

Case Study: The Salmon Open Framework for Internet Applications (SOFIA)

Open Source JSP Development with Visual Tool Support

A common dilemma faced by developers creating web applications using Java technologies is choosing from a variety of J2EE APIs, design patterns, and development tools (see "The Many Sides of J2EE Development", JDJ Volume 6, Issue 11). Java developers need a cohesive application framework based on standard J2EE technologies that simplifies Java web development.

This article is a case study on using the SALMON Open Framework, an open source rapid application development tool for building J2EE-based web applications. SOFIA, developed by Salmon LLC, combines best-of-breed, off-the-shelf tools with an MVC framework to make building web applications easier. We chose SOFIA because it has been successfully used to develop many intranet and Internet applications both large and small, the largest being www.fujifilm.com, which gets 1,000,000 hits per day. In this article, we will examine the use of SOFIA for an Intranet application.

The Challenge

We work in the IT services department of a large insurance company and were requested to rewrite an old PC-based Finance application using a web-based Java solution. The project development team had a background based on mainframe technologies, with some client-server and web experience (HTML, ASP, basic Java and JSP).

Our first attempt to build a different server-side Java application ended in frustration due to several factors: JSP custom tags did not exist yet so our JSP pages were huge. Java scriptlets intermingled with HTML meant that organizing the JSP assets was cumbersome and Java debugging almost impossible. We couldn't believe that we were developing in a 'modern development environment' with hardly any visual tools! There had to be a better way ...

Good Timing

Our department had recently completed a fixed price pilot application with Salmon LLC, which they had built using their SALMON Open Framework as a foundation. As we intend for them to build a large complex custom server-side Java application, which we will maintain in-house, we figured the best way to learn about SOFIA was to build a small application with it. So, we decided to try it out for the rewrite of our PC-based Finance application.

The SOFIA Advantage

SOFIA enables us to develop our web pages in a best-of-breed design tool, using an extensive custom tag library. A big advantage of SOFIA is that the presentation layer does not contain any Java code. We use Java classes to manipulate the properties of components in the presentation and program to an event model similar to the AWT. SOFIA allows us to avoid time-consuming 'plumbing issues' like form submittal and many of the other tedious aspects of web navigation.

Out-of-the box support for Model-View-Controller

The Model-View-Controller design pattern is an established "best practice" for application design. It promotes a clean separation of business logic, presentation and request processing. SOFIA supports the MVC design pattern by using a SOFIA DataStore class for the Model, custom tags for the View, and SOFIA classes called controllers to handle user actions.

DataStore

The Datastore represents the Model component in SOFIA. It is a non-visual collection of data (rows and columns) from various sources (databases, javabeans, EJBs, and XML files). It works with many different database engines, as we learned from experience. We have so far connected to SQL Server, DB2 and Sybase with the same application, the only change being a few lines in a properties file. The Datastore has methods to handle the creation and execution of SQL statements, relieving the developer from hand coding SQL.

Custom tag library

SOFIA has a comprehensive library of HTML components for common HTML tags and form elements. The power of these components is that they can be bound to columns in a Datastore. For example, an HtmlTextEdit component (which corresponds to an "<input type='text'>" HTML tag) can be linked to a Datastore column, and any updates to that field in the GUI are reflected in the Datastore.

Controller

A Controller is a class extended from the SOFIA class JspController. A controller is associated with a JSP using the <salmon:page> tag at the top of the page. If the framework finds an instance variable in your controller with the same name and type as the component in the JSP, it will associate that instance variable with the component.

The SOFIA event model is where your Java code meets the user interface. SOFIA provides several event listener interfaces that can be implemented by controllers to make them aware of user actions. For example, the PageListener interface contains events that fire when a page is requested, and the SubmitListener interface has an event to handle when a user clicks a button.

Using Dreamweaver for GUI Presentation

The GUI components needed for the application are created using SOFIA's integration with Macromedia's Dreamweaver MX tool. Special SOFIA palettes allow drag-and-drop of the custom tags, eliminating hand typing of tags! As we place components on the page, their specific attributes are easily set within the tool. Dreamweaver dynamically interacts with the framework at design time to render the HTML for the custom tags to allow for full visual development. The nice thing about this approach is that the custom SOFIA components are integrated into Dreamweaver so GUI designers don't need to know the difference between standard HTML components and custom SOFIA components.

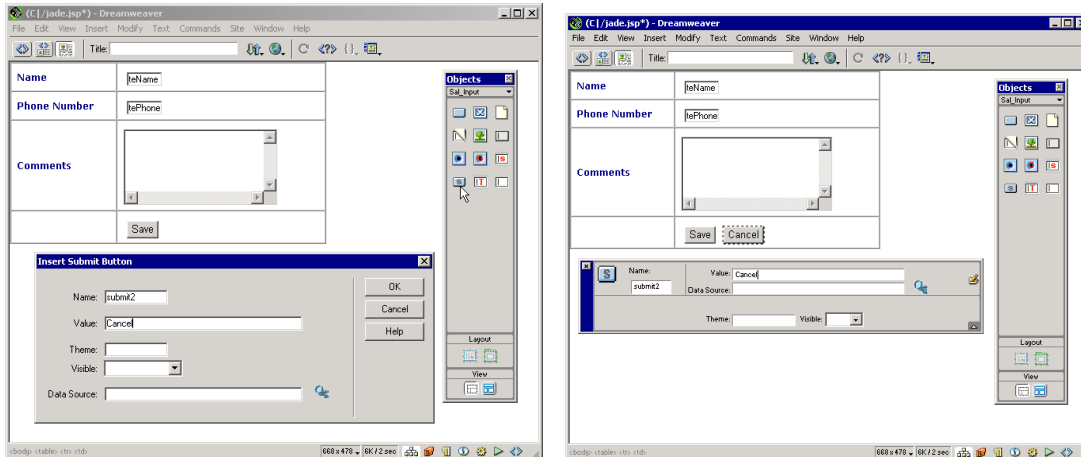


Figure 1: drag-and-drop GUI components using Dreamweaver

Java IDE Integration

We liked that SOFIA had already been proven to work with several best-of-breed IDE's, and as a development shop (not a tool vendor), Salmon shared our distaste for being locked into a particular IDE. Salmon's IDE of choice at first was IBM's VisualAge for Java, which we adopted, and used to complete our in-house development of the Finance application. Salmon transitioned to IDEA from IntelliJ, but we decided to go with the open source Eclipse IDE - SOFIA supports them both. We found that IDE features like code assist, local debugging, color-coding and so on increased our productivity and helped simplify the development experience. SOFIA's multi-IDE support means we can be flexible: if a developer has a strong preference for an IDE, they can go ahead and use it.

