**Paper 49-28**

# Developing Data-Driven Applications Using JDBC and Java Servlet/JSP Technologies

Chad Ferguson, SAS Institute Inc., Cary, NC
Sandra Brey, SAS Institute Inc., Cary, NC

## ABSTRACT

Many enhancements have been made to the SAS Java component library available with Version 3 of AppDev Studio™ (ADS). Several of the improvements include the development of new TransformationBeans™ and JDBC™ adapters for building custom table and form-based, data-driven applications. This paper will provide server-side Java examples that you can use to build Web applications which will allow the end user to:

- edit data via a table or a custom form
- sort the data
- navigate through the data using a flexible NavigationBar component
- filter the data
- add computed items
- export the data to Excel
- and much more.

## INTRODUCTION

Due to the increased need for Web-based applications, Version 3 of ADS includes enhanced support for developing data-driven Web applications using Java Servlet™ and Java Server Pages™ (JSP) technology. The new release includes new component viewers that can access relational data sources via JDBC™ technology. Using WebAF™, AppDev Studio's interactive Java development environment, you can easily create a Java Servlet or JSP project by dragging and dropping the desired components onto the servlet or JSP. The servlet or custom tag code will be generated automatically for you.

This paper will focus on the powerful TableView component and how it is used to display data from JDBC data sources. Other components in the SAS Java component library will be described as well, such as the DataBeanForm, JDBC adapters, and various data selectors.

Extending on the TableView component, the TableViewComposite is a composite component. It is made up of several sub-components, including the TableView, which assist in the manipulation of the data. The NavigationBar component, for example, assists the TableView in scrolling through the rows or columns of data. The TableView allows you to set the number of rows and/or columns that will be visible on each page. Using the visible controls displayed by the NavigationBar, the end user can scroll forward and backwards through the pages of data. Another component, the RelationalMenuBar, gives you an area in the table to display any desired data selectors. The data selectors work in conjunction with the TableView to allow functionality such as filtering the data, adding computed items, sorting the data and exporting the data to Excel. The TableView also allows you to set cell renderers on a per cell basis. This gives you a way to customize the TableView and display your data any way that you like.

An exciting new feature in the TableView for Version 3 is editing. This allows you to build Web-based servlet or JSP applications which enable the end user to edit data via HTML. The data may be edited one row at a time or all at once (like a spreadsheet). When editing, you can set a cell editor which gives you more control over valid values. For example, you can use a ChoiceBox as your cell editor and the ChoiceBox will contain a list of valid values to choose from while editing a particular cell. Or you can use a TextEntry as your cell editor and then you can set a validator to maintain the valid values during the data entry process.

The TableView's model must implement Java's Swing TableModel interface. The SAS component library provides multiple adapters for the TableView that allow it to connect to different data sources. One of these adapters is the JDBCToTableModelAdapter, which allows the TableView to take advantage of the JDBC technology and communicate with a relational database or data source.

If you are interested in building Web applications for data entry or editing, you will also want to take a look at the DataBeanForm component. This component is also new to Version 3 and allows the end user to view and edit data from a JDBC data source as well. Unlike the TableView component which presents data in a spreadsheet view, the DataBeanForm is used to build form-based Web applications that display or edit one row of data at a time.

The DataBeanForm also takes advantage of the NavigationBar component's functionality, allowing the end user to scroll through the data row by row. Using the GoTo field which is part of the NavigationBar, the end user can also enter the row number they want to scroll directly to – a short cut which quickly gets them to the right place in their data.

The model for the DataBeanForm is a databean generated by WebAF's DataBean Wizard. The DataBean Wizard creates a JavaBean which models a row in your data source and provides properties for selected columns in the data source. The databean's properties are then set on various form elements within the DataBeanForm.

The JDBC adapters that are new with Version 3 of ADS allow JDBC data sources to be models for a variety of viewers requiring Java's Swing models. In addition to the TableView, the JDBCToTableModelAdapter can also be used by many of the new chart components in ADS. Other JDBC adapters available in ADS are:

- JDBCToTreeModelAdapter for viewers requiring a Java Swing TreeModel
- JDBCToListModelAdapter for viewers requiring a Java Swing ListModel
- JDBCToComboBoxModelAdapter for viewers requiring a Java Swing ListModel.

Also new in Version 3 of ADS is a set of BusinessQuery adapters. These adapters use an Intelligent Query technology which provides a business view of the data where the end user is unaware of where physical data is located. Similar to the JDBC adapters, these will allow you to use Intelligent Query data sources as models for a variety of viewers. The TableView, for example, uses the BusinessQueryToTableModelAdapter.

