(path));
1840     this->MasterSite::add_data(temp);
1841     temp.reset();
1842 }
1843     sort_tests();
1844     id_tests();
1845 }
1846     MasterSite::~MasterSite() {
1847         for (DatasetPtr test : m_tests) {
1848             delete test;
1849             test = nullptr;
1850         }
1851         for (VerboseResultsPtr result : m_descriptions)
1852             delete result;
1853         m_tests.clear();
1854         m_descriptions.clear();
1855     }
1856     // Observers
1857     void MasterSite::print_tests() {
1858         vector<DatasetPtr>::iterator iter;
```



```

1859     cout << "~~~~~ MASTER ~~~~~" << endl;
1860     for (iter = m_tests.begin(); iter != m_tests.end(); ++iter)
1861         (*iter)->print();

1862     cout << "=== END ===" << " === END ===" << endl;
1863     cout << endl << endl << endl;
1864 }

```

```

1865 void MasterSite::describe() {
1866     VerboseResultsPtr _push;
1867     for (DatasetPtr test : m_tests) {
1868         _push = new TestDescriptor("MASTER",
1869             -1,
1870             test->name(),
1871             test->unit(),
1872             test->id(),
1873             test->size(),
1874             test->up_mean_err(),
1875             test->mean(),
1876             test->lw_mean_err(),
1877             test->minimum(),
1878             test->q1(),
1879             test->median(),
1880             test->q3(),
1881             test->maximum());
1882         m_descriptions.push_back(_push);
1883     }

```

```
1884     }
```

```
1885     void MasterSite::describe(int test_id, VerboseResultsPtr & _result) {
1886         for (VerboseResultsPtr result : m_descriptions)
1887             if (result->test_id() == test_id) {
1888                 _result = result;
1889                 return;
1890             }
1891         _result = nullptr;
1892     }
```

```
1893     int MasterSite::test_size(int test_id) const {
1894         for (DatasetPtr test : m_tests)
1895             if (test->id() == test_id)
1896                 return test->size();
1897         return -1;
1898     }
```

```
1899     void MasterSite::std_refrc(int test_id, double & val, double & basis) {
1900         vector<DatasetPtr>::iterator iter;
1901         for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
1902             if ((*iter)->id() == test_id) {
1903                 val = (*iter)->minimum();
1904                 basis = (*iter)->basis();
1905                 break;

```

```
1906     }  
1907     }  
1908 }
```

```
1909 void MasterSite::std_refrc(string ds_name, double & val, double & basis) {  
1910     vector<DatasetPtr>::iterator iter;  
1911     for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {  
1912         if ((*iter)->name() == ds_name) {  
1913             val = (*iter)->minimum();  
1914             basis = (*iter)->basis();  
1915         }  
1916     }  
1917 }
```

```
1918 DatasetPtr MasterSite::get_test(string name) const {  
1919     vector<DatasetPtr>::const_iterator iter;  
1920     for (iter = m_tests.begin(); iter != m_tests.end(); iter++)  
1921         if ((*iter)->name() == name)  
1922             return *iter;  
1923     return nullptr;  
1924 }
```

```
1925 // Mutators  
1926 void MasterSite::remove_test(string name) {
```

```

1927     sort_tests();
1928     id_tests();
1929     return;
1930     // Not Implement, take the set, remove the values from the Linked List
1931 }

```

```

1932 void MasterSite::add_data(string path) {
1933     SampledSite temp(path);
1934     vector<DatasetPtr>::iterator iter = temp.m_tests.begin();
1935     // check to see if master site has no data, create the first set
1936     if (this->m_tests.size() == 0) {
1937         this->m_tests.push_back(*iter);
1938         iter++;
1939     }
1940     // add (or merge) all (or the rest) of the remaining sets.
1941     while (iter != temp.m_tests.end()) {
1942         (collision((*iter))) ? merge_test(*iter) : add_test(*iter);
1943         iter++;
1944     }
1945     sort_tests();
1946     id_tests();
1947 }

```

```

1948 void MasterSite::add_data(SampledSite &merge) {
1949     vector<DatasetPtr>::iterator iter = merge.m_tests.begin();
1950     // check to see if master site has no data, create the first set

```

```
1951     if (this->m_tests.size() == 0) {
1952         this->m_tests.push_back(*iter);
1953         iter++;
1954     }
1955     // add (or merge) all (or the rest) of the remaining sets.
1956     while (iter != merge.m_tests.end()) {
1957         (collision(*iter) ? merge_test(*iter) : add_test(*iter));
1958         iter++;
1959     }
1960     sort_tests();
1961     id_tests();
1962 }
```