Construction of a JSP using SOFIA

We will walk through the process of designing a JSP page using SOFIA's integration with Macromedia Dreamweaver, and then create a controller class for the page so we can process user actions. This page will retrieve accounting entries from a database and allow users to update and delete records. We will show how easy it is to create a JSP with no embedded Java code in an intuitive, GUI drag-and-drop environment and then, using very little code, use the power of SOFIA's classes to create a dynamic web page.

The Model

First, we created an AccountingEntries class, which extends the SOFIA com.salmonllc.sql.Datastore class. By extending Datastore instead of defining the Datastore in the controller we can reuse our class. This task can be done manually, but SOFIA has a pop-up window (integrated into the Java IDE) that provides a GUI environment for creating models. The addColumn() method adds database fields to the Datastore. In the code below, addColumn() is passed the table name, the field name, the field's type, whether the field is part of the primary key, and whether the field is updateable. An interface containing table and field names is used so they don't have to be hard-coded into each page. We continue to add fields to the Datastore and set their properties. We add fields from another table and specify joins using the Datastore addJoin method, in which we specify which two fields comprise the join, and whether it is an outer (true) or inner (false) join. A section of our AccountingEntries class definition is below:

```

public class AccountingEntries extends com.salmonllc.sql.DataStore implements TableNames{

    public AccountingEntries() {

        /* a bucket is a column that can hold any type.  You can use it to hold data that's not
        * in the database.
        * The checkbox in the JSP has a datasource of bktSelected, and sets the value for
        * each row to 1 if the checkbox is checked, and 0 if it is not.
        */
        addBucket("bktSelected", DATATYPE_INT);

        //method takes table name, field name, data type, T/F for primary key, T/F for updateable.
        addColumn(ACCOUNTING_ENTRIES_TABLE, ACCTING_MM_FIELD, this.DATATYPE_STRING, true, true);
        addColumn(ACCOUNTING_ENTRIES_TABLE, ACCTING_YR_FIELD, this.DATATYPE_STRING, true, true);
        addColumn(ACCOUNTING_ENTRIES_TABLE, ACCT_NO_FIELD, this.DATATYPE_STRING, true, true);
        addColumn(ACCOUNTING_ENTRIES_TABLE, AMT_FIELD, this.DATATYPE_DOUBLE, false, true);
        //more column definitions...

        addColumn(RECORD_TYPE_TABLE, REC_TYPE_NAME_FIELD, this.DATATYPE_STRING,false, false);
        addColumn(RECORD_TYPE_TABLE, REC_TYPE_ID_FIELD, this.DATATYPE_STRING,false, false);

        //inner join
        addJoin(RECORD_TYPE_REC_TYPE_ID_TFC, LOSS_LAE_REC_TYPE_ID_TFC, false);

    }
}

```

Figure 2: AccountingEntries class

The View

Now that we have defined the Datastore for the page, we can open Dreamweaver and begin creating our JSP. Following common web page design, we have a banner at the top of the page and a navigation bar on the left. We start by making a copy of templatePage.jsp, which has a set of includes containing design elements common to our application. SOFIA extends the standard palette of objects in Dreamweaver. To use SOFIA's HTML components we choose one of the dropdowns starting with "Sal_" in the Objects window. For example, selecting the text Object will insert an HtmlIText component in your page:



Figure 3: Dreamweaver Object Palette

SOFIA's components work exactly like standard Dreamweaver components. After a component is dragged onto a page, a property window allows you to set attributes on the component. SOFIA inserts the custom tag code into the JSP and Dreamweaver displays the HTML rendered by the tag.

We add a Datasource tag to the page, which will create an instance of an AccountingEntries Datastore at runtime. We have several options when defining the type of Datasource we want to use. If we choose "SQL", SOFIA presents the tables and fields from our database to allow us to graphically create a Datasource. If you choose this option, the Datasource will be defined within the JSP. You can also choose "XML" to define an XML file as the input for the Datasource. Because we have already extended the Datastore class to present our Accounting Entries, we choose "MODEL" and specify the name of our class.

We continue to drag and drop visual components to the page for our presentation. We use a Datable tag to present the entries to the user. The Datable has several features that make it easy for users to work with data. It has a page selector component for navigating through multiple pages of results and allowing users to choose the number of rows to display per page. It allows sorting of rows based on column values when a column heading is clicked.

When the Datable is dragged onto the page, it presents a heading band that will display once at the top of the page, and a row band to define the data values. The row band will repeat once for every row retrieved by the Datasource query. In order to display the data, we add HtmlText and HtmlTextEdit components to the Datable. We use the Text component for read-only data, and TextEdits to allow the user to update data. Each component in the row band is linked to our Datasource using the "datasource" attribute.

When the design of the page is complete, it will look like this in Dreamweaver:

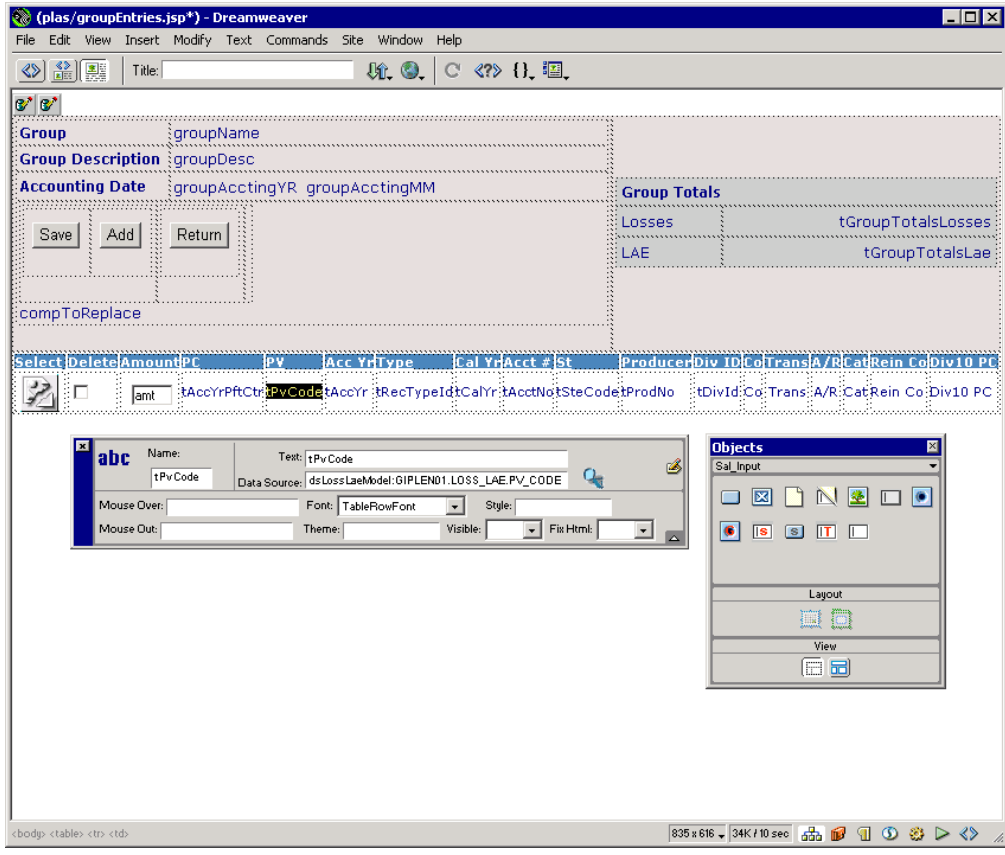


Figure 4: Property window in Dreamweaver

When our page design is complete, we need to create a controller class for the page. We add a link to a controller in the JSP by adding the following tag to the top of the page:

```
<fw:page controller=" com.mycompany.myapp.controller.GroupEntriesController" application="myapp" />
```

With SOFIA, a sophisticated GUI can be designed using standard HTML tags along with SOFIA's extensive set of components. A page designer doesn't need to know any Java, and doesn't need to design around blocks of inline Java and JavaScript. In our experience, we complete the majority of the design of the web pages early in development then rarely have to change anything in the presentation, since we can access all the components in the page in the controller class. SOFIA allows us to concentrate on what is important: the business logic of the application.

The controller

The controller class has two main functions. It has a handle to each component in the page, and the controller's initialize() method automatically fires the first time the browser accesses the page. We create instance variables in the controller for the components we want to reference in code. For example, the following tag may be in a JSP:

```
<fw:text name="text1" text="Hello" font="DefaultFont"/>
```

To get a handle to this component we add the following to our controller, and add code to the initialize method to change a property of our component:

```
package com.demo;

public class HelloWorldController extends com.salmonllc.jsp.JspController {
    public com.salmonllc.html.HtmlText _text1;

    public void initialize() {
        _text1.setText("Goodbye");
    }
}
```

The real power of the SOFIA controller comes from the event listener interfaces, contained in the com.salmonllc.html.events package. The PageListener interface has four methods that fire when a page is requested, and the SubmitListener interface has a method which fires when a user clicks a button. If we add a SOFIA submit button to our JSP, we could change the text when the user clicks the button:

```
package com.demo;

import com.salmonllc.html.*;
import com.salmonllc.html.events.*;
import com.salmonllc.jsp.*;

public class DemoController extends JspController implements SubmitListener {
    public com.salmonllc.html.HtmlSubmitButton _submit1;
    public com.salmonllc.html.HtmlText _text1;

    public void initialize(){
        _submit1.addSubmitListener(this);
    }

    public boolean submitPerformed(SubmitEvent e) throws Exception {
        _text1.setText("GoodBye");
        return true;
    }
}
```

As with the visual components, we have a handle to the JSP DataSource tag, which represents a non-visual collection of data. In the JSP, it is defined as:

```
<salmon:datasource
    name="dsAccountingEntries"
    type="MODEL"
    model="com.mycompany.myapp.datastore.AccountingEntries">
</salmon:datasource>
```

We reference the model class in code as:

```
public com.mycompany.myapp.datastore.AccountingEntries _dsAccountingEntries;
```

When the user visits our page, the pageRequested() fires and we get the parameters passed to the page. We pass these values to _dsAccountingEntries.retrieve(). The retrieve method takes a String containing fields and values, which it uses to construct a WHERE clause. The data is then retrieved into the Datastore. Because the text components in the JSP's Datatable are tied to the Datastore, SOFIA populates the Datatable with the results of the retrieval. The Datatable by default creates multiple pages for the results, so we didn't have to code anything to allow the user to step through multi-page result sets.

Below is the completed page, as it appears in our application:

Finance Application																								
Entries By Group Entries By Profit Center Group Maintenance User Administration Reports Home Help Logout		Group Entries List Accounting Date: 2002/05 Test																						
Group MYGROUP Group Description MYGROUP DESCRIPTION Accounting Date 2002 05 <input type="button" value="Save"/> <input type="button" value="Add"/> <input type="button" value="Return"/>										Group Totals Losses 0.00 LAE 0.00														
Select	Delete	Amount	Dr	Cr	Acc	Yr	Line	Gl	Yr	Acct #	SI	Product	Div	ID	Ch	Trans	A	P	Ch	Item	Co	Div	ID	
<input type="checkbox"/>	<input type="checkbox"/>	5.00																						
<input type="checkbox"/>	<input type="checkbox"/>	-5.00																						
<input type="checkbox"/>	<input type="checkbox"/>	1.00																						
<input type="checkbox"/>	<input type="checkbox"/>	-1.00																						
<input type="checkbox"/>	<input type="checkbox"/>	8.00																						
<input type="checkbox"/>	<input type="checkbox"/>	-7.00																						
<input type="checkbox"/>	<input type="checkbox"/>	12.00																						
<input type="checkbox"/>	<input type="checkbox"/>	-12.00																						
<input type="checkbox"/>	<input type="checkbox"/>	3.00																						
<input type="checkbox"/>	<input type="checkbox"/>	-3.00																						
Page 1 of 3		1 2 3																						
Total Rows: 30		Rows displayed per page: 10																						

