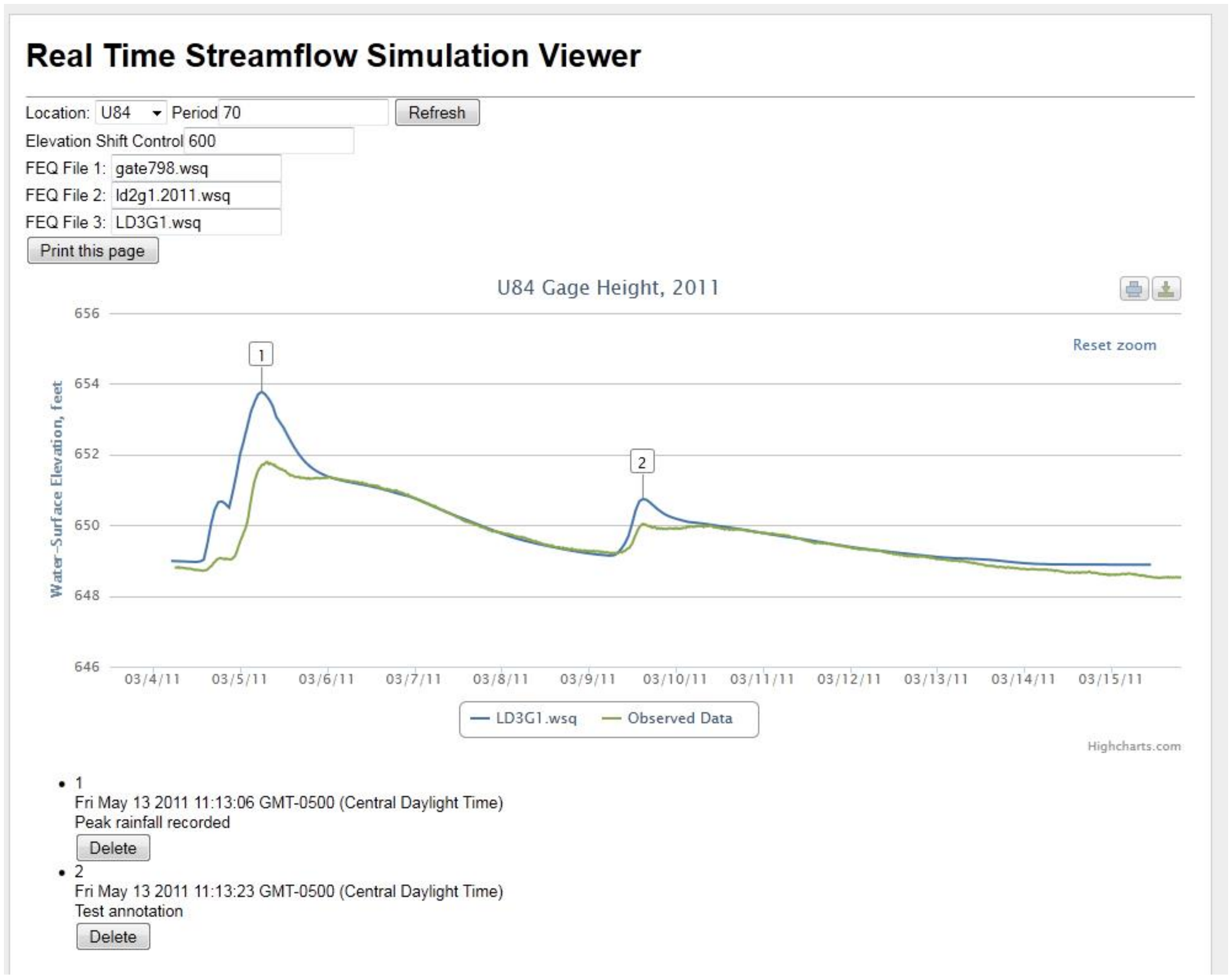
Final Design

Following is both documentation on how to use the application and what is going on behind-the-scenes with each file in the application package. We have also included additional support information for using the program.

Our design has changed significantly since the semester began. We overhauled the PHP and JavaScript so that each was isolated in their own files. Previously, the JavaScript was embedded in the PHP code which made it difficult to read and change. The new design lets us modify the PHP and JavaScript without having to modify the other. The JavaScript communicates with PHP via JSON requests.

Our new design also makes heavy use of jQuery which is a quick and efficient method of traversing and modifying elements of the DOM. The use of jQuery has sped up development time on the JavaScript side of the project.