```
1963 void MasterSite::add_data(s_SampledSitePtr merge) {
1964     vector<DatasetPtr>::iterator iter = merge->m_tests.begin();
1965     // check to see if master site has no data, create the first set
1966     if (this->m_tests.size() == 0) {
1967         this->m_tests.push_back(*iter);
1968         iter++;
1969     }
1970     // add (or merge) all (or the rest) of the remaining sets.
1971     while (iter != merge->m_tests.end()) {
1972         (collision(*iter)) ? merge_test(*iter) : add_test(*iter);
1973         iter++;
1974     }
1975     sort_tests();
1976     id_tests();
1977 }
```

```
1977     }
```

```
1978     void MasterSite::id_enumerator(int test_id, string &name_setter) {
1979         vector<DatasetPtr>::const_iterator iter;
1980         for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
1981             if ((*iter)->id() == test_id) {
1982                 name_setter = (*iter)->name();
1983                 break;
1984             }
1985         }
1986     }
```

```
1987     // Private
1988     void MasterSite::id_tests() {
1989         int test_id = 0;
1990         vector<DatasetPtr>::iterator iter;
1991         for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
1992             (*iter)->id_setter(test_id);
1993             test_id++;
1994         }
1995     }
1996     <<<<<< END
```

```
1997     <<<<<<< sitemaster.h
1998     #ifndef MASTERSITE_H
1999     #define MASTERSITE_H
```

```
2000 // System Directives
2001 #include <iostream>
2002 #include <vector>
2003 #include <memory>
2004 // User Defined Directives
2005 #include "dataset.h"
2006 #include "site.h"
2007 #include "sitesample.h"
2008 #include "types.h"
2009 // using statements
2010 using std::vector;
2011 using std::shared_ptr;
2012 // forward declarations
2013 class SampledSite;
2014 class MasterSite;
2015 // typedefs
2016 typedef shared_ptr<Dataset> sDatasetPtr;
2017 typedef shared_ptr<SampledSite> s_SampledSitePtr;
2018 typedef MasterSite* MasterSitePtr;
2019 typedef shared_ptr<MasterSite> s_MasterSitePtr;
2020 class MasterSite : public Site {
2021 public:
```

```

2022 friend class Integrator;
2023 MasterSite() {};
2024 MasterSite(const MasterSite &copy_in);           // Copy a Master Site
2025 MasterSite(const SampledSite &copy_in);          // Copy in a Sampled Site
2026 MasterSite(string path);                        // Construct from a single site data file
2027 MasterSite(vector<string> paths);                // Construct from a vector of paths!
2028 ~MasterSite();

2029 // Observers
2030 void print_tests();
2031 void describe();
2032 void describe(int test_id, VerboseResultsPtr & result);
2033 int test_count() const { return m_tests.size(); }
2034 int test_size(int test_id) const;

2035 void std_refrc(int test_id, double & val, double & basis);
2036 void std_refrc(string name_setter, double & val, double & basis);

2037 // These really need to be reworked
2038 const vector<DatasetPtr>& getMasterSets() const { return m_tests; }
2039 DatasetPtr get_test(string name_setter) const;

2040 // Mutators
2041 void remove_test(string setname);
2042 void add_data(SampledSite &merge);                // Add or merge into this site
2043 void add_data(s_SampledSitePtr merge);            // Add or merge s_SiteCorePtr into this site
2044 void add_data(string path);                        // Add or merge file path.
2045 void id_enumerator(int test_id, string &name_setter);

```



```
2046 private:
2047     bool stale_descript;
2048     void id_tests();
2049 };
2050 #endif
2051 <<<<<< END
```

```
2052 <<<<<< sitesample.cpp
2053 // System Directives
2054 #include <iostream>
2055 #include <iomanip>
2056 #include <fstream>
2057 #include <cassert>
2058 #include <string>
2059 #include <memory>
```

```
2060 // User Defined Directives
2061 #include "types.h"
2062 // #include "site.h"
2063 #include "sitesample.h"
2064 // #include "sitemaster.h"
```

```
2065 // using statements
2066 using std::cout;
2067 using std::endl;
2068 using std::setw;
```

```

2069 using std::ifstream;
2070 using std::string;
2071 using std::shared_ptr;
2072 using std::vector;

2073 // Constructors & Destructor
2074 SampledSite::SampledSite(string path) : stale_descript(true) {
2075     bool desc;
2076     char order;

2077     string test_name;
2078     string test_unit;
2079     string s_reader;

2080     ifstream InputFile;
2081     vector<double> data;

2082     // Procedure
2083     m_stdcount = 0;