The examples in this paper will use tag extensions from the SAS Custom Tag Library. The tag extensions allow you to take advantage of the TransformationBeans without knowing the required APIs for the underlying TransformationBeans. For more information on SAS custom tags, refer to the paper, *Introduction to the SAS Custom Tag Library*.

The examples will also make use of the Model 2 or M-V-C (Model-View-Controller) architecture. The Model 2 architecture allows you

to separate the business logic from the presentation. Basically, the code needed to control the flow of the application or initialization of the data models can be put into a servlet. Other responsibilities of the controller servlet include managing the application state and verifying authentication and security. The JSP should be responsible for producing the HTML for the Web browser. Thus, the code for the viewers, such as the TableView, should be put into the JSP.

## CONNECTING TO A JDBC DATA SOURCE

New to Version 3 of ADS are a group of adapters that translate the JDBC data model into one of Java's Swing models. These adapters can be found in the com.sas.storage.jdbc package of our API. These adapters help in translating the JDBC data models into a model that can be used by one of our many viewers.

The JDBC adapters require a model object that implements the java.sql.Connection interface, such as the com.sas.storage.jdbc.JCBCConnection. The JDBCConnection class allows you to set the information needed to create a java.sql.Connection object, which it creates using the appropriate java.sql.DriverManager create methods.

In addition to the Connection object, the adapters also need the sql query select statement specified. The adapters can connect to any type of JDBC data source.

The following sample of code shows how to connect to a SAS DataSet through the SAS IOM Server. The instantiated JDBCToTableModelAdapter can then be used as the model for the TableView component.

```
HttpSession session = request.getSession();
JDBCToTableModelAdapter tableModel =
  (JDBCToTableModelAdapter)session.
  getAttribute("tableModel");
if (tableModel == null)
{
  JDBCConnection connection = new
    JDBCConnection();
  connection.setDriverName(
    "com.sas.rio.MVADriver");
  connection.setDatabaseURL(
    "jdbc:sasiom://localhost:5310");
  String query = "select * from
    sasuser.class";
  tableModel = new JDBCToTableModelAdapter(
    connection, query);
  session.setAttribute("tableModel",
    tableModel);
}
```

The JDBCToTableModelAdapter and TableView will work best with JDBC drivers that create scrollable result sets. This is because the TableView will need to scroll backwards to support all the scrolling capabilities. However, the JDBCToTableModelAdapter can handle the scroll backward features of the TableView even if the JDBC driver supports forward-only result sets (ResultSet.TYPE_FORWARD_ONLY). This is accomplished by reapplying the query when needed. There is a performance penalty for doing this since the server has to reapply the query and adapter has to be reinitialized.

In the previous example, you will notice that we check the session object for an instance of the JDBCToTableModelAdapter and then later we set the instance of the JDBCToTableModelAdapter on the session object. This is done to prevent a new instance of the JDBCToTableModelAdapter every time. If a new instance of the JDBCToTableModelAdapter is created every time, then the TableView would always display the data in the initial state and any manipulation of the data would be lost. Sections of code such as this should be added to the controller servlet.

## DISPLAYING THE DATA

Now that you have connected to a JDBC data source, the next step is to display the data in a viewer such as the TableView component. The TableView component displays the data in an HTML table. In the following example, we assume the controller servlet has already been set up and an instance of the JDBCToTableModelAdapter has been place on the session object as "tableModel". We use the TableView custom tag and display all rows and columns of the data source. Then we simply attach the data model to the TableView and entire data source is displayed in the browser.

```
<sas:TableView id="tableView"
 model="tableModel" scope="session"\>
```

| | Employee Name | Employee Id | Job Title | Company | Depart |
|---|---|---|---|---|---|
| 1 | Patrick Lu | 120101 | Director | Orion Austalia | Sales Man |
| 2 | Tom Zhou | 120102 | Sales Manager | Orion Austalia | Sales Man |
| 3 | Wilson Dawes | 120103 | Sales Manager | Orion Austalia | Sales Man |
| 4 | Kareen Billington | 120104 | Clerk II | Orion Australia | Administra |
| 5 | Liz Povey | 120105 | Secretary I | Orion Australia | Administra |
| 6 | John Hornsey | 120106 | Office Assistant II | Orion Australia | Administra |
| 7 | Sherie Sheedy | 120107 | Office Assistant III | Orion Australia | Administra |
| 8 | Gladys Gromek | 120108 | Warehouse Assistant II | Orion Australia | Administra |
| 9 | Gabriele Baker | 120109 | Warehouse Assistant I | Orion Australia | Administra |

**Figure 1. Default TableView**

As you can see in the image, the table is displayed with the default look. The table can be modified with different html styles to change the default look. There are also cell renderers that can be used on a per cell basis to modify the default look of the TableView. For more information on what cell renderers are available and how they are used you can view the API for the com.sas.servlet.tbeans.tableview.html package.