Real Time Streamflow Simulation Viewer

Location: U84 Period 70 Refresh

Elevation Shift Control 600

FEQ File 1: gate798.wsq

FEQ File 2: ld2g1.2011.wsq

FEQ File 3: LD3G1.wsq

- 1
  Fri May 13 2011 11:13:06 GMT-0500 (Central Daylight Time)
  Peak rainfall recorded
  Delete
- 2
  Fri May 13 2011 11:13:23 GMT-0500 (Central Daylight Time)
  Test annotation
  Delete

A screenshot of the final application showing annotations

Detailed Feature List

1. Data Gathering

a. Real time data - The goal of this project was to automatically display the latest streamflow data from the USGS servers. We accomplished this goal and currently support downloading and parsing data from USGS within the past 120 days. The 120 day restriction is imposed by the service provided by USGS to view observed data.

b. Simulated data - The other form of data that we allow the user to display is the simulated data. Simulated data is contained in FEQ flat files which must be parsed before being displayed. Our system automatically parsed FEQ files found in the simulationFiles directory. To use this feature the user needs to upload FEQ files onto the server into the simulationFiles folder. The files need to be in a specific format otherwise there will be an error parsing the data and the correct results will not be returned.

c. Types - The USGS servers contain information for many types of data. We currently support gage height data, discharge data and precipitation.

d. Shifting Values - The simulation files and observed data are measured in different units so to display them properly on the same graph we support data shifting. This shift value has been provided to us by our client. For instance, the observed data for location U22 is off by 663.29 units, so to correctly display it our system will automatically shift the observed data by 663.29.

2. Charts

a. Chart data - Our main feature is displaying series of data in an efficient manner. We currently support charting the gage height, discharge and precipitation of five locations. The number of locations can easily be increased, it is currently restricted to 5 since those were the only locations for which our client provided information to query real time observed data.

b. Multiple series - Our final design includes the ability to chart up to 10 simulation files at once on top of the observed data. The user will also be able to remove plot lines from the current view by clicking on the the Legend Item in case they want to see the plot with specific plot lines or want to compare different plot lines in order to see how well their prediction and modelling system is functioning.

c. Zooming - Highcharts supports zooming along the x-axis. We modified this functionality to automatically update annotation locations as well as change the x-axis labels. The labels on the x-axis become more time specific the more the user zooms in. We currently support zooming in up to a nine hour period. Zooming past this limit makes the graph sparse since we do not have enough data to make smooth plot lines after the nine hour mark.

d. Location Names - The chart title shows the location name instead of the location number, the mapping from location number to location name is provided by the client. This makes the chart more readable for future use.

e. Printing - Highcharts does not support printing so we added functionality to print the currently displayed chart. This is done using print css and javascript to only send the chart and necessary data to the printer.

f. Speed - By default, highcharts enables a lot of visual goodies and animations. Since we are dealing with tens of thousands of points of data, we had to modify our charts to focus on speed. We disabled all of the visual flare and animations.

g. Precipitation - We made a special chart for precipitation values. The precipitation values rarely go over 0.1 inches so we created a small chart that lies below the gage height chart.

3. Annotation

a. Add / Delete - Users can add annotations to the chart by clicking on a series at the point they would like to annotate. A popup allows them to type in the annotation description. The annotation descriptions are displayed beneath the graphs. Each description has a corresponding delete button that allows the user to delete unwanted annotations.

b. Persistent - Annotations are stored in a database and automatically retrieved when a chart is displayed. This allows users to leave annotations for future use.

c. Displaying - Annotations pointers are drawn to the chart. The pointers are automatically updated when the chart is changed or redrawn, for example, when the user zooms in. This makes annotations easy to read and update.

User Documentation

System Installation:

Setting up annotation table: The installation folder should be extracted to the web server. After extraction the user should open a web browser(the application is supported in IE8, IE 9, Firefox 3.8,3.9,4 and Chrome) and navigate to /installation_folder_name/setup.php. This will start the setup script where the user will provide the following information which is required by the application to store annotation data:

- Server location
- Port number
- MySQL username
- MySQL password
- Database name

After the user has entered this information, the script will automatically generate annotation_settings.php file which contains the all the information about the database. The script also creates the MySQL table which will store all the annotation data.

Adding simulation data flat files:

Place any new simulation flat files in the folder “simulationfiles”. This folder already contains some files that have old data.

Accessing the application:

After finishing the setup process, open a web browser and navigate to

/installation_folder_name/

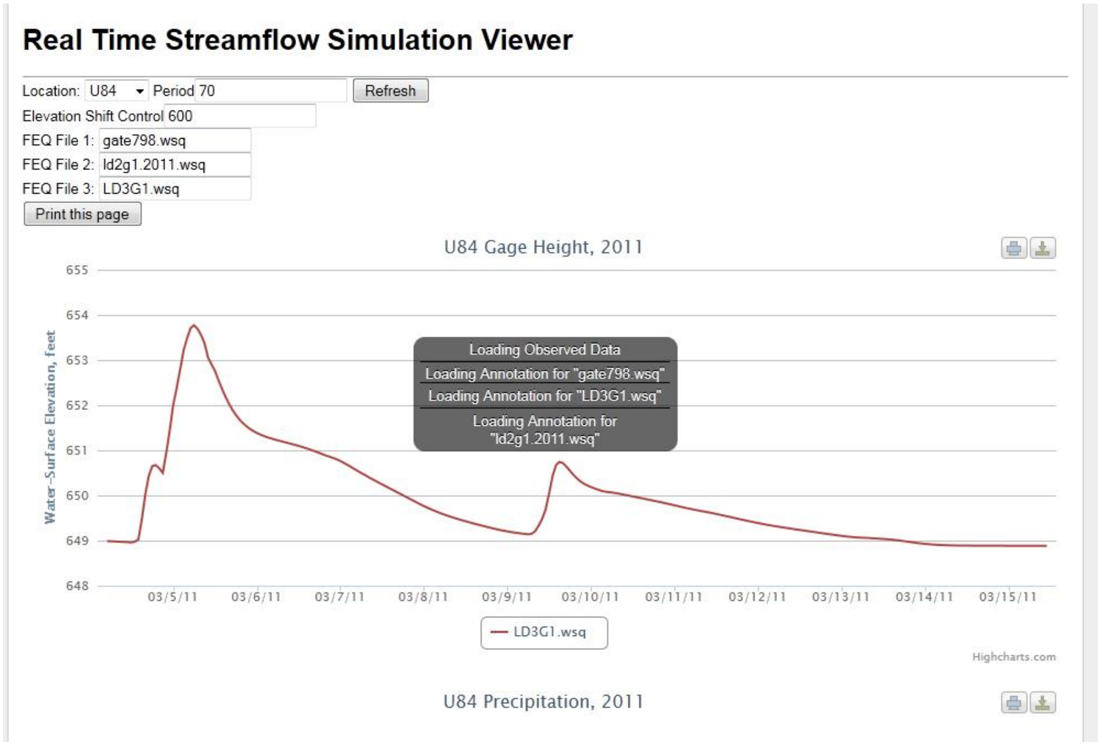
This will load the web application, which loads data for the location ‘U22’ with a period of 7 days as a default. To change location, select another one from the drop down box. To increase or decrease period type a new number into the “Period:” textbox and click on refresh.

Introductory Manual for End Users

The following the core functionality that will be utilized by an end user of our project:

Generating a default chart

The user can start the program by opening a supported web browser and navigating to index.html. This will load the default graph. Floating text will cover the chart indicating what part of the graph is being loaded. The user should not interact with the page until all loading indicators are gone.



Real Time Streamflow Simulation Viewer

Location: U84 Period 70 Refresh

Elevation Shift Control 600

FEQ File 1: gate798.wsq

FEQ File 2: ld2g1.2011.wsq

FEQ File 3: LD3G1.wsq

Print this page

Loading Observed Data

Loading Annotation for "gate798.wsq"

Loading Annotation for "LD3G1.wsq"

Loading Annotation for "ld2g1.2011.wsq"

Generating a chart for a specific location and period

Navigate to index.html and wait for all loading indicators to disappear. Once the page is done loading the user is allowed to interact with the input form near the top of the page. To change the location that the chart is displaying, click on the drop down box that is labeled with “location”. Select the desired location and click the refresh button that is located next to the form (not the browser refresh button).

To change the amount of days that the chart will display, change the value of the input labeled with “period”. The default period is 7 days which means the program will display data for the past 7 days up to the current time. The observed data goes back to a maximum of 120 days. This is not a restriction of the application but of the USGS service which serves the observed data.

Displaying simulated data

The program supports displaying simulated data contained in FEQ files. To display simulated data, place the FEQ files into the folder called simulationFiles. This folder is server side so the user will need to have access to the server or have access to FTP. Once the files are placed within the simulationFiles folder, navigate to index.html. Wait for the page to be done loading. The FEQ files that the program is reading from will be listed near the top of the page.

If the files that were placed in the simulationFiles folder are not listed, double check the folder to make sure those files are in place. If the files are listed, proceed to select the location and period. Click the refresh button and the program will create the charts which will contain the observed data and any simulation data that it can find.