2084     InputFile.open(path.c_str());
2085     getline(InputFile, this->site_name);
2086     assert(InputFile && "could not be read.");

2087     while (getline(InputFile, test_name)) {
2088         InputFile >> order;
2089         toupper(order);
2090         InputFile.get();

                                     // Name of test
                                     // Ordering (A/D)
                                     // Clear '\n'

```

```

2091         desc = (order == 'A') ? false : true;

2092         getline(InputFile, test_unit);
2093         getline(InputFile, s_reader);

2094         while (s_reader.at(0) < 58) {
2095             data.push_back(stod(s_reader));
2096             getline(InputFile, s_reader);
2097         }
2098         getline(InputFile, s_reader);

2099         // Create a dataset
2100         DatasetPtr dataset = new Dataset(data,
2101             test_name,
2102             test_unit,
2103             desc);

2104         if (!collision(dataset))
2105             m_tests.push_back(dataset);

2106         // Prepare for next iteration
2107         data.clear();
2108     }

2109     // Clear stream
2110     InputFile.close();
2111     sort_tests();

```

```

// Unit of measure
// Prime the stream

```

```

// Data acquired
// Clear "END"

```

```

// Add uniques tests only

```

```

2112     }

2113     SampleSite::~SampleSite() {
2114         for (DatasetPtr d_ptr : m_tests) {
2115             delete d_ptr;
2116             d_ptr = nullptr;
2117         }

2118         for (VerboseResultsPtr result : m_descriptions)
2119             delete result;

2120         m_tests.clear();
2121         m_descriptions.clear();
2122     }

2123     // Observers
2124     void SampleSite::print_tests() {
2125         cout << "===== " << site_name << " =====" << endl;

2126         for (DatasetPtr test : m_tests)
2127             test->print();

2128         cout << "=== END === " << site_name << " === END ===" << endl;
2129         cout << endl << endl << endl;
2130     }

```

```

2131 void SampledSite::describe() {
2132     VerboseResultsPtr _push;
2133
2134     if (!stale_descript)
2135         return;
2136
2137     for (DatasetPtr test : m_tests) {
2138         _push = new TestDescriptor(name(),
2139             site_id(),
2140             test->name(),
2141             test->unit(),
2142             test->id(),
2143             test->size(),
2144             test->up_mean_err(),
2145             test->mean(),
2146             test->lw_mean_err(),
2147             test->minimum(),
2148             test->q1(),
2149             test->median(),
2150             test->q3(),
2151             test->maximum());
2152         m_descriptions.push_back(_push);
2153     }
2154 }
2155
2156 bool SampledSite::test_present(int test_id) const

```

```
2154 {
2155     vector<DatasetPtr>::const_iterator iter;
2156     for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
2157         if ((*iter)->id() == test_id)
2158             return true;
2159     }
2160     return false;
2161 }
```

```
2162 vector<TestResults> const * SampledSite::get_payloads() {
2163     return (m_payloads.size() == 0) ? nullptr : &m_payloads;
2164 }
```

```
2165 // Mutators
2166 void SampledSite::id_enumerator(int test_id, string name_setter) {
2167     vector<DatasetPtr>::iterator iter;
```

```
2168         for(iter = m_tests.begin(); iter != m_tests.end(); iter++)
2169             if ((*iter)->name() == name_setter) {
2170                 (*iter)->id_setter(test_id);
2171                 break;
2172             }
2173 }
```

```
2174 void SampledSite::remove_test(string name) {
```

```

2175 vector<DatasetPtr>::iterator iter;
2176 for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
2177     if ((*iter)->name() == name) {
2178         if ((*iter)->standardized())
2179             m_stdcount--;
2180         //delete iter;
2181         //iter = nullptr;
2182         m_tests.erase(iter); // does this free memory???!!??!!
2183     }
2184 }
2185

```

```

2186 // you've remove data, you need update master site's set too...
2187 // this is really bad.
2188 }

```

```

2189 void SampledSite::normalize() {
2190     vector<DatasetPtr>::iterator iter;
2191     for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
2192         if ((*iter)->standardized()) {
2193             (*iter)->normalize();
2194             m_stdcount--;
2195         }
2196     }
2197 }

```

```
2198 void SampledSite::normalize(string name_setter) {
2199     vector<DatasetPtr>::iterator iter;
2200     for (iter = m_tests.begin(); iter != m_tests.end(); iter++)
2201         if ((*iter)->name() == name_setter) {
2202             (*iter)->normalize();
2203             m_stdcount--;
2204         }
2205     }
```

