Paper 024-29

A Guide to Understanding Web Application Development

Corey Benson, SAS Institute, Inc., Cary, NC
Robert Girardin, SAS Institute, Inc., Cary, NC

ABSTRACT

You have been asked by your manager to port a legacy, fat-client application to the web. There are so many directions you can take in completing this project. It is hard to know where to start and what technologies to use. The main focus of this paper is to introduce you to server-side Java. We will answer questions like:

- What is server-side Java technology?
- What are servlets, JavaServer Pages and custom tags?
- What are the benefits of using server-side Java technology?
- What is the Model-View-Controller(MVC) architecture?
- What is the Struts framework?
- What are the benefits of using webAF for Web application development?
- How do you customize the look of your web application?

These topics will be covered while also highlighting how SAS AppDev Studio™ 3.0 makes it easier for you to build web applications that are lightweight, easy to manage and instantly connect to SAS software.

INTRODUCTION

With many corporations facing increased maintenance costs of older legacy systems, they are looking for ways to streamline those systems to reduce costs while enhancing the systems functionality. This is where the Java™ 2 Enterprise Edition Platform (J2EE) 1.3 can greatly benefit a company by enabling them to integrate their legacy systems with evolving standards to create industrial strength enterprise applications.

The J2EE platform is a collection of technologies that include Java Servlets, JavaServer Pages (JSP), Java Database Connectivity (JDBC), Enterprise JavaBeans (EJBs), Remote Method Invocation (RMI), and more. These technologies give you everything you need to build multi-tiered enterprise systems that are scalable, reliable, adaptable and maintainable.

The J2EE platform is the perfect choice for developing enterprise applications because it offers the following benefits:

- Platform/vendor independence
  The J2EE platform is based on a set of specifications rather than an implementation which allows a company to build and deploy applications based on those specifications. This in turn prevents a company from being tied to any particular vendor/platform and allows them to switch to another vendor/platform with minimal effort.

- Scalability
  The J2EE platform provides all of the necessary building blocks to develop systems that scale from working prototypes to complete 24 x 7, enterprise-wide systems, which are accessible by tens, hundreds, or even thousands of simultaneous clients.

- Flexible security model
  The J2EE platform provides a flexible and unified security model which allows application developers to set the security requirements for resources based on roles. Only users with the appropriate permissions are allowed access to specific resources.

- Development productivity
  J2EE uses an object-oriented, component-based model for application development. This approach enhances development productivity because well-designed generic components can be reused across applications. This means faster development time, better quality and maintainability, and portability of applications.

This paper will focus on how SAS AppDev Studio™ 3.0 will help you build and deploy web application using the J2EE platform technologies.

SAS AppDev Studio™ 3.0 provides a single interface for the development of thin-and power-client business intelligence applications. SAS AppDev Studio supports every major Web standard on both the server and the client side. CIOs can choose SAS as their standard for business intelligence with the knowledge that the analytical power of SAS can be deployed throughout the organization, regardless of the applications development and information distribution architecture.

JAVA SERVLET TECHNOLOGY

Servlets are the Java platform technology of choice for extending and enhancing Web servers. With servlets, the process of writing the server-based Java code that creates interactive applications is simplified. Servlets provide an object-oriented and component-based environment for building web applications, without the performance limitations of CGI programs. Unlike CGI, servlets use a single process for all requests against the server and therefore reduce the risk of overloading the server.

If you are familiar with applet development, then you will find servlet development to be straightforward as well. Unlike applets, though, servlets are not downloaded to the client. Servlets enable you to perform processing on the server instead of on the client. A servlet extends the capabilities of servers that host applications accessed by way of a request-response process. Servlets are typically executed as the result of a client (e.g. Internet Explorer or Netscape) request and produce a response of markup language (e.g. HTML) that is then displayed on the client browser.

Servlets have access to the entire family of Java APIs. This includes the J2SE library, JDBC API, and the library of HTTP-specific calls. In addition to these APIs, servlets can take advantage of the SAS Java Component Library included with SAS AppDev Studio. This is a robust library containing JSP/servlet-based visual components (referred to as TransformationBeans), models for accessing relational data via JDBC, and various utility classes.

For more information on the Java Servlet 2.3 Technology, please visit: http://java.sun.com/products/servlet/index.jsp.

JAVASERVER PAGES (JSP) TECHNOLOGY

The JavaServer Pages technology is an extension of Java servlet technology that is designed to allow rapid development of dynamic web pages that are easily maintained and which leverage your existing business systems. The JSP technology makes it easy to separate the user interface from content generation which enables designers to change the overall page layout without altering the underlying dynamic content.

JSP technology uses XML-like tags, referred to as custom tags, to encapsulate common functionality and logic to generate page content. A JSP is simply an HTML page that contains embedded Java code. If you are familiar with HTML, then you can use custom tags without having to learn the Java language but still have access to the power of running server-side Java.

Custom tags are combined together into a custom tag library that is very portable since they can be embedded in a jar file to be used in multiple web applications. Included with SAS AppDev Studio, is the SAS Custom Tag library which enables JSP developers to develop Web applications that take advantage of TransformationBeans without knowing all of the required Java APIs. In order for a JSP to use a custom tag library, you must include a taglib directive at the top of the page. The following is an example of the directive to include in your JSP to use the SAS Custom Tag library:

```
<%@ taglib uri="http://www.sas.com/taglib/sas" prefix="sas" %>
```

The taglib directive contains two attributes: a uri, whose value is a Uniform Resource Identifier which eventually maps to the Tag Library Descriptor, and a prefix for the tags using this library within the page. The Tag Library Descriptor (TLD) is a file that contains information about the tag library and is used by the servlet container running the Web application. The TLD contains information such as tag and attribute names, required attributes, validation information and information about JSP scripting variables. Here is a simple example of a SAS CheckBox custom tag that will write out the HTML code to the requesting client:

```
<sas:CheckBox id="os_used" text="Windows XP" value="winxp" />
```

When a JSP page containing custom tags is first accessed by a client, it is parsed, validated, translated into a servlet, and finally executed. During the parse phase, the TLD file is used to make sure the tag library has proper syntax. If specified in the TLD, validation classes are used to verify that a particular tag library is being used as intended, such as the correct nesting of tags and the proper use of tag attributes. If validation fails, messages will be sent back to the browser to let the user know what is wrong. Translation involves transforming the JSP page into a Java servlet, which includes converting the tags into Java code and setting up scripting variables. Once a servlet is generated, it is compiled and then executed to produce the response which is sent to the client browser.