Figure 5: Finished page

From this screen, users can click the Edit button to view and edit details for an entry (on a separate page), check the Delete boxes for entries they want to delete, and update the Amounts for entries. When the user clicks the Save button, the submitPerformed() event will fire on the controller:

```

public boolean submitPerformed(SubmitEvent e) throws Exception {
    super.submitPerformed(e);

    /* query the SubmitEvent to determine which button was clicked.
     * The SubmitEvent has a handle to which component caused the event.
     * We only show the processing for the Save and Yes buttons here.
     */
    if (e.getComponent() == _btnSave){
        boolean bBktSelected = false;

        //how many rows in our datastore?
        int iMaxRows = _dsAccountingEntries.getRowCount();
        int iDeleteRows = 0;

        //cycle through the datastore, checking value of Delete checkbox
        for (int j=0;j<iMaxRows;) {
            int iBktSelected = _dsAccountingEntries.getInt(j,"bktSelected");

            //if user blanked out the amount, set it to a valid double datatype.
            if (_dsAccountingEntries.getAny(j, LOSS_LAE_AMT_TFC) == null) {
                _dsAccountingEntries.setDouble(j, LOSS_LAE_AMT_TFC, 0.00);
            }

            // For each row, if the delete checkbox is checked, delete the row from the datastore,
            if (iBktSelected == 1) {
                _dsAccountingEntries.deleteRow(j);
                iDeleteRows++;
                bBktSelected = true;
                iMaxRows--; //decrement iMaxRows because one row was deleted from Datastore
            }
            else {
                j++;
            }
        }
        if (bBktSelected) { //rows were selected for delete, prompt user.
            _sMode = MODE_DELETE;
            _datatable1.setEnabled(false); //disable the datatable to prevent editing
            _chkDelete.setVisible(false); //hide the Delete button

            String sConfirmText = "Delete " + ((iDeleteRows == 1) ? "entry? ":"entries? ");
            _confirm.setText(sConfirmText);

            //set visibility and enabling of page components as necessary

```

```

//The Confirm component has Yes and No buttons, and a text component to confirm the
user's actions
_confirm.setVisible(true);
_btnAction.setEnabled(false);
_tblEntriesReturn.setVisible(false);
_tblEntriesMaint.setVisible(false);
} else {
//just update the amounts without prompting
try {
_dsAccountingEntries.update();
}
catch (Exception ex) {
displayError("Rows could not be changed. May no longer exist in DB or changed by
another user. " + ex);
return false;
}
}
}
}

```

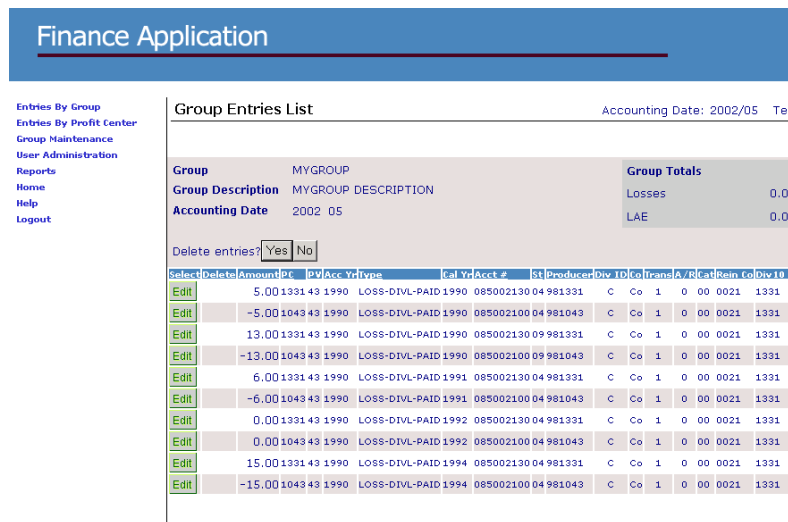


Figure 6: Prompting user to confirm deletes

If the user confirms the changes, the submitPerformed method again fires, this time executing code associated with the Yes button. If the No button is clicked, the data is re-retrieved from the database and the user can continue editing:

```

//This code executes when the user confirms changes by selecting the Yes button
if (e.getComponent() == _btnYes) {
/* We use a method to set the states of components on the page,
* such as whether components are visible or editable.
setBtnVis(e);

/* the update method will iterate through the datastore, compare its current contents
* to the original retrieve, and generate the necessary SQL statements to update the database.
*/
try { _dsLossLaeModel.update(); }
catch (Exception e) {
displayError("Rows could not be deleted or no longer in DB. " + ex);
return false;
}
_sMode = MODE_EDIT;
}

if (e.getComponent() == _btnNo) {
if (_sMode == MODE_DELETE) {

```

```

    setBtnVis(e);
    _dsLossLaeModel.retrieve();
}
}

```

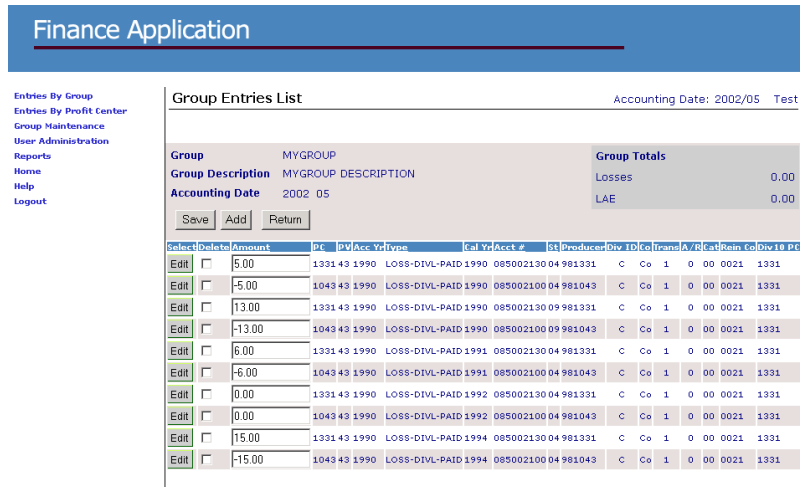


Figure 6: Page after changes are confirmed

The Datastore is tied to the JSP through the Datatable, so any changes made to the Datastore in code (update(), insert(), etc.) are the reflected in the presentation.

All components necessary for accomplishing a task are in the JSP. We use the event model to catch user actions and set the state of each component. SOFIA takes care of all the mechanics of refreshing the page and generating the HTML. By using SOFIA's custom tags, we avoid having to code huge JSPs with embedded Java code.

This page is very basic, showing only simple database updates. A more sophisticated page may have the controller interacting with EJBs or other business objects. It does however show how to work with the Datastore, and also how easy it is to create a page using SOFIA's integration with Dreamweaver.