```
2206 bool SampledSite::standardize(string dataset_name, double ref_min, double ref_basis) {
2207     vector<DatasetPtr>::iterator iter = m_tests.begin();
```

```
2208     while (iter != m_tests.end()) {
2209         if ((*iter)->name() == dataset_name && !(*iter)->standardized()) {
2210             (*iter)->standardize(ref_min, ref_basis);
2211             m_stdcount++;
2212             return true;
2213         }
2214         iter++;
2215     }
2216     return false;
2217 }
```

```
2218 bool SampledSite::standardize(int test_id, double r_val, double r_basis) {
2219     vector<DatasetPtr>::iterator iter = m_tests.begin();
```



```
2220 while (iter != m_tests.end()) {
2221     if ((*iter)->id() == test_id && !(*iter)->standardized()) {
2222         (*iter)->standardize(r_val, r_basis);
2223         m_stdcount++;
2224         return true;
2225     }
2226     iter++;
2227 }
2228 return false;
2229 }
```

```
2230 void SampledSite::generate_payloads() {
2231     vector<DatasetPtr>::iterator dataset_iter = m_tests.begin();
2232     m_summaries.reserve(m_tests.size());
```

```
2233     for (DatasetPtr dataset : m_tests) {
2234         //m_summaries.push_back(dataset.get());
2235     }
```

```
2236     string site_name = this->name();
2237     int site_id = this->site_id();
2238     string name_setter;
2239     string test_unit;
2240     int test_id;
```

```
2241     double ue, m, le;

2242     PayloadPtr payload;
2243     vector<DatasetPtr>::const_iterator iter;

2244     for (iter = m_tests.begin(); iter != m_tests.end(); iter++) {
2245         if ((*iter)->standardized()) {
2246             // create the payload data
2247             name_setter = (*iter)->name();
2248             test_unit = (*iter)->unit();
2249             test_id = (*iter)->id();
2250             ue = (*iter)->n_upper_bound();
2251             m = (*iter)->norm_mean();
2252             le = (*iter)->n_lower_bound();
2253             // store the payload
2254             payload = new TestResults(site_name,
2255                                     test_id,
2256                                     name_setter,
2257                                     test_unit,
2258                                     test_id,
2259                                     ue,
2260                                     m,
2261                                     le);

2262             m_payloads.push_back(*payload);

2263             delete payload;
```

```
2264         payload = nullptr;
2265     }
2266 }
2267 }
```

```
2268 // Private
2269 DatasetPtr SampledSite::get_test(string name) const {
2270     vector<DatasetPtr>::const_iterator iter;
2271     for (iter = m_tests.begin(); iter != m_tests.end(); iter++)
2272         if ((*iter)->name() == name)
2273             return *iter;
2274     return nullptr;
2275 }
2276 <<<<<< END
```

```
2277 <<<<<< samplesite.h
2278 #ifndef SITESMPL_H
2279 #define SITESMPL_H
```

```
2280 // System Directives
2281 #include <iostream>
2282 #include <memory>
2283 #include <vector>
```

```
2284 // User Defined Directives
2285 #include "dataset.h"
```

```

2286 #include "site.h"
2287 #include "types.h"
2288 //include "sitemaster.h"

2289 // using statements
2290 using std::string;
2291 using std::vector;
2292 using std::shared_ptr;

2293 // Forward Declaration
2294 class MasterSite;
2295 class SampledSite;

2296 // Typedefs
2297 typedef shared_ptr<Dataset> sDatasetPtr;
2298 typedef shared_ptr<MasterSite> s_MasterSitePtr;
2299 typedef SampledSite* SampledSitePtr;           // Redefine types for raw pointers
2300 typedef shared_ptr<SampledSite> s_SampledSitePtr;

2301 class SampledSite : public Site {
2302 public:
2303     friend class Integrator;                // Integrator needs to use id_sets();
2304     friend class MasterSite;                // Master needs access to m_sets for merge

2305     SampledSite() : m_stdcount(0) {};          // Constructors & Destructor
2306     SampledSite(string path);
2307     ~SampledSite();

```

```

2308 // Observers
2309 void print_tests();
2310 void describe();

2311 // to remove
2312 int site_id() const { return m_id; }
2313 string name() const { return site_name; }

2314 int test_count() const { return m_tests.size(); }
2315 int standardize_cnt() const { return m_stdcount; }
2316 bool test_present(int test_id) const;

2317 vector<TestResults> const * get_payloads();

2318 // Mutators
2319 void set_name(string site_name) { site_name = site_name; }
2320 void id_setter(int test_id) { m_id = test_id; }
2321 void id_enumerator(int test_id, string name_setter);

2322 void remove_test(string name_setter);