Subsequent access of that JSP page will reuse the previously compiled servlet. JSP pages can also be precompiled prior to deploying which prevents the need to perform the parsing, validation, translation and compilation steps. Instead, just the execution of the servlet is necessary which greatly speeds up the access to the JSP page.

Another important aspect of custom tags is the use of scripting variables. Classes that extend the TagExtraInfo class and are added to the TLD file can be used to add scripting variables for use within a JSP. Scripting variables, whether defined in a servlet, scriptlet code, or another custom tag, allow attributes which reside on a particular JSP scope (page, request, session or application) to be accessible within a tag.

Since JSP is transformed into a servlets, they have all of the benefits of servlets. But using the JSP technology also has its own benefits:

- Since a JSP page is compiled dynamically into a servlet when requested, page authors can easily make updates to presentation code and see those changes instantly.
- Developers can offer page authors a specialized custom tag library that can be incorporated into their static HTML templates to generate dynamic content.
- Using the specialized custom tag library, the page author is shielded from the complex application and business logic that is encapsulated within the custom tags.
- Page authors can change and edit the fixed template portions of pages without affecting the application logic. Similarly, developers can make logic changes at the component level without editing the individual pages that use the logic.

In general, JSP allows developers to easily distribute application functionality to a wide range of page authors. These authors do not have to know the Java programming language, so they can concentrate on writing their HTML code while the developers concentrate on creating your objects and application logic. By separating the underlying page logic from its design and display, JSP technology makes it faster and easier than ever to build Web applications.

For more information on the JavaServer Pages Technology, please visit: http://java.sun.com/products/jsp/index.jsp.

WEB APPLICATIONS

A Web application is a collection of servlets, html pages, classes, and other resources that can be bundled and run on multiple containers from multiple vendors. One of the main characteristics of a Web application is its relationship to the ServletContext. Each Web application has one and only one ServletContext. This relationship is controlled by the servlet container and guarantees that Web applications will not clash when storing objects in the ServletContext.

The Java Servlet Technology specifies the following directory structure for a Web application. In this example, the name of the Web application is Sugi29.

Directory	Contents
/Sugi29	The top-level directory or the document root of the application. The document root is normally where all of the JSP pages and static HTML pages are stored.
/ Sugi29/WEB-INF	This directory contains all resources related to configuring your web application. This is where your web application deployment descriptor, web.xml, file is located. Note that the contents of the WEB-INF directory, or any of its subdirectories, cannot be served directly to a client.
/ Sugi29/WEB-INF/lib	This directory contains Java Archive (JAR) files that the web application depends upon.
/ Sugi29/WEB-INF/classes	This directory is where servlet and utility classes are located.
/ Sugi29/images	This directory is specific to ADS and contains all of the image files for the TransformationBeans.
/ Sugi29/scripts	This directory is specific to ADS and contains all of the JavaScript files for the TransformationBeans.
/ Sugi29/styles	This directory is specific to ADS and contains all of the stylesheets for the TransformationBeans.
/ Sugi29/templates	This directory is specific to ADS and contains all of the template files for the composite TransformationBeans.

The Web application directory structure allows for classes to be placed within both the /WEB-INF/classes and /WEB-INF/lib directories. The Web application class loader will load classes from the /classes directory first followed by the JARs in the /lib directory. If you have duplicate classes in both the /classes and /lib directories, the classes in the /classes directory will be used.

A Web application is defined by the contents of the WEB-INF/web.xml file, also referred to as the deployment descriptor. The deployment descriptor contains the following types of configuration and deployment information for a web application:

- ServletContext Init Parameters
- Session Configuration
- Servlet / JSP Definitions
- Servlet / JSP Mappings
- MIME-type mappings
- Error pages
- Filters
- Event handlers
- Security information

If you want to distribute a Web application, you package it in a Web application archive (WAR), which is similar to a JAR that is used to package Java class libraries. This WAR file can then be deployed to different J2EE-compliant servlet containers without modification. For more information about deploying your Web application, please visit: http://support.sas.com/rnd/appdev/webAF/server/DeployingWebApps.htm.

MODEL-VIEW-CONTROLLER ARCHITECTURE

When developing a web application using JSP technology, there are two approaches that are typically used: JSP Model 1 and JSP Model 2 architectures. The main difference between these two architectures is where the majority of the controlling and business logic resides.

In the JSP Model 1 architecture (see Figure 1 below), the JSP not only contains the HTML necessary to produce the user interface but is also responsible for processing the user's request in order perform the necessary business logic to control the dynamic content generation. The resulting JSP ends up containing a significant amount of embedded Java code, referred to as scriptlet code. While this might not seem like a big deal, the JSP quickly becomes unwieldy and difficult to maintain and modify. The Model 1 architecture makes it impossible to maintain the role separation when developing a Web application. The page author now needs to know and understand Java in order to make even the simplest change to the user interface in the JSP. These types of Web applications are not flexible and difficult to reuse.

