Project Specifications

We are building a small tool into which our users will upload a set of data representing instrumentation readings at specified intervals. Once the data is collected, we will provide line graphs that compare the readings of various groups of the instruments over selectable time periods. The tool will be built in object-oriented PHP, and there is a MySQL database on the backend to store the data.

This tool will be for each user alone, so it does not (and should not) contain any code to facilitate search engine optimization, web crawling, or anything of the like. It is meant to be a simple, standalone tool, albeit web-based.

Note that each user will be identified by a UID, but this UID will already be known and passed to the tool as a parameter when the tool is opened. (So a login and/or user management system will *not* need to be built as part of this project)

Section One – Data Collection

The user will provide a CSV spreadsheet containing a number of columns of data. Each row will contain either a timestamp or a separate date column and time column, and a number of columns representing any of the following types of data:

- Floating point numbers (8 columns)
- Floating point numbers representing temperature readings (13 columns)
- Floating point numbers representing percentages (6 columns)
- Boolean values 1/0 (3 columns)
- A text value representing the location of the instrumentation from which the reading was gathered (2 columns)

As the information is proprietary, I am not providing specific column headers here. A sample spreadsheet will of course be provided to whomever wins the data collection portion of the project.

As mentioned, each row represents a number of instrument readings at a point in time, but any row of data may not necessarily contain a value for every column of data. (For instance, row 5 may contain instrument readings for today at noon for the first 5 columns, and row 6 may contain instrument readings for columns 6 - 10 at the same time.)

Please note that the user will be importing "sets" of data over time. Each set will represent instrument readings over a few days. The graphs (discussed in the next section) will compare instrument readings over time *within that set*. Later on, the user will return to the tool with another spreadsheet(s) that will represent another few days of data. We will *not* be comparing data across sets. So with each import,

the user must be able to provide a name that will represent that set of data, *or* select the name of an existing set to which the incoming data should be appended.

There will need to be some manipulation of the incoming data:

The user will need to indicate whether their temperature data is Fahrenheit or Celsius. (Their entire spreadsheet will be one or the other.) If the data is Celsius, we will need to convert it to Fahrenheit before storing it in the DB. (The original Celsius value can be discarded.)

One column of the temperature data may or may not be provided by the user, but if the value is not provided, it can be calculated from the values in three of the other columns in the spreadsheet. If the incoming field is blank, we will calculate the value for the user before storing it in the database. (This could also be accomplished as a routine after the data has been imported.)

Two nice-to-haves that are negotiable if the time-to-completion is affected:

- The user's spreadsheet may have the columns in any order. If possible, the user would indicate the order of their columns before uploading the spreadsheet. If this is not possible, we may constrain the user to providing their spreadsheet in a certain column order.

- Each row of data will represent a moment in time - all the other columns are instrument readings taken at that specific timestamp. A user may have the time represented in two columns (a date column and a time column) or in one column (a timestamp like "2013-11-04 19:34:34"). As with the temperature readings, the entire spreadsheet will be one or the other.

    - If possible, the program will auto-detect the format of the incoming rows and store the data correctly.
    - A second option would be that the user would indicate whether their date/time information is contained in one column or two, and the program will store the data correctly.
    - Finally, if necessary, we may constrain the user to providing their spreadsheet in a certain column order.

Once the data is imported, the user will need the following functionality:

They must be able to provide two times that represent their company's hours of operation (for instance, open at 8am and close at 6pm). This will be saved with their user record (the values will remain consistent for any data analysis) but the user should be able to edit the values at any time.

They must be able to view all of the data in any "set" of data provided. For each row, they should be able to edit any of the fields to provide missing or corrected data. They should be able to delete any row of data, or an entire set at once (preserving other sets that may be uploaded). They should be able to add a row of data to the set at any time, *even if they have not already uploaded data* (some users will be entering all of their information manually).