## ACTIONS ON THE VIEWER

To take full advantage of the functionality of the TableView, you can use the TableViewComposite component along with the ActionProvider framework. Using all this together, the default actions on the TableView and NavigationBars are active. These actions include sorting and scrolling.

The TableViewComposite is a composite component which is a group of subcomponents. The subcomponents include the TableView component, NavigationBar components, and a MenuBar component. The components work in conjunction to manipulate the data source. For example, the NavigationBar provides controls to scroll back and forth through the rows and/or columns of the data source.

The ActionProvider framework is an integrated set of classes that work with JSP component viewers to provide actions for each renderable portion of the view. The framework also provides mechanisms for customizing viewer functionality. For more information on the ActionProvider framework, you can refer to the HttpActionProvider API. The recommended use of the ActionProvider is to place it in the controller servlet of your application. The code would look like:

```
HttpActionProvider actionProvider =
session.getAttribute("actionProvider");
if (actionProvider == null)
{
```

```
    actionProvider = new HttpActionProvider();
    session.setAttribute("actionProvider",
      actionProvider);
  }
  else
  {
    actionProvider.executeCommand(request,
      response, response.getWriter());
  }
```

Once again you will see that we use the session object and place the ActionProvider on the session. This is so that the custom tags can use and reference the ActionProvider. Now that we have an ActionProvider and JDBCToTableModelAdapter, we can put it all together in the JSP with the TableViewComposite.

```
<sas:TableViewComposite id="tableComposite"
 actionProvider="actionProvider"
 model="tableModel" scope="session" />
```

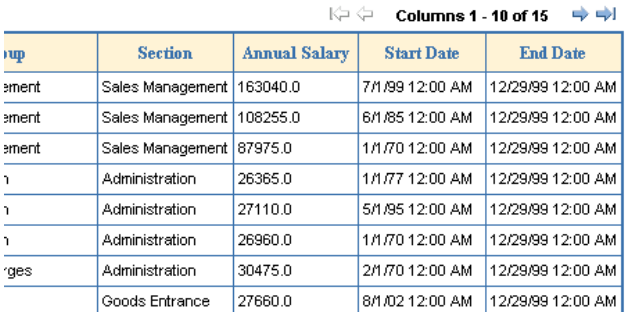


**Figure 2. Column Scrollers**

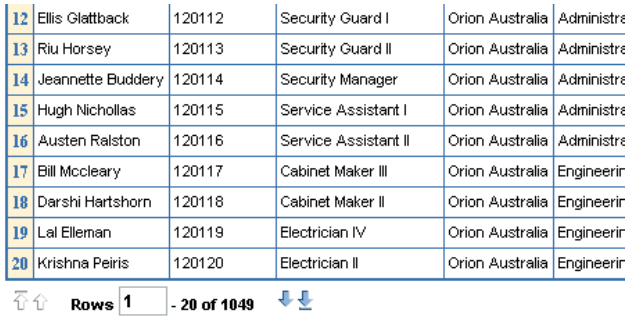


**Figure 3. Row Scrollers**


**Figure 4. Default Pop-up Menus**

The previous three figures show the default actions that become visible when using an ActionProvider and JDBCToTableModelAdapter. In figures 2 and 3, you can see the default NavigationBar that displays at both the top and bottom of the table. The NavigationBar enables the user to easily scroll through the remaining rows or columns that are not currently visible in the table. Figure 4 shows the default items that are available on the popup menu when the user clicks on a column label in the table. Using these actions you can

- Sort the table by a specific column. Nested sorting is supported by default as you continue to click on additional columns. The last column selected is used as the primary sort key.
- Rearranging the order of the columns in the table using the Move command.