Conclusion

The SOFIA framework enabled us to build and deploy a server-side Java application in a short amount of time. The SOFIA framework is easy to use and understand. Salmon LLC provided excellent documentation and examples to help us get our project started. With integration and support for best-of-breed tools, Dreamweaver to create the view and VisualAge for Java to develop the model and controller, SOFIA makes for a very productive visual development environment. The best feature of SOFIA is that it handles all the tedious plumbing associated with developing business applications allowing us to spend most of our effort satisfying the business requirements. We liked how SOFIA integrates with our existing tools to make development easier. Our users liked that we completed the project on time and on budget. Now that SOFIA is Open Source, all Java developers can benefit from this great foundation for developing web applications.

For more details visit www.salmonllc.com.

Listing 1: A JSP using SOFIA custom tags

```
<%@ taglib uri="/taglib.tld" prefix="fw"%>
<%@ page errorPage="ErrorPage.jsp" extends="com.salmonllc.jsp.JspServlet"%>
<fw:page controller=" com.mycompany.myapp.controller.GroupEntriesController" application="myapp" />
<jsp:include page="templateBefore.jsp" flush="true"/>
<jsp:include page="message.jsp" flush="true"/>
<fw:container name="mainCont">
<fw:datasource name="dsGroupInfo" type="SQL">
  <fw:datasourcedef>
    <fw:field tablename="GROUP_INFO" fieldname="ACCTING_YR"/>
    <fw:field tablename="GROUP_INFO" fieldname="ACCTING_MM"/>
    <fw:field tablename="GROUP_INFO" fieldname="GRP_NAME"/>
    <fw:field tablename="GROUP_INFO" fieldname="GRP_DESC"/>
  </fw:datasourcedef>
</fw:datasource>
<fw:datasource name="dsAccountingEntries" type="MODEL" model="com.mycompany.myapp.datastore.LossLaeModel">
</fw:datasource>
<fw:table name="table3" width="100%" border="0" bgcolor="#E0DCDC">
  <fw:tr>
    <fw:td width="500px">
      <fw:table name="table1" width="500px" border="0">
        <fw:tr>
          <fw:td width="130px">
            <fw:text name="text6" text="Group" visible="True" font="BoldFont"/>
          </fw:td>
          <fw:td width="370px">
            <fw:text name="tGroupName" text="groupName" visible="True"/>
          </fw:td>
        </fw:tr>
        <fw:tr>
          <fw:td width="130px">
            <fw:text name="text7" text="Group Description" visible="True" font="BoldFont"/>
          </fw:td>
          <fw:td width="370px">
            <fw:text name="tGroupDesc" text="groupDesc" visible="True"/>
          </fw:td>
        </fw:tr>
        <fw:tr>
          <fw:td width="130px">
            <fw:text name="text44" text="Accounting Date" visible="True" font="BoldFont"/>
          </fw:td>
          <fw:td width="370px">
            <fw:text name="tGroupAcctingYR" text="groupAcctingYR" visible="True"/>&nbsp;   <fw:text
name="tGroupAcctingMM" text="groupAcctingMM" visible="True"/>
          </fw:td>
        </fw:tr>
        <fw:tr>
          <fw:td colspan="3">
            <fw:table name="tblButtons" width="-1%" border="0">
              <fw:tr>
                <fw:td>
                  <fw:table name="tblEntriesMaint" width="-1%" border="0">
                    <fw:tr>
                      <fw:td>
                        <fw:input type="submit" name="btnSave" value="Save"></fw:input>
                      </fw:td>
                      <fw:td>
                        <fw:input type="submit" name="btnAdd" value="Add"></fw:input>
                      </fw:td>
                    </fw:tr>
                  </fw:table>
                </fw:td>
                <fw:td>
                  <fw:table name="tblEntriesReturn" width="-1%" border="0">
                    <fw:tr>
                      <fw:td>
                        <fw:input type="submit" name="btnReturn" value="Return"></fw:input>
                      </fw:td>
                    </fw:tr>
                  </fw:table>
                </fw:td>
              </fw:tr>
            </fw:table>
          </fw:td>
        </fw:tr>
      </fw:table>
    </fw:td>
  </fw:tr>
</fw:table>
```

```

        </fw:td>
      </fw:tr>
    </fw:table>
    <fw:text name="compToReplace" text="compToReplace"/>
  </fw:td>
</fw:tr>
</fw:table>
</fw:td>
<fw:td>
  <fw:table name="table9" width="100%" border="0" cellspacing="0" cellpadding="5" bgcolor="#CCCCCC"
height="100%">
    <fw:tr>
      <fw:td colspan="2">
        <fw:text name="text26" text="Group Totals" datasource="datasource5:ADJUSTMENT_IND.ADJ_IND_DESC"
font="BoldFont"/>
      </fw:td>
    </fw:tr>
    <fw:tr>
      <fw:td>
        <fw:text name="text27" text="Losses"/>
      </fw:td>
      <fw:td align="right">
        <fw:text name="tGroupTotalsLosses" text="tGroupTotalsLosses"/>
      </fw:td>
    </fw:tr>
    <fw:tr>
      <fw:td>
        <fw:text name="text28" text="LAE"/>
      </fw:td>
      <fw:td align="right">
        <fw:text name="tGroupTotalsLae" text="tGroupTotalsLae"/>
      </fw:td>
    </fw:tr>
  </fw:table>
</fw:td>
</fw:tr>
</fw:table>
<fw:datatable name="datatable1" width="100%" align="Left" datasource="dsAccountingEntries">
  <fw:datatableheader>
    <fw:tr>
      <fw:td>
        <fw:text name="tx99" text="Select" font="TableHeadingFont" />
      </fw:td>
      <fw:td>
        <fw:text name="tx94" text="Delete" font="TableHeadingFont" />
      </fw:td>
      <fw:td>
        <fw:text name="tx5" text="Amount" font="TableHeadingFont"/>
      </fw:td>
      <fw:td>
        <fw:text name="tx7" text="PC" font="TableHeadingFont"/>
      </fw:td>
      <fw:td>
        <fw:text name="tx8" text="PV" font="TableHeadingFont"/>
      </fw:td>
      <fw:td>
        <fw:text name="tx16" text="Acc Yr" font="TableHeadingFont"/>
      </fw:td>
      <fw:td>
        <fw:text name="tx10" text="Type" font="TableHeadingFont"/>
      </fw:td>
      <fw:td>
        <fw:text name="tx15" text="Cal Yr" font="TableHeadingFont"/>
      </fw:td>
      <fw:td>
        <fw:text name="tx11" text="Acct #" font="TableHeadingFont"/>
      </fw:td>
      <fw:td>
        <fw:text name="tx12" text="St" font="TableHeadingFont"/>
      </fw:td>
      <fw:td>
        <fw:text name="tx13" text="Producer" font="TableHeadingFont"/>
      </fw:td>
      <fw:td>