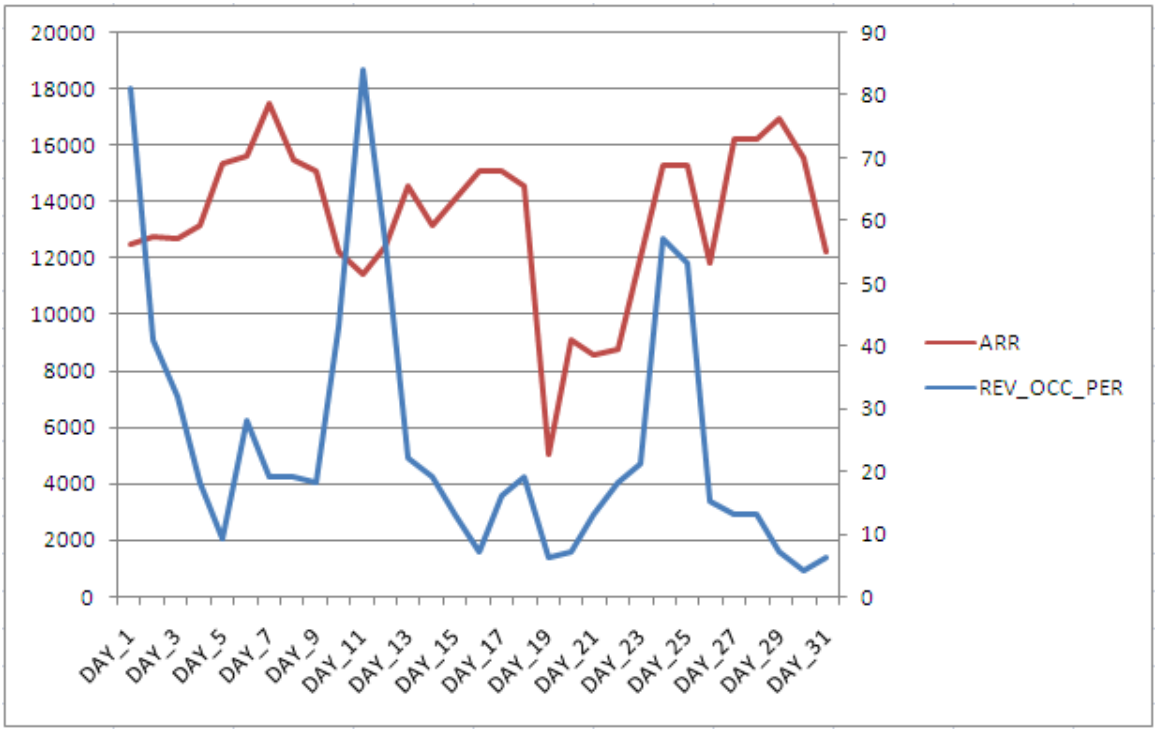
Section Two – Graph Analysis

All graphs provided will be line graphs, with the x-axis always representing time. Each graph will have a fixed number of time-slices – the width of each chart will be fixed. But the time period represented therein may differ; for instance, the graph may always contain 12 individual points on the x-axis, but the labels will represent every two hours if a 24-hour period is selected, or every 5 minutes if an hour-period is selected. Before each chart is created, the user will select the start and end dates/times for which they wish to view the selected chart, so the granularity of the x-axis labels must be determined on-the-fly.

The y-axis will represent the gradations of the data presented (for instance, a scale of 0 – 100 if the instrument provides a percentage, or a scale from 30 – 90 if the instrument reads degrees Fahrenheit). Some charts have two y-axes (for instance, the right-hand scale represents a percentage and the left-hand scale represents Boolean data).

The lines of the graph will represent the trending data of the instruments to be compared in the specific graph.

Each graph will have a *specific* function that will not change. There are 18 different charts, and each chart will compare a different set of instruments. As the information is proprietary, I will provide the specific instruments for each chart to whomever wins the graph section of the project.



example of two y-axes

If the user selected an open and close time for their company, the background of the chart should be shaded to indicate any times that fall within each of the times. For instance, if the user's business opens at 8 and closes at 6, and the graph shows data for a 24-hour period, there should be a light shading behind the area of the graph that falls between 8am and 6pm. This feature may be omitted if time-to-completion is threatened.

The Database Backend

Has already been built and information to perform MySQL queries, as well as the database structure etc. will be provided to whomever wins the project. We have full control over the database and its structure, so if any modifications need to be made to accommodate the functionality mentioned, those modifications can be quickly made.

Tool Specification

As mentioned, I would like the code to be written in object-oriented PHP. Any additional code (javascript, ajax, etc.) needed to support the desired functionality is acceptable, as are pre-defined libraries (i.e. to create charts). Simple CSS styling for the tool is also desirable, to enhance the user experience.