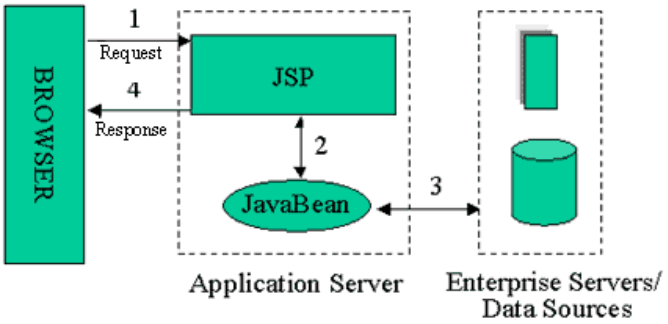


Figure 1: JSP Model 1 architecture

In the JSP Model 2 architecture (see Figure 2 below), both the Servlet and JSP technologies are leveraged to develop a Web application. A servlet is used to control the flow of the application, processing the user's request, performing the business logic in addition to creating and modifying any objects or beans that are needed by the JSP. A JSP is strictly used as a presentation layer which consumes the previously created objects or beans from the servlet via custom tags. The JSP contains no control or business logic. Using this architecture allows a clear separation of roles between the Java developer and page author. The Java developer focuses on the servlet and all associated JavaBeans that contain the business logic. The page author concentrates on creating the user interface using HTML and custom tags in the JSP.

Using the JSP Model 1 architecture may work nicely for simple, small scale Web applications. But as your Web application becomes more complex and larger scale, you will quickly see all the benefits that come with using the JSP Model 2 architecture.

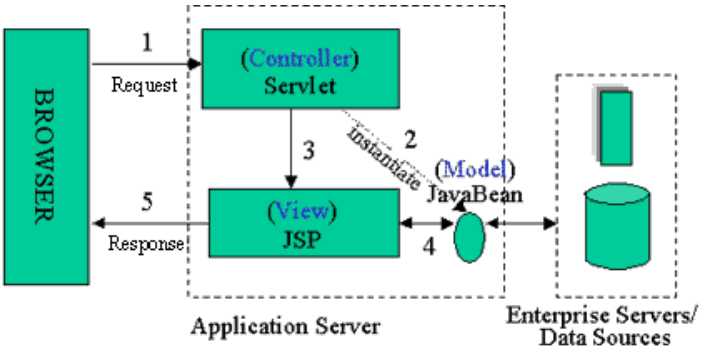


Figure 2: JSP Model 2 architecture

As we have seen with the JSP Model 2 architecture, also know as the Model-View-Controller (MVC) architecture, the JSP technology complements the Servlet technology allowing you to create a maintainable, flexible and robust web application.

As show in Figure 2, the MVC architecture consists of three parts: Model, View, and Controller.

- The Model, typically represented by one or more JavaBeans, is the data and business logic behind the application.
- The View, typically a JSP, is used to present the information represented by the Model.
- The Controller, typically a servlet, processes user interaction and controls the flow of the web application.

All requests made by the user from the client browser are funneled to the controller servlet. The controller servlet processes the user's requests and modifies the model as necessary. Based on the input from the user, the controller decides which view it will pass control to. In Figure 2 there is only one view shown, but in reality there usually are multiple views. The view to which the control is forwarded then consumes the model and returns the results to the client browser.

STRUTS

Struts is a Web application development framework that is based on the MVC architecture and is designed for the J2EE platform. Struts has practically become the de facto standard for building Web applications. Struts consists of a set of tools and components based on standard technologies like Java Servlets, JavaServer Pages, JavaBeans, and XML that speed up the development process of building Web applications and make developers more productive. Not only is Struts a framework for developing Web applications, it also contains an extensive tag library and utility classes that work independently of the framework.

Using a framework like Struts allows you, the developer, to focus in on developing just the business logic and presentation layers instead of the entire Web application. In addition to narrowing down what you have to develop, you will be starting with a robust set of tools and reusable components that will help you develop your business logic and presentation layer more quickly and efficiently.

The main classes that make up the framework are:

ActionServlet and ActionMapping classes

In Struts, the ActionServlet class is the controller part of the MVC architecture and is the core of the framework. This servlet is configured by defining a set of ActionMappings which are specified via the struts-config.xml file. The struts-config.xml file contains the entire logical flow of the application. Each ActionMapping defines a path that matches possible incoming URI requests and usually specifies the fully qualified class name of an Action class.

Action class
The Action classes are the link between the Struts framework and your business logic. Most of the business logic can be represented using JavaBeans. An Action can call the properties of a JavaBean without knowing how it actually works. This encapsulates the business logic, so that the Action can focus on interpreting the outcome, error handling and ultimately dispatching control to the appropriate View component to create the response.

ActionForm class
The ActionForm class makes it easy to store and validate the data for your application's input forms. The ActionForm bean is automatically saved in one of the standard, shared context collections, so that it can be used by other objects, like an Action object or another JSP.

The form bean can be used by a JSP to collect data from the user, by an Action object to validate the user-entered data, and then by the JSP again to re-populate the form fields. In the case of validation errors, Struts has a shared mechanism for raising and displaying error messages.

Struts is part of the Apache Jakarta Project which is sponsored by the Apache Software Foundation. The official Struts home page is at http://jakarta.apache.org/struts.

BENEFITS OF WEBAF FOR WEB APPLICATION DEVELOPMENT

Web applications use a diverse collection of technologies stitched together with various kinds of linkages and relationships, often in a large number of files. Conventional Web application development usually involves using a variety of command-line tools in various packages and developing batch files to try to add a degree of consistency to the process. webAF offers a refreshing alternative. It helps you manage this complexity with a single consistent, integrated tool, thus reducing or eliminating much of the drudgery of development, testing and deployment. Best of all, webAF provides tight integration with other SAS technologies, bringing all the power of SAS to life in your Web applications.