2323 void normalize(); // Normalize All Datasets
2324 void normalize(string name_setter); // Normalize a Dataset
2325 bool standardize(string name_setter, double ref_min, double ref_basis);
2326 // Returns true is dataset was Normalized
2327 bool standardize(int test_id, double r_min, double r_basis);
2328 void generate_payloads();

```

```
2329     private:
2330         bool stale_descript;
2331         int m_id;
2332         int m_stdcount;
2333
2334         // This site's ID
2335         string site_name;
2336         vector<TestResults> m_payloads;
2337         // This site's name
2338         vector<DatasetSummary*> m_summaries;
2339
2340         DatasetPtr get_test(string name_setter) const;
2341         // Returns a shared pointer to a dataset, nullptr if not found
2342     };
2343     #endif
2344     <<<<<< END
2345
2346     <<<<<< styles.css
2347     body {
2348         font-family: 'Open Sans', sans-serif;
2349     }
2350
2351     svg text{
2352         font-family: 'Open Sans', sans-serif;
2353         font-size: 10px;
2354     }
2355     /* Wrappers */
```

```
2350 .wrapper {
2351   width: 600px;
2352   border: 1px solid #000;
2353 }

2354 .table-wrap {
2355   width: 600px;
2356   border: 1px solid #000;
2357 }

2358 svg {
2359   font-family: labels;
2360 }

2361 /* Results Table */
2362 #results-table {
2363   width: 100%;
2364   margin: 0 auto;
2365   border-collapse: collapse;
2366 }

2367 .header {
2368   height: 12px;
2369 }

2370 .header-label {
2371   background-color: #4C9ED9;
2372   font-size: 12px;
```

```
2373     color: #FFF;
2374 }
2375
2376 .site-col {
2377     width: 300px;
2378 }
2379
2380 .site-name {
2381     font-size: 12px;
2382     font-weight: 100;
2383     padding: 2px;
2384     padding-left: 5px;
2385 }
2386
2387 .num-result {
2388     border-left: 1px solid #555;
2389     font-family: 'Droid Sans Mono';
2390     font-size: 12px;
2391     text-align: right;
2392     padding: 2px;
2393     padding-right: 5px;
2394 }
2395
2396 tr:nth-child(even) {
2397     background-color: #d9ebf7;
2398 }
2399
2400 <<<<<< END
```



```

2396 <<<<< svg.cpp
2397 // System Directives
2398 #include <string>
2399 #include <vector>

2400 // Other Directives
2401 #include "svg.h"

2402 // Using Statements
2403 using std::vector;
2404 using std::to_string;

2405 void SVG::footer(string & svg) {
2406     svg += "</svg>\n";
2407 }

2408 void SVG::header(string & svg,
2409     int width,
2410     int height) {
2411     svg += "<svg version='1.1' baseProfile='full' width='" + to_string(width) + "' ";
2412     svg += "height='" + to_string(height) + "' xmlns='http://www.w3.org/2000/svg'>\n";
2413 }

2414 void SVG::Line(string & buffer,

```

```
2415 double str_x,
2416 double srt_y,
2417 double end_x,
2418 double end_y,
2419 string extra) {
2420     buffer += "<line ";
2421     buffer += "x1='" + to_string(str_x) + "' y1='" + to_string(srt_y) + "' ";
2422     buffer += "x2='" + to_string(end_x) + "' y2='" + to_string(end_y) + "' ";
2423     buffer += extra;
2424     buffer += " />\n";
2425 }
```

```
2426 void SVG::path(string & buffer,
2427               const vector<double> & xs,
2428               const vector<double> & ys,
2429               string extra) {
```

```
2430     vector<double>::const_iterator x_it = xs.begin();
2431     vector<double>::const_iterator y_it = ys.begin();
2432     // Create path tag and attribute d...
2433     buffer += "<path d='";
2434     buffer += 'M' + to_string(*x_it) + ' ' + to_string(*y_it);
```

```
2435     while (x_it != xs.end() && y_it != ys.end()) {
2436         buffer += " L" + to_string(*x_it) + ' ' + to_string(*y_it);
2437         x_it++;
2438         y_it++;
```

```
2439     }
2440     buffer += " Z' ";
2441     buffer += extra;
2442     buffer += " />\n";
2443 }
```

```
2444 void SVG::point(string & buffer,
2445                double x,
2446                double y,
2447                double r,
2448                string extra) {
2449     buffer += "<circle cx='";
2450     buffer += to_string(x);
2451     buffer += "' cy='";
2452     buffer += to_string(y);
2453     buffer += "' ";
2454     buffer += "r='";
2455     buffer += to_string(r);
2456     buffer += "' ";
2457     buffer += extra;
2458     buffer += " />\n";
2459 }
```