## BUSINESSQUERY ADAPTERS

In addition to the JDBC adapters, there are also a set of BusinessQuery adapters. The BusinessQuery adapters help integrate the viewers with the new Business Intelligence Platform. Before reviewing these adapters, a brief summary of the BusinessQuery is useful. The BusinessQuery is an integral part of the Intelligent Query Services. Intelligent Query (IQ) is a set of components that provide all aspects of query functionality resulting in the correct information to the right people. A key aspect of this functionality is meta-driven queries. IQ includes components that leverage the information provided by the metadata, components for creating and updating this metadata, as well as ad-hoc query components. IQ leverages the knowledge stored in the physical data and metadata. Some examples include where data is stored, joins, calculations, and database specific knowledge. The BusinessQuery covers both relational tables and cubes.

These BusinessQuery adapters work in much the same way as the JDBC adapters. Instead of beginning with a JDBC ResultSet, you can start with a BusinessQuery. That BusinessQuery can then be transformed, via the adapters, into a number of different models. For example, the TableView can use the BusinessQueryToTableModelAdapter. There are many advantages to using IQ technology in a Web application, such as the ability to Filter and Rank the results. While this technology is new it builds on the JDBC technology. The Query is defined through the IQ Service, but the relational data is returned to the application in the form of a JDBC ResultSet. The adapter conveniently executes the BusinessQuery and transforms the JDBC ResultSet to whatever model is needed by the various viewers.

The next code sample shows how to connect to a data source via one of the new BusinessQuery adapters:

```
BusinessQueryToTableModelAdapter tableModel =
  (BusinessQueryToTableModelAdapter)
  session.getAttribute("tableModel");
if (tableModel == null)
{
  System.setProperty("java.security.policy",
    "C:\\AppDevStudio\\WebAF\\
    resources\\policy");
  System.setProperty(
    "java.security.auth.policy",
    "C:\\AppDevStudio\\WebAF\\
    resources\\auth.policy");
  System.setProperty(
    "java.security.auth.login.config",
    "C:\\AppDevStudio\\WebAF\\
    resources\\login.config");
  System.setProperty("cache.auth.policy",
    "true");

  PFSUtilities.deployPlatformServices(
    "file:\\C:\\AppDevStudio\\WebAF\\
    resources\\sas_service_deployment_
    export_queryandReporting_1.xml",
    "Query and Reporting",
    "BIP Core Services");

  IntelligentQueryMetadataService
```

```
    queryService =
    (IntelligentQueryMetadataService)
    IntelligentQueryMetadataServiceFactory.
    newService();

  SessionContextInterface sessionContext =
    PFSUtilities.loginUser("omruser",
    "DemoDemo1", "carynt");

  InformationMap informationMap =
    queryService.getInformationMap(
    sessionContext, new PathURL(
    "SBIP://Repository/BIP Tree/
    Organization(BriefInformationMap)"));

  DataSelection dataSelection =
    DataSelectionFactory.newDataSelection(
    informationMap);

  List items = informationMap.getObjects(
    false, DataItem.class);
  dataSelection.addResultItems(items,
    Role.COLUMN);

  BusinessQueryToTableModelAdapter tableModel
    = new BusinessQueryToTableModelAdapter(
    dataSelection);
  session.setAttribute("tableModel",
    tableModel);
}
```

The following images show the default actions that are available when connecting to the BusinessQueryToTableModelAdapter.

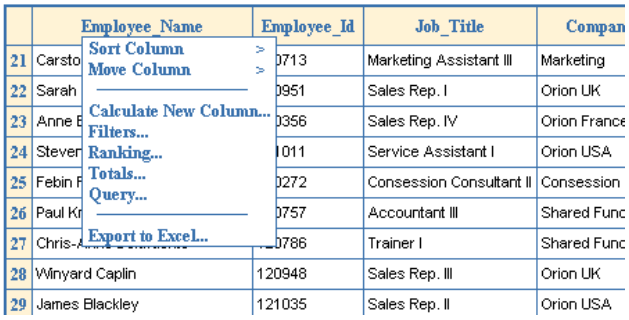


**Figure 5. BusinessQuery Pop-up Menus**

Because of the IQ layer that we've added to this example, there are many more default actions available to the end user when the popup menu is invoked on an item in the table. Along with sorting and moving items, there are default selectors that display. These selectors enable the end user to:

- Add new calculated items
- Create filters to subset the data displayed in the table
- Apply ranking criteria to view things like "top 10" or "bottom 5 percent"
- Specify row or column totals
- Change what is displayed on the columns and rows of the table using the Query selector
- Export data to excel

## EDITING

The new TableView in Version 3 allows you to create an HTML editable table. The TableView has two modes of editing:

- single row mode which allows the end user to edit one row at a time
- all row mode which allows all rows to be edited at once(similar to a spreadsheet).