FEATURES

1. Integrated Development Environment (IDE)
   webAF presents you with a familiar and intuitive GUI development center, with the features of program development extended to include web development.

2. Project wizard with content templates
   The project wizard jump-starts your Web development by letting you choose the features that you want to include and initial content of your project. Starter files are then generated for you, including a web.xml file. An initial JSP file includes declarations of tag libraries supporting the features you selected. Servlets have boilerplate code for the standard methods. You can also "adopt" an existing Web application in the new project wizard.

3. Systematic locating of projects and directories
   When you start a new Web application project, webAF automatically supplies the necessary directory structure, and populates the WEB-INF\Lib directory with the JAR files needed. You can specify the base directory for your Web application to be external to the project directory, too.

4. Automatic updating of build file and web.xml
   As you add or remove servlets and/or Java source files from your project, webAF automatically updates the build.xml file to include the file and classpath information and updates the deployment descriptor (web.xml) to include the necessary servlet declaration and mapping.

5. SAS Java Component Library
   A robust library containing JSP/servlet-based visual components (TransformationBeans), Swing-based visual components, models for accessing relational data via JDBC, and various utility classes.

6. SAS Custom Tag Library
   Custom tags that enable a page author to take advantage of TransformationBean without knowing all of the required Java APIs.

7. Debugger with support for remote and JSP debugging
   The integrated debugger extends familiar use of breakpoints to Web applications. You can even set breakpoints in the compiled JavaServer Pages source tab of the editor (available for supporting Web

servers, including the Tomcat server that is integrated in webAF).

8. Auto-serve of current project
   As you create a new Web application project, the webAF Web server will automatically serve your application, so you can just click the Execute in browser toolbar button to test it. After you make a change and rebuild the project, clicking this button again will automatically stop and restart the Web server, so your changes will be reflected in the served content.

9. Deployment support and generation of WAR file
   A menu command consolidates your entire Web application into a single .war file that can be deployed to compliant web servers. During construction the deployment descriptor (web.xml) file is validated automatically and a user-supplied or generated Manifest.mf file is included in the .war file.

10. Ant-based customizable builds
    Build scripts are generated by webAF using standard Ant XML files, and are automatically modified as you add new files to your project. Build files can also be customized with user-defined targets.

11. Start and stop Web servers
    Convenient menu commands are provided to start and stop the current Web server.

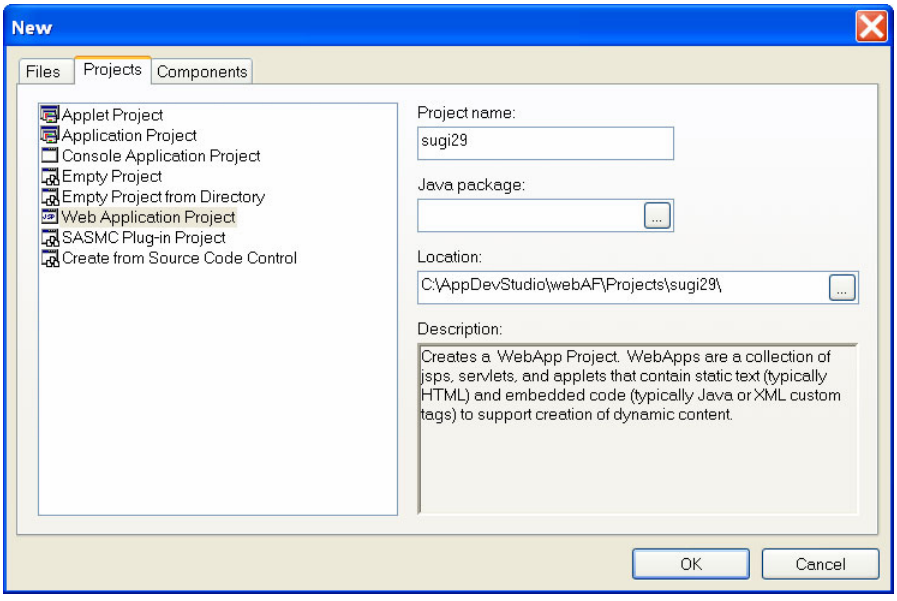
12. Integrated Struts and JSTL support
    The popular open-source, industry standard Jakarta Struts framework can serve as foundation for your Web application. Select an option in the project wizard, and webAF will automatically include the required XML files and tag libraries. The JavaServer Pages Standard Tag Library (JSTL) is also integrated, to reduce or eliminate the need for Java code snippets in JSP files.

USING WEBAF TO BUILD A WEB APPLICATION DEVELOPMENT

Creating a new Web application project is easy using the New Project Wizard in webAF. The wizard will walk you through all of the necessary steps in order to produce a web application that is customized for your needs. The following steps show you how to create a new Web application that will display a data from a JDBC source in an HTML table.