```

```

        <fw:text name="tx14" text="Div ID" font="TableHeadingFont"/>
    </fw:td>
<fw:td>
    <fw:text name="tx66" text="Co" font="TableHeadingFont"/>
</fw:td>
<fw:td>
    <fw:text name="tx17" text="Trans" font="TableHeadingFont"/>
</fw:td>
<fw:td>
    <fw:text name="tx18" text="A/R" font="TableHeadingFont"/>
</fw:td>
<fw:td>
    <fw:text name="tx19" text="Cat" font="TableHeadingFont"/>
</fw:td>
<fw:td>
    <fw:text name="tx20" text="Rein Co" font="TableHeadingFont"/>
</fw:td>
<fw:td>
    <fw:text name="tx21" text="Div10 PC" font="TableHeadingFont"/>
</fw:td>
</fw:tr>
</fw:datatableheader>
<fw:datatablerows>
<fw:tr>
<fw:td>
    <fw:input type="image" name="btnAction" value="Edit"></fw:input>
</fw:td>
<fw:td>
    <fw:input type="checkbox"
        name="chkDelete"
        checkedtruevalue="1"
        checkedfalsevalue="0"
        datasource="dsAccountingEntries:bktSelected">
    </fw:input>
</fw:td>
<fw:td align="right">
    <fw:input type="text" name="teAmt" value="amt" size="12" maxlength="16"
datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.AMT"></fw:input>
</fw:td>
<fw:td>
    <fw:text name="tAcc" text="tAccYrPftCtr" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.ACC_YR_PFT_CTR"
font="TableRowFont"/>
</fw:td>
<fw:td>
    <fw:text name="tPvCode" text="tPvCode" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.PV_CODE"
font="TableRowFont"/>
</fw:td>
<fw:td>
    <fw:text name="tAccYr" text="tAccYr" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.ACC_YR"
font="TableRowFont"/>
</fw:td>
<fw:td>
    <fw:text name="tRecTypeId" text="tRecTypeId"
datasource="dsAccountingEntries:GIPLN01.RECORD_TYPE.REC_TYPE_NAME" font="TableRowFont"/>
</fw:td>
<fw:td>
    <fw:text name="tCalYr" text="tCalYr" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.CAL_YR"
font="TableRowFont"/>
</fw:td>
<fw:td>
    <fw:text name="tAcctNo" text="tAcctNo" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.ACCT_NO"
font="TableRowFont"/>
</fw:td>
<fw:td>
    <fw:text name="tSteCode" text="tSteCode" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.STE_CODE"
font="TableRowFont"/>
</fw:td>
<fw:td>
    <fw:text name="tProdNo" text="tProdNo" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.PROD_NO"
font="TableRowFont"/>
</fw:td>
<fw:td align="center">
    <fw:text name="tDivId" text="tDivId" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.DIV_ID"
font="TableRowFont"/>

```

```

        </fw:td>
    <fw:td>
        <fw:text name="tCo" text="Co" font="TableRowFont"/>
    </fw:td>
    <fw:td align="center">
        <fw:text name="tTrans" text="Trans" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.TRANS_CODE"
font="TableRowFont"/>
    </fw:td>
    <fw:td align="center">
        <fw:text name="tArInd" text="A/R" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.ASND_RISK_IND"
font="TableRowFont"/>
    </fw:td>
    <fw:td>
        <fw:text name="tCatCode" text="Cat" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.CAT_CODE"
font="TableRowFont"/>
    </fw:td>
    <fw:td>
        <fw:text name="tReinCo" text="Rein Co" datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.REIN_CO_CODE"
font="TableRowFont"/>
    </fw:td>
    <fw:td>
        <fw:text name="tDiv10PC" text="Div10 PC"
datasource="dsAccountingEntries:GIPLN01.LOSS_LAE.DV10_ORIG_PFT_CTR" font="TableRowFont"/>
    </fw:td>
</fw:tr>
</fw:datatables>
<fw:datatablefooter>
</fw:datatablefooter>
</fw:datatable>
</fw:container>
<jsp:include page="templateAfter.jsp" flush="true"/>
<fw:endpage/>

```

Listing 2: AccountingEntries class

```

package com.mycompany.myapp.datastore.*;

import com.salmonllc.sql.*;
import com.mycompany.myapp.*;

public class AccountingEntries extends DataStore implements TableNames{

public AccountingEntries (String appName) {
    super(appName);

    addBucket("bktSelected", DATATYPE_INT);
    addColumn(AACCOUNTING_ENTRIES_TABLE, ACCTING_MM_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, ACCTING_YR_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, ACCT_NO_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, ACC_YR_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, ACC_YR_PFT_CTR_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, ADJ_IND_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, ASND_RISK_IND_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, CAL_YR_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, CAL_YR_PFT_CTR_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, CAT_CODE_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, CO_CODE_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, DIV_ID_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, DV10_ORIG_PFT_CTR_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, GRP_NAME_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, PROD_NO_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, PV_CODE_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, REC_TYPE_ID_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, REIN_CO_CODE_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, STE_CODE_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, TRANS_CODE_FIELD, this.DATATYPE_STRING, true, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, DV10_PC_SORT_IND_FIELD, this.DATATYPE_STRING, false, true);
    addColumn(AACCOUNTING_ENTRIES_TABLE, AMT_FIELD, this.DATATYPE_DOUBLE, false, true);

    addColumn(RECORD_TYPE_TABLE, REC_TYPE_NAME_FIELD, this.DATATYPE_STRING, false, false);
    addColumn(RECORD_TYPE_TABLE, REC_TYPE_ID_FIELD, this.DATATYPE_STRING, false, false);

    addColumn(GROUP_INFO_TABLE, GRP_DESC_FIELD, this.DATATYPE_STRING, false, false);