```
2460 void SVG::rectangle(string & buffer,
2461                    double x,
2462                    double y,
```

```
2463 double w,  
2464 double h,  
2465 string extra){
```

```
2466     buffer += "<rect x='" + to_string(x) + "' y='" + to_string(y) + "' ";  
2467     buffer += "width='" + to_string(w) + "' height='" + to_string(h) + "' ";  
2468     buffer += extra;  
2469     buffer += ">\n";  
2470 }
```

```
2471 void SVG::text(string & buffer,  
2472 string label,  
2473 double x,  
2474 double y,  
2475 string extra) {
```

```
2476     buffer += "<text x='" + to_string(x) + "'";  
2477     buffer += to_string(x);  
2478     buffer += "' y='" + to_string(y) + "'";  
2479     buffer += to_string(y);  
2480     buffer += "' ";  
2481     buffer += extra;  
2482     buffer += '>';  
2483     buffer += label;  
2484     buffer += "</text>\n";  
2485 }  
2486 <<<<<<< END
```

```
2487 <<<<<< svg.h
2488 #ifndef SVG_H
2489 #define SVG_H
2490 #include <string>
2491 #include <vector>
2492 using std::string;
2493 using std::vector;
2494 namespace SVG {
2495     void footer(string & svg);
2496     void header(string & svg,
2497                 int width,
2498                 int height);
2499     void line(string & ins_line,
2500              double str_x,
2501              double str_y,
2502              double end_x,
2503              double end_y,
2504              string extra = "");
2505     void path(string & ins_path,
2506              const vector<double> & x,
2507              const vector<double> & y,
2508              string extra = "");
2509     void point(string & ins_point,
```

```
2510     double pos_x,
2511           double pos_y,
2512           double r,
2513           string extra = "");
2514     void rectangle(string & buffer,
2515                  double x,
2516                  double y,
2517                  double w,
2518                  double h,
2519                  string extra = "");
2520     void text(string & ins_text,
2521              string label,
2522              double x,
2523              double y,
2524              string extra = "");
2525 };
```

```
2526 #endif
2527 <<<<<< END
```

```
2528 <<<<<< types.cpp
2529 // System directives
```

```
2530 // User Defined Directives
2531 #include "types.h"
```

```
2532 Results::Results(string site_name,
2533                  int site_id,
2534                  double u,
2535                  double m,
2536                  double l) {
2537     m_site_name = site_name;
2538     m_site_id = site_id;
2539     m_up_err = u;
2540     m_mean = m;
2541     m_lw_err = l;
2542 }
```

```
2543 TestResults::TestResults(string site_name,
2544                          int site_id,
2545                          string name_setter, // test
2546                          string test_unit,
2547                          int test_id,
2548                          double u,
2549                          double m,
2550                          double l) :
2551     Results(site_name, site_id, u, m, l) {
2552     m_test_name = name_setter;
2553     m_unit = test_unit;
```

```
2554         m_test_id = test_id;
2555     }
```

```
2556     TestDescriptor::TestDescriptor(const TestResults & t_results, double min, double q1, double med,
2557         double q3, double max)
2558         : TestResults(t_results) {
```

```
2559         m_min = min;
2560         m_q1 = q1;
2561         m_median = med;
2562         m_q3 = q3;
2563         m_max = max;
2564     }
```

```
2565     TestDescriptor::TestDescriptor(string site_name,
2566         int site_id,
2567         string test_name,
2568         string test_unit,
2569         int test_id,
2570         int n,
2571         double ue,
2572         double mean,
2573         double le,
2574         double mini,
2575         double q1,
2576         double med,
```



```

2577     double q3,
2578     double maxi)
2579     : TestResults(site_name, test_id, test_name, test_unit, test_id, ue, mean, le) {

2580         m_n = n;
2581         m_min = mini;
2582         m_q1 = q1;
2583         m_median = med;
2584         m_q3 = q3;
2585         m_max = maxi;
2586     }
2587     <<<<<< END

2588     <<<<<< types.h
2589     #ifndef TYPES_H
2590     #define TYPES_H

2591     #include <iostream>
2592     using std::string;

2593     // Forward Declarations
2594     struct DataNode;
2595     class Results;
2596     class TestResults;
2597     class TestDescriptor;
2598     class Dataset;

```

```

2599     class DatasetSummary;