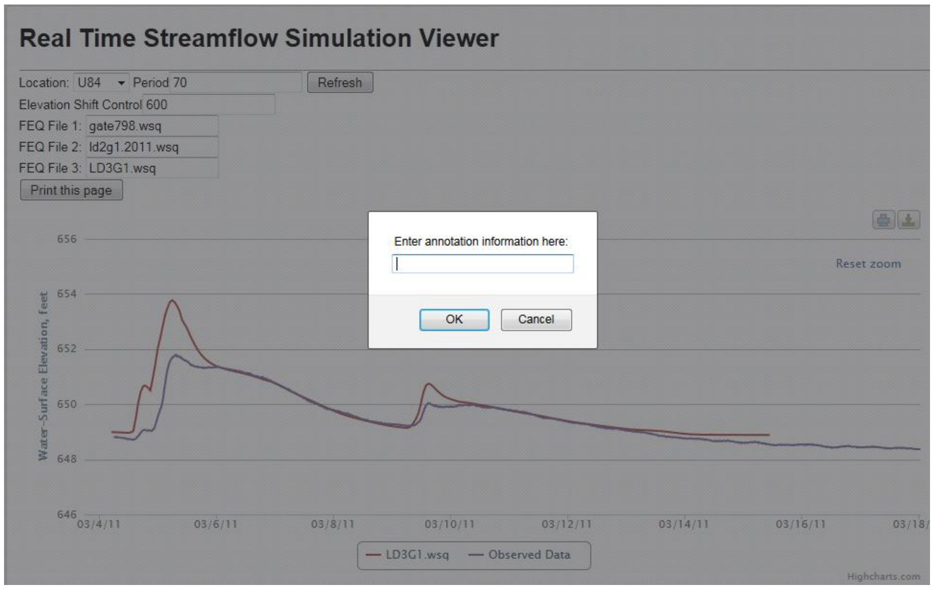
Interacting with the chart

The charts that are loaded are interactive. To see the details of a specific point on the graph, hover the mouse over the point to see a tool-tip that contains the series name and x and y values. To zoom in on a specific time period, click on the graph and drag the mouse to the right to highlight the area you would like to see. This will cause the graph to zoom in. The user can zoom in to a minimum of 9 hours. To zoom back out, click on the “reset zoom” button in the top right corner of the graph.

Annotations

To add an annotation, click on the point that you would like to annotate. At the prompt, enter a description for the annotation. The annotation will automatically be added to the chart and be stored for future use. The annotation will remain on the graph even in future viewings until it is deleted.

Each annotation on the chart is labeled with a number which corresponds to an entry beneath the graph. These entries contain the title and description for the annotation. After each description there is a delete button that allows the user to permanently delete the annotation. Deleting an annotation cannot be undone.


Real Time Streamflow Simulation Viewer

Location: U84 Period 70

Elevation Shift Control 600

FEQ File 1: gate798.wsq

FEQ File 2: ld2g1.2011.wsq

FEQ File 3: LD3G1.wsq

Enter annotation information here:

Saving the chart as an image

The user can save the chart as an image by clicking on the save image icon located in the top right corner of the chart. There are several options for saving an image which include jpeg and png. Note that the image that is saved is of the graph in its default zoom without annotations.

Printing the chart

To print all charts on the screen along with the annotation information for each chart, click on the “print” button found above the charts. To print a single chart, click the print icon found in the top right corner of the chart that you wish to print. To print the entire page including the header and footer, use the browsers file->print functionality.