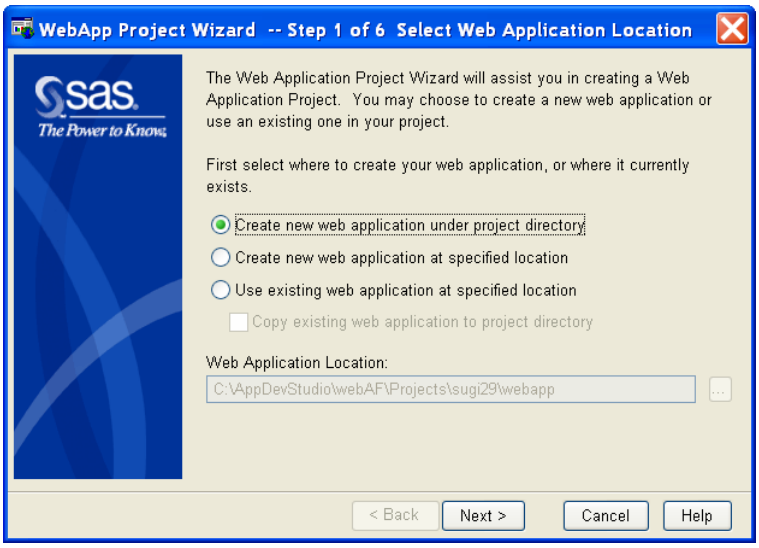
NOTE: Prior to building this example, please make sure that you are using the latest webAF example templates by visiting: http://support.sas.com/rnd/appdev/webAF/server/template_updates.htm.

1. Select **File→New** and select the Projects tab if it is not already visible.

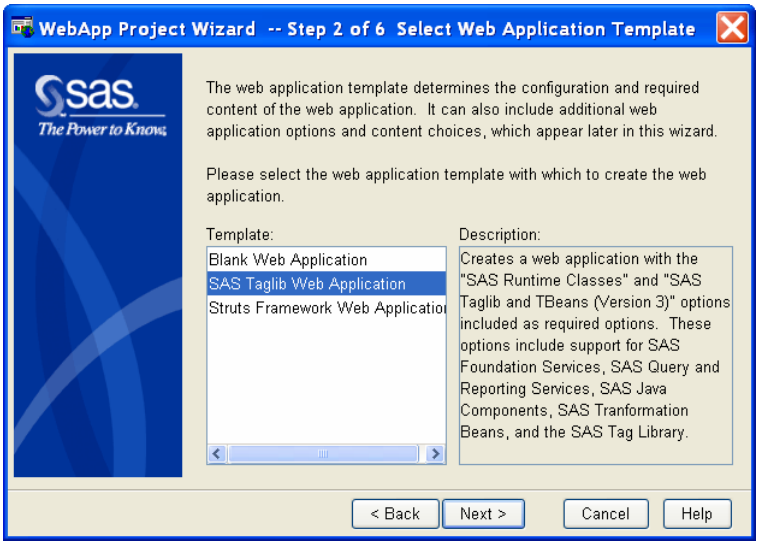


a. In the list of project types at the left, select ***Web Application Project***.
b. In the ***Project name*** text entry field, enter *Sugi29*.
c. Click ***OK*** to start the WebApp Project Wizard.

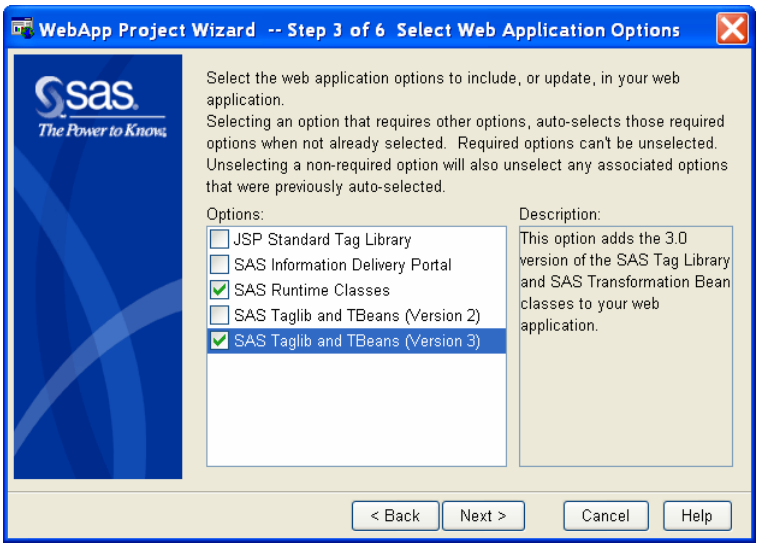
2. In next step of the wizard, you choose where your Web application will be located. Usually you will want to accept the default location under your project directory from the previous step. Click ***Next***.



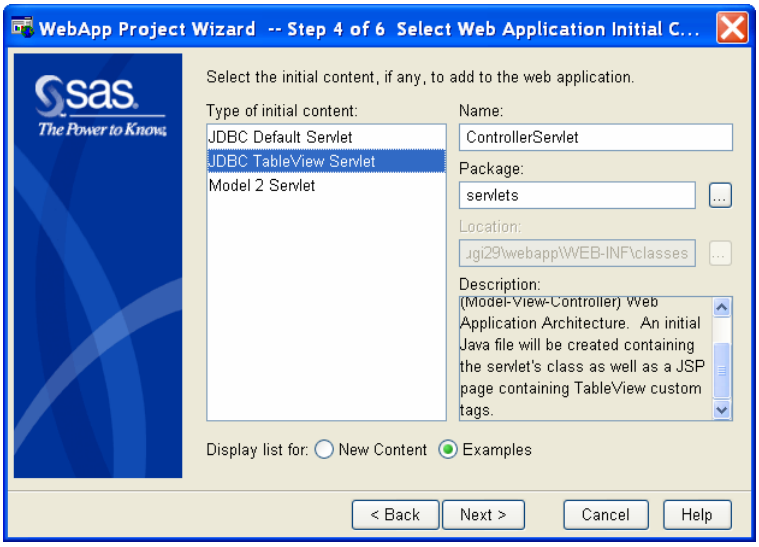
3. In next step of the wizard, you choose a template for your web application. The template determines what "boilerplate" files and options will be required and automatically created for your project. For this example, the ***SAS Taglib Web Application*** template is going to be used. Click ***Next***.