2600     typedef DataNode* DataNodePtr;           // Redefined type for Pointers.
2601     typedef TestDescriptor* VerboseResultsPtr; // Remove
2602     typedef TestResults* PayloadPtr;          // Remove
2603     typedef Results* ResultsPtr;
2604     typedef DatasetSummary* SummaryPtr;

2605     enum NORM_T { SELF, EXTERNAL };

2606     struct DataNode {
2607         int rank;

2608         double norm_value;
2609         double value;
2610     };

2611     class Results {
2612     public:
2613         Results() {}
2614         Results(string site_name,
2615                 int site_id,
2616                 double u,
2617                 double m,
2618                 double l);

```

```
2619 string site_name() const { return m_site_name; }
2620 int site_id() const { return m_site_id; }
```

```
2621 double upper_error() const { return m_up_err; }
2622 double mean() const { return m_mean; }
2623 double lower_error() const { return m_lw_err; }
```

```
2624 protected:
2625     string m_site_name;
2626     int m_site_id;
```

```
2627 double m_up_err;        // Final data (Areas)
2628 double m_mean;
2629 double m_lw_err;
2630 };
```

```
2631 class TestResults : public Results {
2632     // Once constructed, unmodifiable.
2633     public:
```

```
2634         TestResults() {}
```

```
2635         TestResults(string site_name,
```

```
2636                     int site_id,
```

```
2637                     string test_name,
```

```
2638                     string test_unit,
```

```
2639                     int test_id,
```

```
2640                     double u,
```

```
2641         double m,
2642         double l);
2643
2643         // Observers
2644         string test_name() const { return m_test_name; }
2645         string test_unit() const { return m_unit; }
2646         int test_id() const { return m_test_id; }
2647
2647         // Inherited members serve as single data points (Distances).
2648
2648         protected:
2649         string m_test_name;           // Descriptor
2650         string m_unit;
2651         int m_test_id;
2652         int m_site_id;
2653     };
2654
2654     // Depreciate this
2655     class TestDescriptor : public TestResults {
2656     public:
2657         TestDescriptor(const TestResults & t_results,
2658         double min,
2659         double q1,
2660         double med,
2661         double q3,
2662         double max);
```

```
2663 TestDescriptor(string site_name,
2664               int site_id,
2665               string test_name,
2666               string test_unit,
2667               int test_id,
2668               int n,
2669               double ue,
2670               double mean,
2671               double le,
2672               double mini,
2673               double q1,
2674               double med,
2675               double q3,
2676               double maxi
2677            );
```

```
2678 double mini() const { return m_min; }
2679 double q1() const { return m_q1; }
2680 double median() const { return m_median; }
2681 double q3() const { return m_q3; }
2682 double maxi() const { return m_max; }
```

```
2683 private:
2684     int m_n;
2685     double m_min;
2686     double m_q1;
2687     double m_median;
2688     double m_q3;
```

```

2689         double m_max;
2690     };
2691 #endif
2692 <<<<<< END

2693 <<<<<< index.php
2694 <?php
2695     error_reporting(E_ALL);
2696     ini_set('display_errors', 1);
2697     require 'include/header.php';

2698     $fileIter = new FilesystemIterator('uploads/', FilesystemIterator::SKIP_DOTS);
2699     $fileCount = iterator_count($fileIter);

2700     $target_dir = "uploads/";

2701     $target_file = $target_dir . $fileCount . '.dat';

2702     if (isset($_POST['upload']) && !file_exists($target_file)) {
2703         if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file))
2704             header('location: ' . $_SERVER['PHP_SELF']);
2705     }
2706     ?>
2707     <div class="palette palette-pomegranate">
2708         ERROR: NO FILE SELECTED
2709     </div>

```

```

2710 <?php
2711 }
2712 } else if (isset($_POST['compute'])) {
2713     exec('ETP/a.out');
2714 } else if (isset($_POST['reset'])) {
2715     exec('rm -rf uploads/*');
2716     header('location: ' . $_SERVER['PHP_SELF']);
2717 }
2718 ?>
2719 <body>
2720 <div class="container">
2721 <div class="row">
2722 <h1>Ecotox Project Demo</h1>
2723 <small>Written By Michael Schmidt in C++, PHP</small>
2724 </div>
2725 <div class="row">
2726 <h6>Input your data files</h6>
2727 <small>
2728 <?php
2729     printf("%d File(s) loaded.", $fileCount);
2730     ?>
2731 </small>
2732 </div>
2733 <div class="row">
2734 <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="post"
2735 enctype="multipart/form-data">

```

