mpdb-client

Anthony Waters

February 2, 2008

Contents

0.1 Introduction

The part of the project mpdb-client is responsible for creating, display, and updating the user interface of the application. All of the source code within this project is converted to JavaScript through the use of Google Web Toolkit (GWT).

0.2 Bulk Upload

Bulk upload is the process in which a document in a preset format is parsed for data, basically a scientists adds a lot of samples to a spreadsheet and then the application parses the spreadsheet to add samples. This is beneficial because it allows the scientists to circumvent the process of adding the samples individually by hand. The process of performing a bulk upload can be summarized into four steps

1. Uploading the document
2. Parsing the document
3. Saving the objects in the database
4. Reporting the status of the operation (success/failure)

0.2.1 Uploading the Document

In order to transfer documents from the client to the server a file upload needs to occur. This is accomplished with a GWT widget named a `FormPanel`, this allows for sending POST request to the server. Another situation that this is used in is in upload images to the server. A very simple interface for testing bulk upload is located in the package `edu .rpi .metpetdb .client .ui .bulk .upload` the name of the class is `BulkUploadPanel`. If you look at it you will notice how simple it is, just a file upload control. The interesting thing to note here though is where the `FormPanel` posts to, which is `fp.setAction(GWT.getModuleBaseURL() + "/bulkUpload");`. What this means is it post to the bulkUpload servlet on the server, which brings us to the next section.

0.2.2 Parsing the Document

The post request is sent to the bulkUpload servlet which is located in the package `edu .rpi .metpetdb .server` within the class `BulkUploadServlet`. This is a simple servlet whos only function is to save the file on the server, and then based on the type of bulk upload it passes the file onto the parsers located in the package `edu .rpi .metpetdb .server .bulk .upload`. Parsing the files is a fairly simple task because there is a library already available from Apache called POI, which stands for "Poor Obfuscation Implementation". The part of the libary that is used for handling the spreadsheets is

called HSSF, which stands for "Horrible SpreadSheet Format" (it is refering to Microsoft Excel in this case). A simple tutorial on how to use POI and in particular HSSF is located at http://poi.apache.org/hssf/how-to.html. An example is also located in the source file `SampleParser.java` in the package `edu.rpi.metpetdb.server.bulk.upload.sample`. It works exactly like going through a table, you go row by row and column by column. So in order to parse the spreadsheet one would first read the first row, which contains the column headers, then subsequently read every other row parsing the data and determining what it is based on it's column header. How to actually save that data is discussed in the next section.

0.2.3 Saving the objects in the database

In order to explain this section I will use a Sample object as an example, therefore the user uploaded a spreadsheet that contains just samples. The java bean for the sample is located in the class `Sample` in the package `edu .rpi .metpetdb .client .model` this contains all of the properties for a sample that are read and written by the database. With that in mind in order to save the data to the database the following has to happen

1. Create a new sample object
    - `final Sample s = new Sample()`
2. Set the properties of the sample based on the data
    - `s.setAlias(‘‘My Lovely Sample’’);`
3. Save the sample to the database
    - `saveSample(s)`, located in `SampleServiceImpl`

The next step is to notify the user of the result of the operation.

0.2.4 Reporting the status of the operation (success/failure)

To send back a message to the client about the status of the operation you could do the following

1. If there was an error
    - throw an exception that will get passed to the client
    - `throw new InvalidFormatException`
    - the exception will contain the necessary details to will be displayed to the client about what caused the error
2. If it was successful
    - Return a string telling the user that is was successful
    - Add in things like what was added and links to them