4. In next step of the wizard, you choose the Web application options. The options that are available depend on the template you chose in previous step of the wizard. For this example, we are not going to choose any other options than the defaults. Click ***Next***.

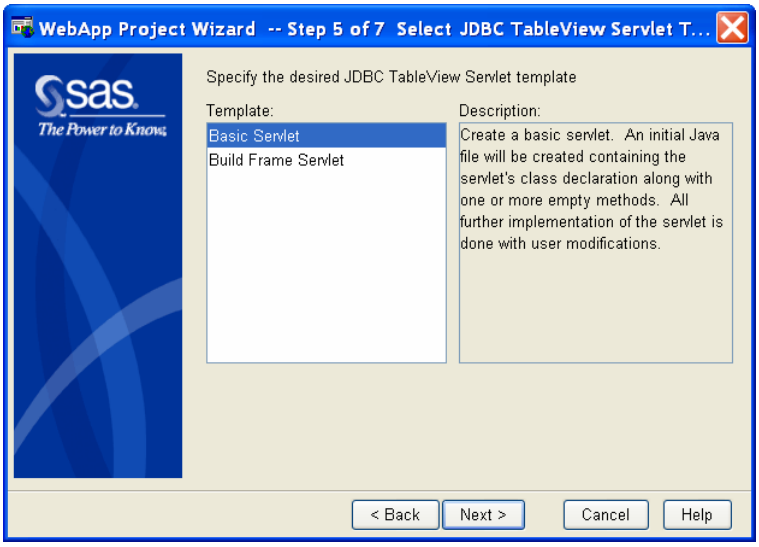


5. In next step of the wizard, you choose the initial content for your Web application. Note: Depending on options that you selected in previous steps, a page of additional project options might be displayed before this page. webAF ships with several examples that are all based on the MVC architecture as described in a previous section.

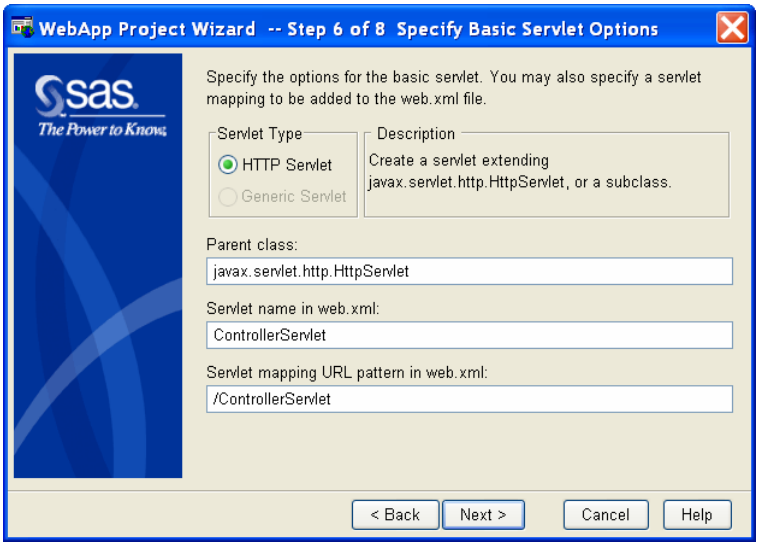


   a. Select ***Examples*** in the ***Display list for*** radio box.
   b. In the ***Type of initial content*** list box, choose ***JDBC TableView Servlet***.
   c. Click ***Next***.

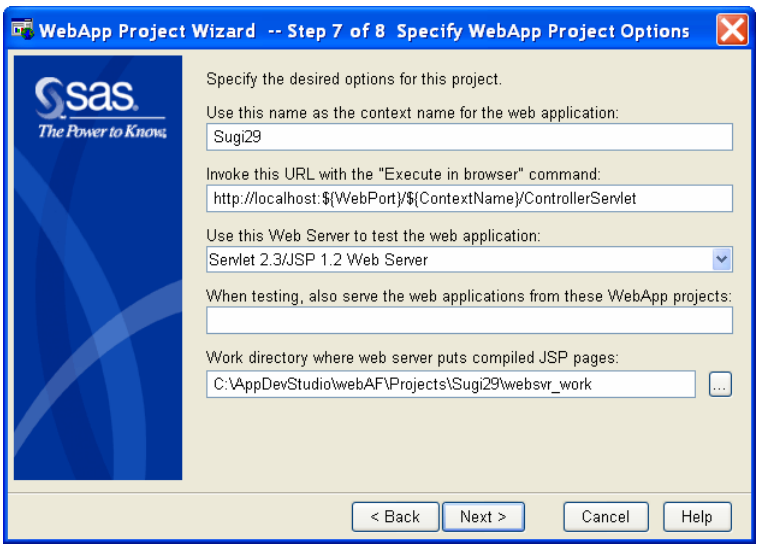
6. In next step of the wizard, you choose the template for the initial file. What you chose in the previous wizard step will determine what is presented in this step. For this example, we chose the initial content of ***JDBC TableView Servlet*** so you will be prompted with the type of servlet to create. Keep the default of **Basic Servlet**. Click ***Next***.