```

2736         <div class="input-group">
2737             <span class="input-group-btn">
2738                 <input type="file" name="fileToUpload" id="fileToUpload"
2739                     class="btn btn-primary" />
2740             </span>
2741             <input type="submit" name="upload" value="upload" class="btn btn-info"
2742                 />
2743         <?php
2744             if ($fileCount > 2)
2745                 echo '<input type="submit" name="compute" value="compute"
2746                     class="btn btn-primary" />';
2747
2748             if ($fileCount > 0)
2749                 echo '<input type="submit" name="reset" value="reset"
2750                     class="btn btn-warning" />';
2751             ?>
2752         </div>
2753     </div>
2754 </div>
2755 </body>
2756 <?php
2757     require 'include/footer.php';
2758     ?>
2759 <<<<<< END

```



## Bibliography & References

- “8 Paths.” *Paths – SVG 1.1 (Second Edition)*, [www.w3.org/TR/SVG/paths.html](http://www.w3.org/TR/SVG/paths.html).
- “9 Basic Shapes.” *Basic Shapes – SVG 1.1 (Second Edition)*,  
[www.w3.org/TR/SVG/shapes.html](http://www.w3.org/TR/SVG/shapes.html).
- “Area of a triangle - "side angle side" (SAS) method.” *Area of a triangle (Side-Angle-Side method) - Math Open Reference*,  
[www.mathopenref.com/triangleareasas.html](http://www.mathopenref.com/triangleareasas.html).
- “Best Practices for File Formats.” *Stanford Libraries*,  
[library.stanford.edu/research/data-management-services/data-best-practices/best-practices-file-formats](http://library.stanford.edu/research/data-management-services/data-best-practices/best-practices-file-formats).
- Browse : Python Package Index*, [pypi.python.org/pypi?%3Aaction=browse](http://pypi.python.org/pypi?%3Aaction=browse). Retrived  
Mar. 2018
- Choi, Jung Kyoon, and Sang Cheol Kim. *Genetics*, Copyright © 2007 by the Genetics  
Society of America, Apr. 2007,  
[www.ncbi.nlm.nih.gov/pmc/articles/PMC1855137/](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1855137/).
- Collins, A R. “The comet assay for DNA damage and repair: principles, applications, and  
limitations.” *Molecular biotechnology*, U.S. National Library of Medicine, Mar.  
2004, [www.ncbi.nlm.nih.gov/pubmed/15004294](http://www.ncbi.nlm.nih.gov/pubmed/15004294).
- “Flask home.” *Welcome | Flask (A Python Microframework)*, [flask.pocoo.org/](http://flask.pocoo.org/).
- “How to Choose a Data Format - Silicon Valley Data Science.” *Silicon Valley Data  
Science*, 21 Sept. 2017, [svds.com/how-to-choose-a-data-format/](http://svds.com/how-to-choose-a-data-format/).

“Is it a good practice to always scale/Normalize data for machine learning?” *Cross Validated*,  
[stats.stackexchange.com/questions/189652/is-it-a-good-practice-to-always-scale-normalize-data-for-machine-learning](https://stats.stackexchange.com/questions/189652/is-it-a-good-practice-to-always-scale-normalize-data-for-machine-learning).

MacKay, Jory, and Jory MacKay Editor at Crew. Writer. Reader. Coffee drinker.

“Building Facebook today: Is PHP still relevant in 2017?” *Crew.co*, 29 Jan. 2017,  
[crew.co/blog/is-php-still-relevant-in-2017/](http://crew.co/blog/is-php-still-relevant-in-2017/).

Mark Otto, Jacob Thornton, and Bootstrap contributors. “Bootstrap.” · *The most popular HTML, CSS, and JS library in the world.*, [getbootstrap.com/](http://getbootstrap.com/).

“NumPy.” *NumPy - NumPy*, [numpy.org/](http://numpy.org/).

“SciPy.org.” *SciPy.org*, Enthought, 2018, [scipy.org/](http://scipy.org/).

Smith, Keith. *Environmental hazards: assessing risk and reducing disaster*. Routledge, 2013.

“The Computer Language Benchmarks Game.” *The Computer Language Benchmarks Game*, [benchmarksgame.alioth.debian.org/](http://benchmarksgame.alioth.debian.org/).

Williams, Sarah. *Normalizing Census Data : Census : 11.520*, 1 Oct. 2002,  
[web.mit.edu/11.520/www/labs/lab5/normalize.html](http://web.mit.edu/11.520/www/labs/lab5/normalize.html).

Zamaratskaia, Galia, and Vladimir Zlabek. *Sensors (Basel, Switzerland)*, Molecular Diversity Preservation International (MDPI), 2009,  
[www.ncbi.nlm.nih.gov/pmc/articles/PMC3345832/](http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3345832/).