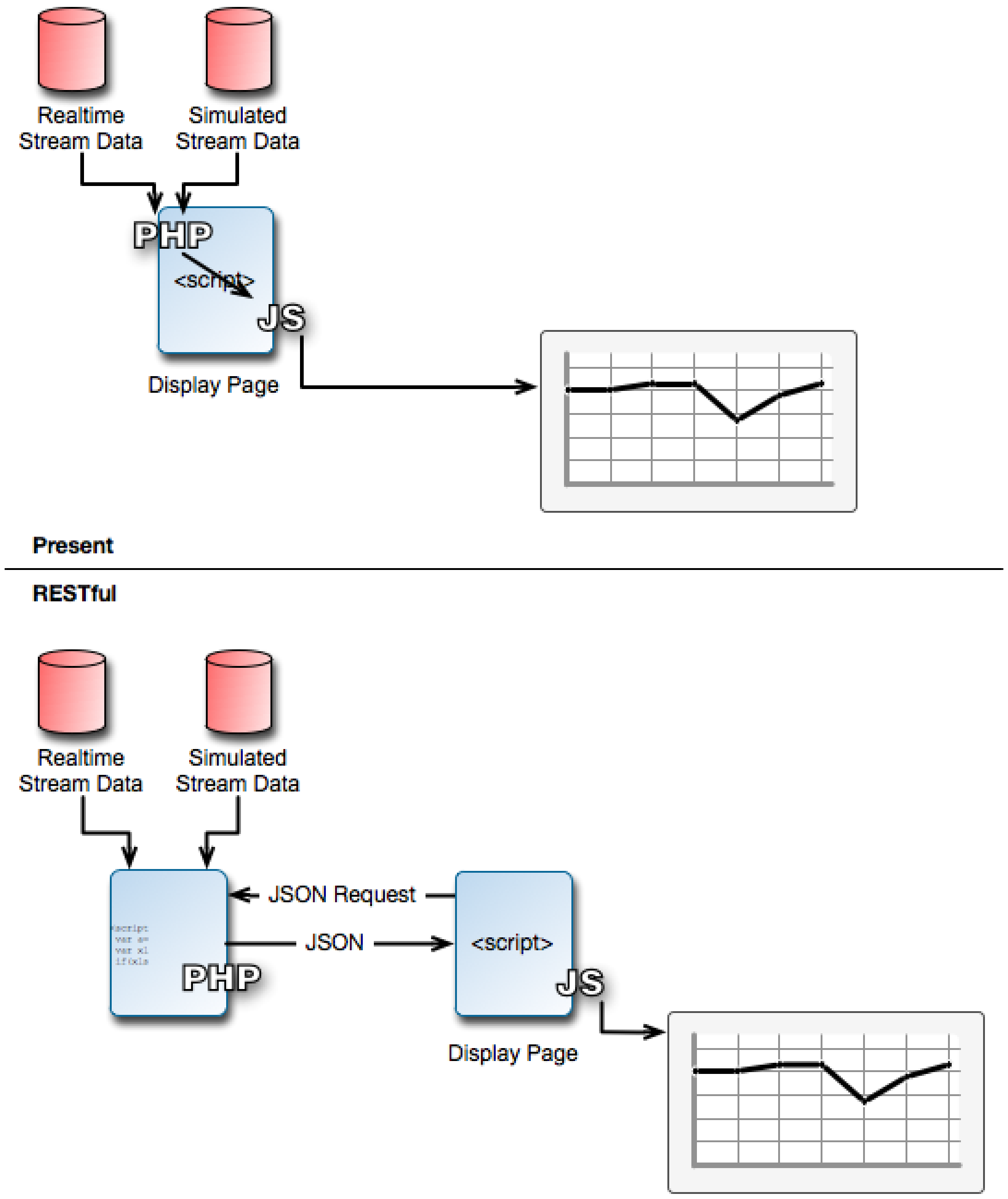
System Documentation

Work-flow

The user now starts the process of chart creation by navigating to index.html. Index.html includes the file rsfd.js which initializes empty charts. There are a few input fields in index.html that contain the parameters for what data should be displayed. The javascripts takes those parameters and requests data from the php files in the form of JSON requests. The php files download data from the USGS servers and read the simulation files that are in the folder simulationFiles. They then parse this data into arrays which makes the data much more accessible. The arrays are then encoded into JSON objects which are returned to the javascript. The javascript loads the data into the charts which are then ready to be viewed by the user.

Annotations are also requested during the time that the php is downloading the data. The annotations are retrieved from a MySql database. Only annotations that belong on the graph are retrieved. Once the annotations are retrieved they are displayed underneath the graph. There are pointers on the graph that correspond to the data point and its annotation.

Once the graph is loaded, users can use the mouse to interact with the chart. Hovering over a series displays a tool-tip that shows specific data for the point they are hovering over. They can also click and drag to zoom. Clicking on a data point brings up a prompt that allows a user to create a new annotation. New annotations are sent to a php file that then places the annotation in the database.



The top image shows our previous implementation while the bottom shows the final design.

index.html

This is the launching point for our program. It contains the html that layouts the divs and forms that the charts operate on. Any changes to the layout of the program need to be done in this file. Each chart needs to have a div container that is styled with the “chart_container” class.

This file includes the javascript file rsfd.js which contains the functionality for actually loading and displaying the charts.

By default the page initializes the forms to display the charts for the location U22 for a period of 7 days. To load the page with different settings the GET variables location and period must be set. For instance, to set the page to auto load the location D108 for a period of 60 the user could navigate to index.html?location=D108&period=60.

rsfd.js

This file contains most of the inner workings of our program. It uses jQuery to automatically start chart loading once the html finishes loading. The chart system is broken down into a model view controller system. The mvc is broken up into rsfd.ui, rsfd.data and rsfd.controller.

- **rsfd.ui -** Any function that interacts with the ui is placed in rsfd.ui. The ui functions use jQuery to quickly read values from the input form and to display prompts to the user.
- **rsfd.data** - The data functions all send JSON requests to the php files to get the data series that the charts need to display.
- **rsfd.chart** - The chart functions contain the functionality for loading and interacting with charts. Charts are initialized with no data. To add data the controller must get data using rsfd.data and then use the chart function `displayData()`. Other features of the chart object is to draw annotations and to display messages on top of the chart. The chart model also contains the even handler for when a user wants to add an annotation. The user is given a prompt to create a new annotation which is then sent to `annotation.php` which performs the database interactions.
- **rsfd.controller** - The controller contains the functionality that ties the data and ui together. To initialize the charts it gets the values the user has

entered in the form and gets the corresponding data which it then loads into a chart.