```

```

addJoin(RECORD_TYPE_REC_TYPE_ID_TFC, LOSS_LAE_REC_TYPE_ID_TFC, false);
addJoin(GROUP_INFO_GRP_NAME_TFC, LOSS_LAE_GRP_NAME_TFC, false);
addJoin(GROUP_INFO_ACCTING_MM_TFC, LOSS_LAE_ACCTING_MM_TFC, false);
addJoin(GROUP_INFO_ACCTING_YR_TFC, LOSS_LAE_ACCTING_YR_TFC, false);

    setFormat(LOSS_LAE_AMT_TFC, "###,###,##0.00");

}
}

```

Listing 3: EntriesListController class

```

package com.mycompany.myapp.controller.*;

//Salmon import statements
import com.salmonllc.jsp.*;
import com.salmonllc.html.*;
import com.salmonllc.html.events.*;
import com.salmonllc.*;
import com.salmonllc.sql.*;
import com.salmonllc.properties.*;
import com.mycompany.myapp.*;

//JAVA import statements
import java.io.*;
import java.util.*;
import java.lang.*;

public class GroupEntriesController extends BasePrivateController {

    String _sMode;
    String _sEntryType;

    //Instance variables for all components we need to reference in code

    public com.mycompany.myapp.datastore.LossLaeModel _dsAccountingEntries;
    public com.salmonllc.sql.DataStore _dsGroupInfo;

    public com.salmonllc.jsp.JspDataTable _datatable1;
    public com.salmonllc.jsp.JspTableCell _datatable1TDHeader1;
    public com.salmonllc.jsp.JspTableCell _datatable1TDRow1;

    public com.salmonllc.jsp.JspTable _tblEntriesMaint;
    public com.salmonllc.jsp.JspTable _tblEntriesReturn;

    public com.salmonllc.jsp.JspRowSelector _rsGroup;

    public com.salmonllc.html.HtmlText _tGroupDesc;
    public com.salmonllc.html.HtmlText _tGroupName;
    public com.salmonllc.html.HtmlText _tGroupAcctingYR;
    public com.salmonllc.html.HtmlText _tGroupAcctingMM;

    public com.salmonllc.html.HtmlText _tGroupTotalsLosses;
    public com.salmonllc.html.HtmlText _tGroupTotalsLae;

    public com.salmonllc.html.HtmlSubmitButton _btnAdd;
    public com.salmonllc.html.HtmlSubmitButton _btnSave;
    public com.salmonllc.html.HtmlSubmitImage _btnAction;
    public com.salmonllc.html.HtmlSubmitButton _btnReturn;

    public com.salmonllc.html.HtmlSubmitButton _btnYes;
    public com.salmonllc.html.HtmlSubmitButton _btnNo;

    public com.salmonllc.html.HtmlCheckBox _chkDelete;

    public com.salmonllc.html.HtmlTextEdit _compToReplace;
    ConfirmSection _confirm;

/**
 * Set button visibilities depending on the mode.
 */

```

```

public void setBtnVis() {

    if (_sMode.equals(MODE_EDIT)) {
        _datatable1TDRow1.setVisible(true);
        _datatable1TDHeader1.setVisible(true);
        _datatable1.setEnabled(true);
        _chkDelete.setVisible(true);
        _btnAction.setText("Edit");
        _tblEntriesMaint.setVisible(true);

    } else
        if (_sMode.equals(MODE_VIEW)) {
            _datatable1TDRow1.setVisible(false);
            _datatable1TDHeader1.setVisible(false);
            _datatable1.setEnabled(false);
            _btnAction.setText("View");
            _tblEntriesMaint.setVisible(false);
        }

    _tblEntriesReturn.setVisible(true); //always visible
}
/**
 * Set button visibilities depending on the mode.
 * @param e SubmitEvent
 */
public void setBtnVis(SubmitEvent e) {

    if (e.getComponent() == _btnNo) {
        _confirm.setVisible(false);
        if (_sMode == MODE_DELETE) {
            _datatable1.setEnabled(true);
            _chkDelete.setVisible(true);
            _btnAction.setEnabled(true);
            _tblEntriesReturn.setVisible(true);
            _tblEntriesMaint.setVisible(true);
        }
    } else
        if (e.getComponent() == _btnYes) {
            _confirm.setVisible(false);
            if (_sMode == MODE_DELETE) {
                _datatable1.setEnabled(true);
                _chkDelete.setVisible(true);
                _btnAction.setEnabled(true);
                _tblEntriesReturn.setVisible(true);
                _tblEntriesMaint.setVisible(true);
            }
        }
}

public boolean submitPerformed(SubmitEvent e) throws Exception {
    writeMessage(getClassName() + ".submitPerformed()");
    super.submitPerformed(e);

    _datatable1.setPage(0); //sets to first page for each search

    //if Delete or Update Amounts
    if (e.getComponent() == _btnSave){
        boolean bBktSelected = false;
        int iMaxRows = _dsAccountingEntries.getRowCount();
        int iDeleteRows = 0;
        for (int j=0;j<iMaxRows;) {
            int iBktSelected = _dsAccountingEntries.getInt(j,"bktSelected");
            //if the amount field is blank, make it a valid double
            double dAmt = _dsAccountingEntries.getDouble(j, LOSS_LAE_AMT_TFC);
            if (dAmt == 0.0) {
                _dsAccountingEntries.setDouble(j, LOSS_LAE_AMT_TFC, 0.00);
            }
            if (iBktSelected == 1) {
                _dsAccountingEntries.deleteRow(j);
                iDeleteRows++;
                bBktSelected = true;
                iMaxRows--; //decrement iMaxRows because one row was deleted from Datastore
            }
        }
        else {