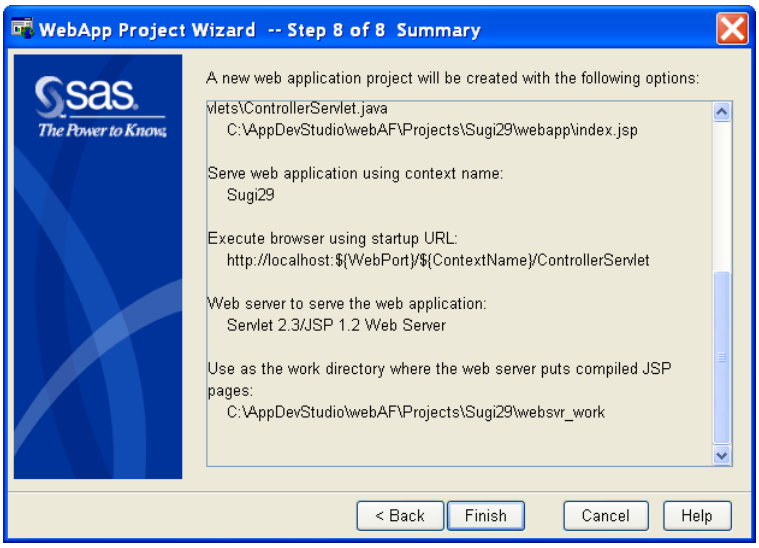
7. In next step of the wizard, you choose the Basic Servlet options. Keep the default values and click ***Next***.



8. In next step of the wizard, you choose the Web application project options. Keep the default values and click ***Next***.

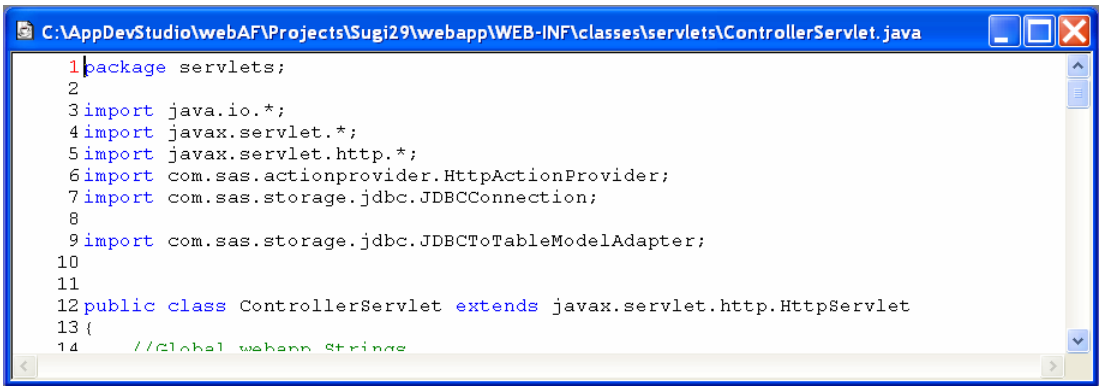


9. The final step of the wizard displays a summary of selections made in the wizard. Click Finish to create your new Web application.

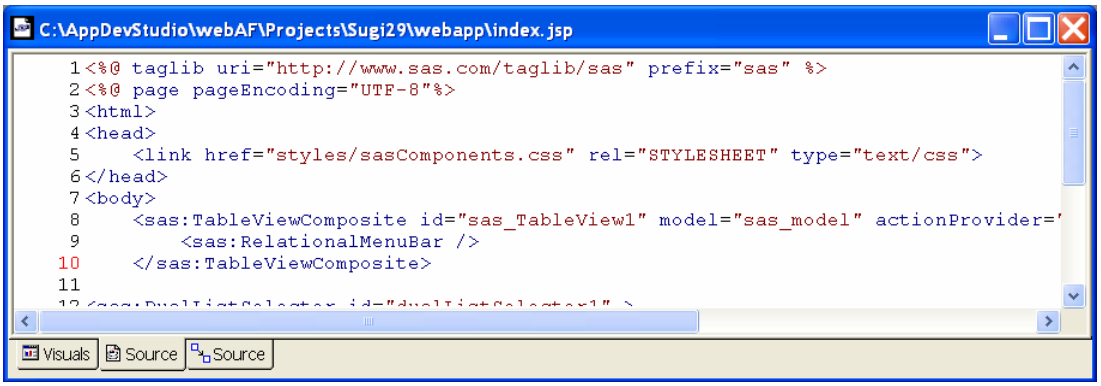


After the WebApp Project Wizard closes, webAF will create the directories and content files based on the selections you made. This may take a few seconds. Then webAF will open the initial file(s) for editing. For the selections made in the wizard above, the following files will have been created for you:

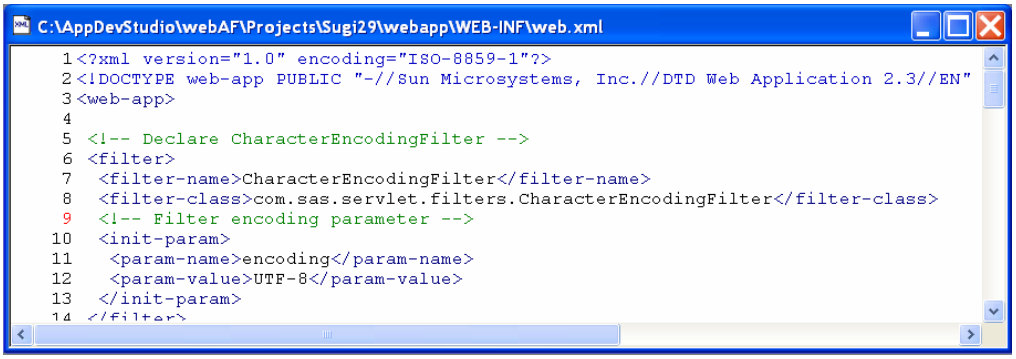
- ControllerServlet.java: Contains the code to connect to the JDBC Data source along with the application control code to process any user interaction and forward to the index.jsp page to display the results.



- index.jsp: Containing the custom tag for the TableViewComposite



- web.xml: deployment descriptor containing all of the necessary servlet mappings already defined.



Now that you have created a default Web application for viewing a JDBC data source in a table, you need to perform a few final steps to get a working example.