Work-flow starts in the read `$(document).ready` function where each chart is initialized and assigned to a div. The `controller.showData()` function is then called which starts the loading process for each chart. The data model gets the JSON objects which are then loaded into the charts. Further interaction with the charts is handled by highcharts.

annotation.php

This file contains the functionality for interacting with the annotations database. This is typically started by JSON requests sent from rsfd.js. This file supports getting, adding and deleting annotations from the database. Database credentials are obtained from annotation_settings.php.

setup.php

This file contains the functionality for getting the MySQL database information from the user and utilizing create_table.php to create annotation_settings.php

create_table.php

Runnning this file creates the annotations table. This file contains MySQL queries that use the information passed to it through the GET variables server, username, password, db_name and port. After creating the MySQL table it dynamically generates the annotation_settings.php file which stores the necessary information to connect with the Annotations table.

annotation_settings.php

This file contains the database settings. It is generated automatically, the client should not change the contents of this file by editing the file itself. If any changes are required the client should run setup again.

get_plot_data.php

Before we can display a chart we need this file to download and parse the data we want to display. The first step that this file takes to obtain the real-time data is to download the tab delimited data from USGS servers based on the specific parameters the user input. The data is then parsed into data structures that we can use further in the chart creation process. The parsing is done by realtimeParser.php. The data that is obtained from this process is encoded into a JSON object that the javascript can use to display a chart.

The type of data that is downloaded is specified by several parameters. The first is the location parameter which specifies what stream location to get data for. The period parameter specifies how many days to get data for. Finally, the chartType specifies what kind of data to download. This could be elevation, discharge or precipitation. These parameters are obtained via GET variables in the url.

get_simulation_data.php

Similar to get_plot_data.php, this file loads simulated data from the specified file location. It takes in the same parameters via GET variables with the addition of a variable called simLocation which tells the code where to load the simulated data from. The simulated data is read into an array and then parsed with simulationParser.php. The result is encoded into a JSON object that teh javascrip can use to display the simulated data on the charts.

get_simulationfile_names.php

This file browses the simulationfile directory for FEQ files and puts the file names into JSON object. This is used by the javascript to get the names of the FEQ files it should use.

get_site_offset.php

This file contains the offset values for the observed data. The observed data for most sites does not match the simulation values so our client gave us numbers for each location that the observed data must be offset by. Loading this file returns a JSON object containing the offset values for each site.

realtimeParser.php

This file contains the functionality for downloading and parsing the realtime data. The function `getFileAsArray` takes in a url to download the data from and returns an array where each value in the array corresponds to a line of the file downloaded. This data is then parsed into a multidimensional array where each column is a column of data from the original file.

simulationParser.php

This file contains the functionality for parsing an FEQ file. The function `parseFile` takes in a file location and a period. It reads data from that file and returns any data that is within the specified period.

site_no.php

This file contains the mapping from site numbers to their id. This is used when downloading real time data because the USGS servers require the id. For example, U22 is the site number while 05531175 is the id.

Adding Additional Location Support

We currently do not support easy additions to the number of sites that the program supports. To add a site that is currently not available a user must follow the following steps.

1. Obtain the site number (“U22”), site id (“05531175”), and site name (“Irving Road Stream”).
2. Open the code for index.html and add a new entry to the select form where the options value is the site number. For example, the select form starts with the following line:
   ```
   <select name="location" id="location">
   <option value="U22">U22</option>
   ```
   The option tag is the added location. Save the file.
3. Open the source corde for rsfd.js in the js folder. At the beginning of the file there is a list of site names. Add the following line to the end of the list:
   ```
   site_names["U22"] = "Irving Road";
   ```
   Save the file.
4. Open the source code for site_no.php. There is a list of entries that map site numbers to id’s. Add the following line to the beginning of that list:
   ```
   "U22" => "05531175",
   ```
   Save the file.
5. Open the source code for get_site_offset.php. There is a list of entries that map site numbers to offsets. Add the following line to the beginning of that list:

```
"U22" => 663.29,
```

Save the file.

If done correctly, the user should now be able to select the new location from the drop down form and view the data for that location.

Known Deficiencies

Below is a list of known issues with the application as it stands in its finished state.

- Caching: At the moment our web application does not cache the data that it parses into a database. Due to this reason the application needs to query and parse the same data each time the page is loaded even though the data related to that query might have been parsed previously. Though it does not slow down the application significantly, caching could significantly speed up the load time for pages which contain simulation data from flat files that have been parsed on previous instances of the website being loaded.
- Parsing FEQ file format: The current simulationParser is only able to parse FEQ files.
- Mobile device detection: The functionality for detecting mobile devices was created last semester and was fully functional. Due to the change in backend design and understanding the difficulty in comprehending touch event on a touch screen interface we have left out mobile device detection. However, even though it has been excluded from our current version, our web application functions correctly according to the requirements by USGS due to the nature of touch events on mobile devices.
- Locations: the current version of the application only displays data for 4 locations; however this is because we were not given the information regarding other locations by our clients. It should be straightforward for clients to add more

locations into the interface; our documentation includes comments that will explain how a developer can add more locations.