The default mode is to edit in the single row mode. To enable editing, editing options must be set on both the model must be enabled on the model (JDBCToTableModelAdapter) and the viewer (TableView). The following code shows how to enable editing on the model:

```
tableModel = new
JDBCToTableModelAdapter(connection, query);
tableModel.setReadOnly(false);
```

and on the viewer:

```
<sas:TableViewComposite id="tableComposite"
 actionProvider="actionProvider"
 model="tableModel" scope="session">
   <sas:TableView>
      <sas:Edit enabled="true" />
   </sas:TableView>
</sas:TableViewComposite>
```

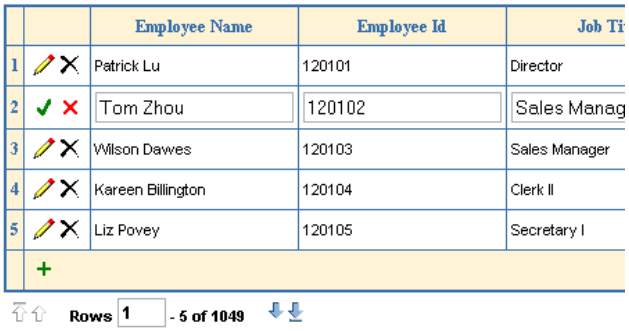


**Figure 6. SingleRow Edit Mode**

In Figure 6 you can see the edit actions enabled. Each row has a set of actions to edit that row. In this figure, you can notice that row 2 is the only row that is editable. The followingPage: 4

- Makes the given row editable. In Figure 6, the pencil has been clicked on row 2. This is why the 2nd row is editable.
- Deletes the given row.
- Commits changes.
- Cancels changes.
- Inserts a row. When the "+" image is on the row, then a new row is inserted above the given. When the "+" image is at the end, as pictured in Figure 6, then the new row will be add at the default location in the model.

To change the edit mode and place the TableView in all rows edit mode, then you can use the following code:

```
<sas:TableViewComposite id="tableComposite"
 actionProvider="actionProvider"
 model="tableModel" scope="session">
   <sas:TableView>
       <sas:Edit enabled="true"
        singleRowEditing="false" />
   </sas:TableView>
</sas:TableViewComposite>
```

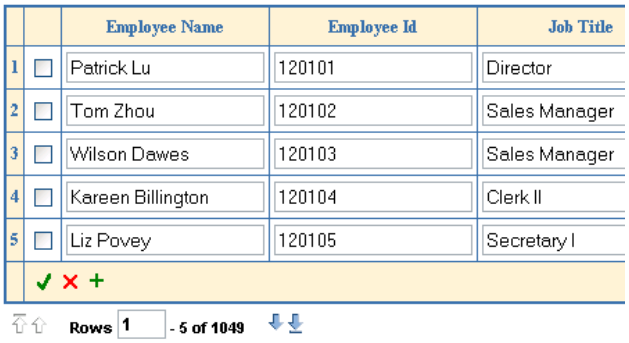


**Figure 7. AllRows Edit Mode**

Note that in this example, you no longer have the individual editing actions available on each row. Instead, all rows are automatically enabled and ready for editing. The end user just goes directly to the cell they want to edit and begins making changes. To commit the changes, use the ✓ image, displayed at the bottom of the table. Similarly, to delete rows, you use the checkbox next to each row number and then select ✗ to delete them in one step. Finally, to add a new row, you select + and it will add a new row in the table.

## DATABEAN FORM

The WebAF DataBean Wizard allows you to connect to a JDBC data source and create a databean that contains a property for specified columns in the data source. This databean allows you to have row-by-row access to the data. This is useful for form-based applications that need to display or edit one row of data at a time.

