How to work with Excel in Wicket using JExcel API

Table of contents

Prerequisites

This HOW-TO assumes that you have some working knowledge of Wicket, Models and some basic usage of JExcel API.
JExcel API can be downloaded at JExcel API Download Page

In this HOW-TO, we will upload an Excel File using wicket File upload and then render the Excel spreadsheet as an HTML table of labels.

1. Create a POJO to represent the basic attributes of the uploaded Excel Spreadsheet. The Number of columns and rows. This makes binding operations within Wicket Easy and also makes the Display Table auto re render with new uploaded sheets.

```
private class SheetMetaData implements Serializable {
        private int cols = 0;
        private int rows = 0;

        public int getCols() {
            return cols;
        }

        public void setCols(int cols) {
            this.cols = cols;
        }

        public int getRows() {
            return rows;
        }

        public void setRows(int rows) {
            this.rows = rows;
        }
    }
```

2. Create a WebPage holding the file upload form and the expected table

```
    transient Sheet sheet; // an instance of an Excel WorkSheet
    private SheetMetaData meta;

    public UploadExcel() {
        meta = new SheetMetaData (); //init Sheet Meta Data

        //create a file upload field
        final FileUploadField fup = new FileUploadField("fileUp");

        //create a form

        Form f = new Form("uploadform"){

            public void onSubmit(){
              //JExcel API Sheet Processing Logic

                Workbook wkb;

                FileUpload fupload = fup.getFileUpload();

                if(fupload != null)
                    if(fupload.getContentType().equalsIgnoreCase("application/vnd.ms-excel"))
                        try {
                            /*
                             *Streams directly from File upload into Workbook.getWorkbook(Inputstream)
                             */
                            wkb = Workbook.getWorkbook(fupload.getInputStream());
                            sheet = wkb.getSheet(0); //get First Work Sheet
                             /*
                             *Sets Sheet meta data. The HTML table creation needs this object to know about the
rows and columns
                             */
                            meta.setRows(sheet.getRows());
                            meta.setCols(sheet.getColumns());


                        } catch (Exception ex) {
                        }
            }
        };
        f.setMultiPart(true);

        f.add(fup);

        f.setMaxSize(Bytes.megabytes(5));

        add(f);

        add(createMainGrid());

        add(createHeadings());
}
```

4. Create the Loop that will generate the table from the Uploaded Spreadsheet Data Grid. Actually Wicket Models works magic and using models makes sure that your table re-draws each time another Excel Spreadsheet is uploaded within the same session. We will wrap the POJOs we have created above into models to use for creating the table.

HTML for rendering the Table based on Wicket Loop Class:

```
 <form wicket:id="uploadform">
            <input wicket:id="fileUp" type="file" name="file" />
            <input id="submit" type="submit" value="Upload Excel File" />
  </form>

<table>
<!-- the Heading Part -->
            <tr>
                <td>
                    <label>No.</label>
                </td>
                <td wicket:id="heading">
                 <label wicket:id="cellHead" ></label>
                </td>
            </tr>

<!-- the Excel Grid Body -->
            <tr wicket:id="rows">
                <td>
                    <!-- the Row Numbering Part -->
                    <label wicket:id="rowNo"></label>
                </td>
<!-- the Cell Data Part -->
                <td wicket:id="cols">
                    <label wicket:id="cellData"></label>
                </td>
            </tr>

        </table>
```

In Java: For the main data grid:

```
/*generating rows using the Loop class and the PropertyModel with SheetMetaData instance works magic
 *We bound the numbers of rows stored in SheetMetaData instance to the Loop using PropertyModel. No table will
 * be displayed before an upload.
 */
   private Loop createMainGrid(){
                //We create a Loop instance and uses PropertyModel to bind the Loop iteration to ExcelMetaData
"rows" value
        return new Loop("rows", new PropertyModel(meta,"rows")) {
            public void populateItem(LoopItem item) {

                final int row = item.getIteration();

                //creates the row numbers

                item.add(new Label("rowNo" , new Model(String.valueOf(row))));


                    //We create an inner Loop instance and uses PropertyModel to bind the Loop iteration to
ExcelMetaData "cols" value
                    item.add(new Loop("cols", new PropertyModel(meta,"cols")) {
                        public void populateItem(LoopItem item) {

                            final int col = item.getIteration();
                            /*this model used for Label component gets data from cell instance
                             * Because we are interacting directly with the sheet instance which gets updated
each time
                             * we upload a new Excel File, the value for each cell is automatically updated
                             */
                            IModel model=new Model() {

                               public Object getObject(Component c) {

                                    Cell cell = sheet.getCell(col, row);

                                    return cell.getContents();

                                }};
                                 Label cellData = new Label("cellData", model);

                                    item.add(cellData);

                        }
                    });
            }
        };


    }
```

Creating Headings for the table to follow Spreadsheet naming convention A, B, C, . Z, AA, AB, etc.

```
 private Loop createHeadings(){

        return new Loop("heading", new PropertyModel(meta,"cols")) {

            public void populateItem(final LoopItem item) {

                int column = item.getIteration();
                /** Copied from javax.swing.table.AbstractTableModel,
                 * to name columns using spreadsheet conventions:
                 *  A, B, C, . Z, AA, AB, etc.
                 */
                String colheading = "";

                for (; column >= 0; column = column / 26 - 1) {
                    colheading = (char)((char)(column%26)+'A') + colheading ;
                }

                item.add(new Label("cellHead", new Model(colheading )));

            }
        };
    }
```

To Make SpreadSheet Editable

To use textfields and another form around the Excel HTML Table so that Instant Editing and persistence can take place on Submit. The basic construct of every spreadsheet is to store data (Numbers, String, Dates etc) in rows and columns. So we create class to model each cell in the spreadsheet. Simply a POJO. You can enhance this object to suit your own persistence requirements or Object data processing schemes

```
public class XCell implements Serializable {

    public XCell() {
        data = null;
        persist = Boolean.TRUE;
    }
    public XCell(long rowNo, long colNo, Object cellData) {
        setRowId(rowNo);
        setColId(colNo);
        setData(cellData);
        setPersist(Boolean.TRUE);
    }
    private long rowId; //cell row location

    private long colId; //cell column location

    private Object data; //cell data

    //create getters and setters

}
```

2. In the ExcelUpload WebPage discuessed above, add this to the class members

```
private XCell[][] values;
```

This is a multi dimensional array that will automatically fill up as we move through the Excel file.