- Using PHP files to store constant values instead of a MySQL database: the application uses PHP files to store some constant values related to each location, these files when queried with specific parameters return JSON encoded data, however the correct approach would have been to store this information inside a database and query the table to get the required information.

How to Implement Deficiencies

Adding support for mobile device detection:

The installation folder already contains a script(“Mobile_detect.php”) that detects mobile devices, however it is not used since the necessary functionality that was USGS is functional without the use of that script. However, if the client required specific functionality for mobile devices it can be easily implemented through the use of the “Mobile_detect.php” script. To implement specific features related to mobile devices, the programmer will need to create a PHP script that detects whether the device is a computer or a mobile device by following the example presented below:

```
include("Mobile_Detect.php");
        $detect = new Mobile_Detect();


        if($detect->isMobile())
        {
                echo "Mobile";
        }
        else
        {
                echo "Computer";
        }
```

In this example the “Mobile_Detect.php” is included at the top and then a variable $detect is initialized to Mobile_Detect, then the function isMobile is called on the $detect variable. The function returns a boolean value which is true if the device is mobile, false otherwise, so in the example above, if the device is a smart phone the web page would display “Mobile” and if it is a computer then it will display “Computer”. After the correct device has been detected, return this information back to rsfd.js in JSON encoded format. The rsfd.js file will then decide based on this information what needs to be done further. Functionality to render the page differently will be incorporated into the rsfd.js file.

Adding support for editing annotations:

To add functionality to edit annotations, the functions will be added to rsfd.js, annotation.php and the index.html page. To be able to edit annotations the user will require a separate “Edit” button, this will be added in the index.html file. When this button is clicked it will trigger a function which should be located in rsfd.js which will query the database and return the text for the annotation in a pop up box text box. The user will be able to edit the text located inside the textbox. After the user is done they will click on a button that will call the functionality to edit the annotation. This will be located inside of rsfd.js which will in turn call a function inside annotation.php. Specifically a function needs to be created which looks up the value for the annotation according to the primary key(“id”), this function will then execute a MySQL query which should follow a pattern similar to the following example:

```
 $location     = addslashes($_GET["location"]);
 $chart_type   = addslashes($_GET["chartType"]);
 $series_name  = addslashes($_GET["seriesName"]);
 $time         = addslashes($_GET["timestamp"]);
 $annotation   = addslashes($_GET["content"]);

 $query = "REPLACE INTO Annotation (location, chart_type, series_name, time,
annotation) VALUES ('" . $location . "','" . $chart_type . "','" . $series_name . "','" . $time .
"','" . $annotation . "');";

 callDB($query);
```

In this example, the replace statement will replace the old entry in the Annotation table with the new values that are passed in to the “annotation.php” file. This will successfully edit the existing annotation with the new values.

Adding support for caching:

Create a new table to store values:

Create a new MySQL table to store the simulated and observed values for different charts.

Detecting the range:

Detect the time range specified by the user in the rsfd.js file where the user input is parsed. With the range known, query the MySQL table created above to see if it contains any data within the specified time range, if it does then create the chart using this information. However, if the required data does not exist in MySQL table then query the USGS website for observed data and parse the simulation flat files located in the simulationFiles folder to create the chart.

Adding support to parse formats other than FEQ:

Writing the parser:

Create a PHP file that will parse the file based on the formatting scheme that is being followed in the file. The file should be named similarly to “realtimeParser.php” or “simulationParser.php”. The input file which contains the simulation data will most likely contain the data that is tab or space separated. If the format is in the form of a regular expression, use the PHP preg_split function. If the format is not in the form of a regular expression then parse it using the PHP explode function. Take the values returned by these functions and store them in a global variable that can will be accessed by a separate file that will return this data encoded in the JSON format.

Returning the data:

Create another PHP file that will include the parser that was create in the step above. This file will organize the data such that it is stored according to location name and there is a unique value for elevation or discharge which corresponds to a time. It will return JSON encoded data when it is queried.

Recommendations for Improvements and Future Work

Google Maps Integration

The client expressed that if we had the time, it would be interesting and helpful for the application to have the ability to integrate Google Maps so that analysts at USGS could visually navigate the state's hydrologic data collection sites on a map and then view the plots for each site by clicking on the location, rather than selecting the name of the site from a list. This would certainly be a visually appealing landing page for the application, and would deliver many conveniences for the end user, but the team simply did not have the ability to integrate mapping given the time constraints. It is a great idea, and would be one of the most interesting features for consideration in the future.