```

```

        j++;
    }
}
if (bBktSelected) { //rows were selected for delete, prompt
    _sMode = MODE_DELETE;
    _datatable1.setEnabled(false);
    _chkDelete.setVisible(false);

    String sConfirmText = "Delete " + ((iDeleteRows == 1) ? "entry? ":"entries? ");
    _confirm.setText(sConfirmText);

    //set visibility and enabling of page components as necessary
    _confirm.setVisible(true);
    _btnAction.setEnabled(false);
    _tblEntriesReturn.setVisible(false);
    _tblEntriesMaint.setVisible(false);
} else {
    //just update the amounts without prompting
    try {
        _dsAccountingEntries.update();
    }
    catch (Exception ex) {
        displayError("Rows could not be changed. May no longer exist in DB or changed by
another user. " + ex);
        return false;
    }
}

}

//for all modes, pass entry type, group name, description, and accounting date
//encode(String) deprecated in 1.4, change to (String, "UTF-8") to add encoding scheme
String urlParms = ENTRY_TYPE + "=" + _sEntryType + "&";
urlParms += GRP_NAME_FIELD + "=" + java.net.URLEncoder.encode(_tGroupName.getText()) + "&";
urlParms += GRP_DESC_FIELD + "=" + java.net.URLEncoder.encode(_tGroupDesc.getText()) + "&";
urlParms += ACCTING_MM_FIELD + "=" + _tGroupAcctingMM.getText() + "&";
urlParms += ACCTING_YR_FIELD + "=" + _tGroupAcctingYR.getText();

if (e.getComponent() == _btnReturn) {

    //need to format link

    if (_sEntryType.equals(ENTRY_TYPE_GROUP)) {
        sendRedirect(GROUP_PAGE + "?NavBarId=navEntriesGroup&" + urlParms);
    } else if (_sEntryType.equals(ENTRY_TYPE_PC)) {
        sendRedirect(PC_ENTRIES_PAGE + "?" + urlParms);
    }
    return true;
}

//for add, modify or delete, always put the datastore in the session.
//MODE on detail will determine action on datastore.
//put datastore in session, pass row number selected and group name/desc
getSessionManager().setObject("LossLaeModel", null);
getSessionManager().setObject("LossLaeModel", _dsAccountingEntries);

//ADD
if (e.getComponent() == _btnAdd) {
    sendRedirect(
        GROUP_ENTRIES_DETAIL_PAGE + "?" + MODE + "=" + MODE_ADD + "&" + urlParms);
    return true;
}

//Build part of URL with passed data

//EDIT or VIEW
if (e.getComponent() == _btnAction) {
    sendRedirect(GROUP_ENTRIES_DETAIL_PAGE + "?rowId=" + e.getRow() + "&" + MODE
        + "=" + _sMode + "&" + urlParms);
    return true;
}

if (e.getComponent() == _btnYes) {

```

```

        if (_sMode == MODE_DELETE) {
            setBtnVis(e);
            try {
                _dsAccountingEntries.update();
            }
            catch (Exception ex) {
                displayError("Rows could not be deleted or no longer in DB. " + ex);
                return false;
            }
            _sMode = MODE_EDIT;
        }
    }

    if (e.getComponent() == _btnNo) {
        if (_sMode == MODE_DELETE) {
            setBtnVis(e);
            _dsAccountingEntries.retrieve();
        }
    }
    return true;
}

/**
 * runs first time user enters page.
 * Sets confirm section components and adds buttons to the submit listener
 *
 * @throws Exception
 */
public void initialize() throws Exception {
    super.initialize();
    /* The ConfirmSection component is an example of extending SOFIA components to provide more functionality.
     * The ConfirmSection component is created by giving it a name, the text to display, and a reference to
     * the controller.
     */
    _confirm = new ConfirmSection("confirm", "Confirm change?", this);
    //get yes and no buttons from ConfirmSection component
    _btnYes = _confirm.getYesButton();
    _btnNo = _confirm.getNoButton();

    //JSP sets a dummy value to placeholder for the confirm component.
    replaceComponent("compToReplace", _confirm);

    _confirm.setVisible(false);

    //add button listeners for all functions
    _btnSave.addSubmitListener(this);
    _btnAction.addSubmitListener(this);
    _btnAdd.addSubmitListener(this);
    _btnReturn.addSubmitListener(this);
    _btnYes.addSubmitListener(this);
    _btnNo.addSubmitListener(this);

    this.setHeading(HEADING_GROUP_ENTRIES_LIST_PAGE);
}

public void pageRequested(com.salmonllc.html.events.PageEvent e) throws Exception {
    writeMessage(getClassName() + ".pageRequested()");
    super.pageRequested(e);

    String cf = _props.getProperty(_props.CURRENCY_FORMAT);

    //if user comes to this page from another page then initialize as necessary
    if (!isReferredByCurrentPage()) {
        _chkDelete.setVisible(true);
        _tblEntriesMaint.setVisible(true);
        _confirm.setVisible(false);

        //get mode from page request
        String sMode = getParameter(MODE);

        String sAcctingYR = getParameter(ACCTING_YR_FIELD);

```

```

String sAcctingMM = getParameter(ACCTING_MM_FIELD);
String sGrpName = getParameter(GRP_NAME_FIELD);
String sGrpDesc = getParameter(GRP_DESC_FIELD);
_sEntryType = getParameter(ENTRY_TYPE);

//Error checking
if (sAcctingYR == null) {
    displayError("Accounting Year not passed");
    return;
}
if (sAcctingMM == null) {
    displayError("Accounting Month not passed");
    return;
}
if (sGrpName == null) {
    displayError("Group Name not passed");
    return;
}
if (sGrpDesc == null) {
    displayError("Group Description not passed");
    return;
}
if (_sEntryType == null) {
    displayError("Entry Type not passed");
}

//set display-only fields
_tGroupName.setText(sGrpName);
_tGroupDesc.setText(sGrpDesc);
_tGroupAcctingYR.setText(sAcctingYR);
_tGroupAcctingMM.setText(sAcctingMM);

//Retrieve data for datastore
String sCriteria = LOSS_LAE_ACCTING_YR_TFC + "=" + sAcctingYR + "";
sCriteria += " AND " + LOSS_LAE_ACCTING_MM_TFC + "=" + sAcctingMM + "";
sCriteria += " AND " + LOSS_LAE_GRP_NAME_TFC + "=" + sGrpName + "";

_dsAccountingEntries.setCriteria(sCriteria);
_dsAccountingEntries.retrieve();
//_dsAccountingEntries.waitForRetrieve();

//Check if any data is retrieved to the datastore.
if (!_dsAccountingEntries.gotoFirst()) {
    displayError("There is no data available.");
    return;
}

//format amount field
//_dsAccountingEntries.setFormat(LOSS_LAE_AMT_TFC, cf);

//if the date passed is not current, make amounts read-only
String dateParam = sAcctingYR + sAcctingMM;

//Sets the text value of the submit buttons for each row to "View" if a previous accounting period,
//and "Edit" if the current accounting period.
if (dateParam.equals(AppDate.getCurYr() + AppDate.getCurMM())) {
    _sMode = MODE_EDIT;
    setBtnVis();
} else {
    //if previous acct date, view only
    _sMode = MODE_VIEW;
    setBtnVis();
}
_tblEntriesReturn.setVisible(true); //always visible
}
}
}

```