3. You will create an IModel for each textfield that will be created into the Table Grid. The IModel as used above will be updated to be backed by the XCell instance. So that when you hit submit, the XCell instance that was changed will be updated

In Java: For the main data grid with TextFields instead of Labels:

```
   private Loop createMainGrid(){
        return new Loop("rows", new PropertyModel(meta,"rows")) {
            public void populateItem(LoopItem item) {

                final int row = item.getIteration();

                //creates the row numbers
                item.add(new Label("rowNo" , new Model(String.valueOf(row))));


                    //generating columns for each row and binding the inner cells of the Excel table to XCell
instances in the array
                    item.add(new Loop("cols", new PropertyModel(meta,"cols")) {
                        public void populateItem(LoopItem item) {

                            final int col = item.getIteration();

                            IModel model=new Model() {
                               /*We will bind this IModel to each TextField and overide the setObjects and
getObjects method
                                *When the form is submitted, setObject is called, and then we assign the edited
data.
                                *
                                */
                               public void setObject(Component c, Object o) {
                                 /* basically there is no need for this conditional statement here because the
getObject is called
                                  * first which must have initiated the XCell instance within the array
                                  */
                                 if(values[row][col] != null)
                                    values[row][col].setData(o);
                                  else
                                  values[row][col] = new XCell(row, col,o);

                               }
                                /* The getObject is what determines what you will see in each textfield when
it renders as HTML.
                                  * It is called for each Textfield Creation.It is where we will create the
instances within the XCell
                                  * array. We will retreive the Cell content from the sheet.getCell() and then
assign its data into
                                  * XCell during XCell initialization.
                                  */

                               public Object getObject(Component c) {
                                   //at first creation before any edit and submit, the XCell instance in the
array is null
                                   if(values[row][col] != null)
                                       return values[row][col].getData();

                                   Cell cell = sheet.getCell(col, row);
                                   values[row][col] = new XCell(row, col,cell.getContents());
                                   return values[row][col].getData();
                               }
                            };
                             TextField tx = new TextField("cellField", model);

                             item.add(tx);

                        }
                    });
            }
        };


    }
```

3. Modify the HTML of the Loop and add a TextField instead of Label

```
 <form wicket:id="uploadform">
            <input wicket:id="fileUp" type="file" name="file" />
            <input id="submit" type="submit" value="Upload Excel File" />
  </form>


<form wicket:id="gridform">

<table>

<!-- the Heading Part -->
            <tr>
                <td>
                    <label>No.</label>
                </td>
                <td wicket:id="heading">
                 <label wicket:id="cellHead" ></label>
                </td>
            </tr>

<!-- the Main Grid Part -->
            <tr wicket:id="rows">
                <td>
                    <label wicket:id="rowNo"></label>
                </td>
                <td wicket:id="cols">
                    <!-- <label wicket:id="cellData"></label> -->
                    <input type="text" wicket:id="cellField" />
                </td>
            </tr>


<!-- Submit Button added -->
             <tr>
                <td>
                <input wicket:id="submit" type="submit" value="Submit" />
                </td>
            </tr>

        </table>

</form>
```

4. Modify the Java Code of the WebPage Component initialization to add up the new form components. Remember to re create the XCell array for each upload of a new Excel Sheet. This will be done in the onSubmit of the FileUpload Form.

```
    transient Sheet sheet; // an instance of an Excel WorkSheet
/**
*This multi dimension array will be used to map every generated textfield to a XCell instance.
*/
    private XCell[][] values;

    private SheetMetaData meta;

    public UploadExcel() {

        meta = new SheetMetaData (); //init Sheet Meta Data

        values = new XCell[0][0]; //no-upload-yet init
        Form f = new Form("uploadform"){
            public void onSubmit(){
                 /**
                   *The multi dimension array is re instantiated again based on the new sheet attributes
                   */
                   values = new XCell[sheet.getRows()][sheet.getColumns()];
                     submitGrid.setVisible(true);
                  } catch (Exception ex) {
               }
            }
        };
      //This new form component will hold the textfield gird
     Form grid = new Form("gridform"){
            public void onSubmit(){
                for(int i = 0; i < meta.getRows() ; i++)

                    for(int j = 0; j < meta.getCols() ; j++)
                        /*Delegate each cell processing class here e.g Move to Database, Persist with hibernate
or something
                         *On Submit, the XCell instances stored in the multi dimension array will be made
available here
                         */
                        System.out.println(values[i][j]);
            }
        };
        grid.add(createHeadings());
        grid.add(createLoop());
        submitGrid = new Button("submit");
        //if nothing is uploaded yet, the submit button wont show
        if(meta.getCols() == 0){
            submitGrid.setVisible(false);
        }
        grid.add(submitGrid);
        add(grid);
}
```

What we have just done is to work with Excel Spreadsheets in Wicket using JExcelAPI

You can download the source for these article at Example Source. Just attach to any existing Wicket samples

--dabar 15:20, 31 Aug 2006 (BST)