In Version 3, we provide the ability to easily create form-based Web applications that use a databean generated by the DataBean Wizard with the use of custom tags. In the example below you will see how to do this by using the DataBean and DataBeanForm custom tags.

```
<!-- Create the DataBean custom tag -->
<sasads:DataBean id="dataBean"
 className="Organization"
 dataSource="jdbcConnection" scope="session"
 resultSetConcurrency=
 "<%=java.sql.ResultSet.CONCUR_UPDATABLE%>"/>

<!--Create the DataBeanForm custom tag -->
<sasads:DataBeanForm id="dataBeanForm"
 name="myForm" model="dataBean"
 actionProvider="actionProvider"
 scope="session" render="true">
    <tr>
        <td>Name:</td>
        <td><sas:TextEntry
            name="employee_name"
            size="25" maximumLength="25"/>
        </td>
    </tr>
    <tr>
        <td>ID:</td>
        <td><sas:TextEntry
            name="employee_id"
            size="25" maximumLength="25"/>
        </td>
    </tr>
    <tr>
        <td>Title:</td>
        <td><sas:TextEntry name="job_title"
            size="25" maximumLength="25"/>
        </td>
    </tr>

    <tr>
        <td>Company:</td>
        <td><sas:TextEntry name="company"
            size="25" maximumLength="25"/>
        </td>
    </tr>
    <tr>
        <td>Department:</td>
        <td><sas:TextEntry name="department"
            size="25" maximumLength="25"/>
        </td>
    </tr>
</sasads:DataBeanForm
```

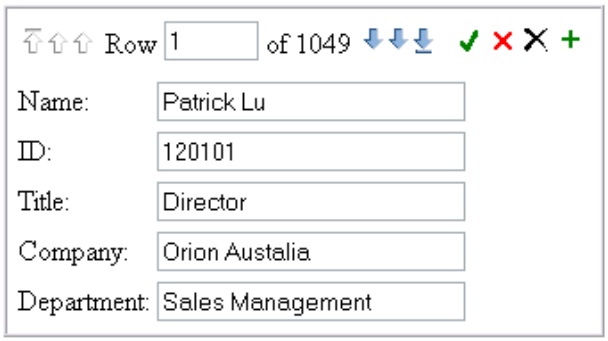


**Figure 8. DataBeanForm**

The DataBean tag is the custom tag for a DataBean Wizard generated databean. The className attribute of the DataBean tag specifies the name of the databean class, which is the fully qualified name that is given to the databean when it is created in the Wizard. In addition to the className, the dataSource must also be specified. The data source for the databean is a java.sql.Connection. In general, the code to create the java.sql.Connection and place it on the session should be in the controller servlet. See the **CONNECTING TO A JDBC DATA SOURCE** section above for more information about creating the JDBC Connection.

The DataBeanForm tag is extended from the FormTag. The model attribute for the DataBeanForm is the databean. In addition to the model, an Action Provider must be set on the DataBeanForm.

Contained within the DataBeanForm are SAS Version 3 Form elements. The name attribute of these form elements should be set to correspond to a property of the databean. This is how the DataBeanForm determines which databean property to access and set on the form element. In the example above sas:TextEntry tags are used, but any of the SAS Version 3 form element tags can be used within the DataBeanForm tag.

The NavigationBar provided with the DataBeanForm allows the end user to scroll through the data row by row. The end user can also "page" forward or backward a specified number of rows, or use the go to field to quickly move to a specific row when the JDBC driver support scrollable result sets. Forward-only result sets may also be used with the DataBeanForm, but row navigation will be limited, with the row paging, back buttons, and go to rows being disabled.

You can provide editing capability in the DataBeanForm when the JDBC driver you are using supports result sets that can be updated. As illustrated in the example above, you need to set the resultSetConcurrency attribute in the DataBean tag to java.sql.ResultSet.CONCUR_UPDATABLE. By default the

databean is read only. When editing is enabled the DataBeanForm's editing buttons are visible and enabled. This allows the end user to update, insert and delete rows.

## CONCLUSION

AppDev Studio gives application developers a simple way to build data-driven Web applications that utilizes JDBC technologies. Using a combination of the new TransformationBeans available in Version 3 – tables, charts, treeviews, listboxes, comboboxes and more – combined with the new JDBC adapters, you can easily build sophisticated data-driven applications using server-side Java.

## ADDITIONAL RESOURCES AVAILABLE

The AppDev Studio Developer's Web site is designed to help you develop and implement enterprise applications that use the power of SAS software to support information delivery and decision making. The AppDev Studio Developer's Web site is continuously updated with new information, including comprehensive tutorials, how-to topics, and technical papers.

A snapshot of the AppDev Studio Developer's Web site is installed to you local Web server when you install AppDev Studio. You can always access the most current version of this site at http://support.sas.com/rnd/appdev/index.htm.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Chad Ferguson
SAS Institute Inc.
Work Phone: (919) 531-6127
Email: Chad.Ferguson@sas.com

Sandra Brey
SAS Institute Inc.
Work Phone: (919) 531-2544
Email: Sandra.Brey@sas.com