Click-and-Drag

The client expressed the desire for the ability to click and drag the plot to move the graph to another time period along the X-axis. The application as it stands only has the ability to zoom in and zoom out to view different parts of the graph in detail. The liaisons would like the application to eventually have capabilities to move freely along the X-axis with a click-and-drag functionality when zoomed in to view a section of the graph in detail.

This would save time for the end user by allowing them to investigate other areas of data on-the-fly and not have to zoom out to reset the graph and zoom in again on the desired section of data.

Reporting System

After one of our later meetings with the client, we found out that a report system may have worked out better for what they were trying to achieve. We came across this when they asked for more in depth printing functionality. They wanted the print functionality because they print out and store a lot of the charts they make.

Their current system of cataloguing seems really out of date considering all of the data is already on their server. A lot of companies are now moving to paperless reporting and I think if we redesigned our system it would help USGS become a lot more efficient in the way they handle storing old data.

Our current design only lets users load data from the past 120 days which means annotation past that time frame become useless. Also, if a user wants to recreate a chart that they had made previously, they would have to manually enter all the same parameters and make sure that the right FEQ files are in the simulationFiles folder. This makes it difficult to look up past results.

If we had more time to work on this we would have liked to incorporate a save functionality that would save all the charts that they had on screen into a database. The The user would be able to name the report and open it again at a later time. So for example, the user loaded some charts and simulation files for a location and annotated it. They then could hit the save button and name the report. Later they could open the report and it would load all the data and annotations just how they had it. We think this would save them a lot of time and work.

To incorporate this we would have to spend a lot of time coming up with an efficient way to store the chart data. We would likely need to create a database table that stored the parameters for the chart. We would also need to create a table to hold each point of data in the chart. This would be tricky because most charts have thousands of points of data. We're not sure how we would create these databases but it would definitely be something worth looking into.

Better Location System

A major hurdle that we faced was trying to create a consistent representation of locations that we needed to get data from. It was very unclear how the USGS referred to the locations that they worked with. Even in discussions they would often become confused themselves about how to refer to the location. Each location has multiple identifiers, some of which may not be unique.

As result of this confusion, there are several disjointed parts in our code that refer to locations in different manners. Our php code that downloads data from the USGS servers uses what we call a location id which appears to be a unique eight digit number. Our javascript uses what we call a location number which is a alpha numeric identifier for a stream. The javascript also references a location title which is the English title that a location is given, such as “Irving Road”.

We also found out that different offices call the same streams by different names and id's. A USGS office in Northern Illinois might refer to U22 stream by D108. This means that any time we hard code the references, a user has to go into the source code to change the location id or name.

We would have liked to clean up the location name, number and id confusions. We had a difficult time trying to get a straight answer from the client about how they reference the locations. Our solution now requires the user to edit the source files in several places to change, remove or add locations. Obviously, this is not an optimal solution. Ideally there would be one config file that lists all the locations and mappings between the number, id, and name. This would make it a lot easier for transferring the program between locations since the user would only have to edit one easy to read file.

One way to implement this would be to have a php file that a user could navigate to that would display the current locations available. This list would display the id, number and name which the user would be able to edit. The user would also have a button that they could click to add and remove locations. This php page would then automatically update the config file with the new locations. This would make prevent users from having to modify source code which would additionally prevent them accidentally modifying vital code.

Cache System

The majority of the loading time comes from waiting on the php to download and parse the observed data. The current implementation requires the data to be downloaded and parsed for each chart type. This means that if the users wants to display discharge, precipitation and gage height, our system will download the observed data three times. The file that the system downloads contains the data for all three so there is a lot of wasted time re-downloading the same file.

A caching system would save a lot of time and cpu power because it would prevent the system from downloading redundant data. Ideally, the system would download data once and parse that into the three different graphs.

One possible and quick way to implement this would be to save the downloaded data into a temp file that the parsers could then read from. This of course would only store the latest chart data. A more involved system might include a database for storing multiple chart queries.

Automated Testing

The development team has looked into automated testing tools that will help test the robustness of the solution that has been created. However, due to time constraints, we have not prepared an automated testing suite for the application to be used in the future. Since the application will primarily be used for flood prevention measures, it is critical that the solution created is robust and accurate. An additional feature that was initially recommended by the liaison was the ability to view animated charts. The development team has not had the resources to assess available options in order to determine which would suit the given data best and would be visually appealing and informative, but it is something that should be taken into consideration as this application evolves.

Editing Annotations

The application currently only has functionality to add or delete annotations, however an important improvement to that would be the ability to edit annotations that have previously been entered. This is important as with more information a user could potentially add value to a previously entered annotation or could fix a mistake that has been made. Users can also add temporary notes and action items in the form of annotations on the graph, and colleagues would be able to update the notes as research assignments are completed.