1. On line 90 of the ControllerServlet, you will need to specify which JDBC data source to use where it has ENTER_QUERY_STRING_HERE. For this example, replace that line with:

```
String jdbcQuery = "select * from sashelp.class";
```

2. Select ***Build→Compile File*** to compile the servlet.

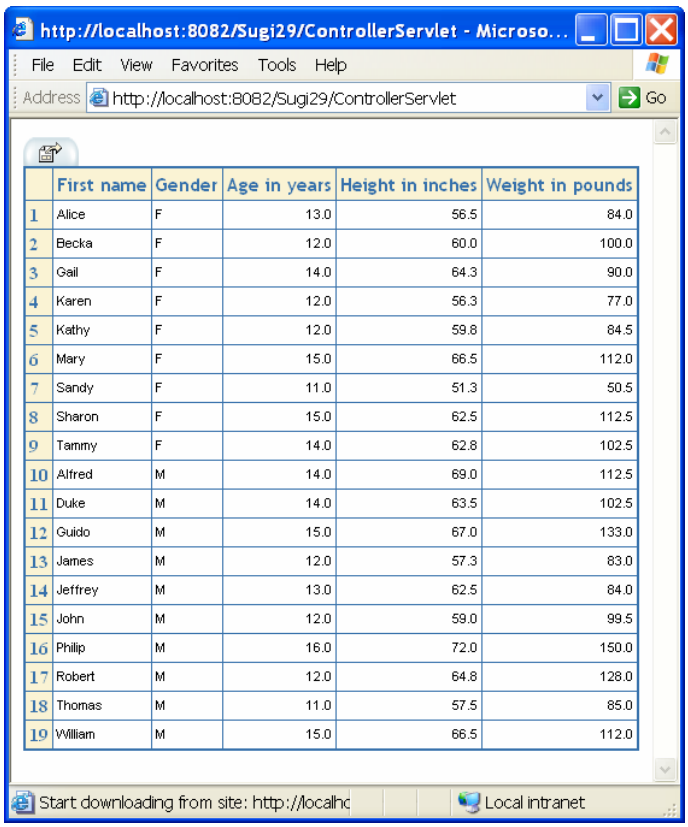
3. Start the IOM spawner by selecting ***Start→All Programs→SAS AppDev Studio→Services→SAS***

V9.1→Start SAS V9.1 IOM Spawner. The IOM spawner is necessary in order to use the IOM JDBC driver which is included with SAS 9.1.

4. Start the Java Web Server by selecting ***Tools→Services→Start Java Web Server***. This will start up the bundled Tomcat along with serving up the Sugi29 project.

5. Select ***Build→Execute*** to bring up a browser to execute Sugi29 web application in your default browser.

You should see results similar to the following:



	First name	Gender	Age in years	Height in inches	Weight in pounds
1	Alice	F	13.0	56.5	84.0
2	Becka	F	12.0	60.0	100.0
3	Gail	F	14.0	64.3	90.0
4	Karen	F	12.0	56.3	77.0
5	Kathy	F	12.0	59.8	84.5
6	Mary	F	15.0	66.5	112.0
7	Sandy	F	11.0	51.3	50.5
8	Sharon	F	15.0	62.5	112.5
9	Tammy	F	14.0	62.8	102.5
10	Alfred	M	14.0	69.0	112.5
11	Duke	M	14.0	63.5	102.5
12	Guido	M	15.0	67.0	133.0
13	James	M	12.0	57.3	83.0
14	Jeffrey	M	13.0	62.5	84.0
15	John	M	12.0	59.0	99.5
16	Philip	M	16.0	72.0	150.0
17	Robert	M	12.0	64.8	128.0
18	Thomas	M	11.0	57.5	85.0
19	William	M	15.0	66.5	112.0

CUSTOMIZING THE LOOK OF YOUR WEB APPLICATION

After creating a Web application in webAF, you will probably want to customize the look to match your company's standards. The TransformationBeans and custom tags included with the SAS Java Component Library give you the flexibility to customize the way you want them to look using Cascading Style Sheets, composites and templates.

CHANGING STYLES

Many of the TransformationBeans included in the SAS Java Component Library enable you to customize the HTML markup using Cascading Style Sheets (CSS). CSS is a technology standard endorsed by the World Wide Web Consortium (W3C) which enables rich layout of HTML elements. You can use CSS to control the colors, fonts, as well as the size and spacing of the HTML elements generated by TransformationBeans. The default styles for all of the TransformationBeans are contained in the sasComponents.css stylesheet located in your <webApp>/styles directory. To use the default TransformationBean styles, the following line should be added to the <HEAD> section of your JSP:

```
<link rel="stylesheet" type="text/css" href="styles/sasComponents.css">
```

TransformationBeans are divided into two categories with respect to supporting styles: simple and complex. Both simple and complex TransformationBeans use the com.sas.servlet.tbeans.StyleInfo class to specify the classid and/or the inline style to use.

When a TransformationBean only has a single area to apply a style, then it is referred to as a "simple" style component. Examples would be the TextEntry or PushButton TransformationBeans. These components implement the com.sas.servlet.tbeans.ComplexStyleInterface. This interface contains methods whish allow you to get and set

the StyleInfo object used by that component. For example, the following code is used to override the style class used by a TextEntry TransformationBean:

```
com.sas.servlet.tbean.StyleInfo si = myTextEntry.getStyleInfo();
si.setClassid( "myClassId" );
```

When a TransformationBean has multiple areas that a style can be applied to, then it is referred to as a "complex" style component. Examples would be the ComboBoxView or TableView TransformationBeans. These components implement the com.sas.servlet.tbeans.ComplexStyleInterface. This interface has a single method, getStyleMap(), which returns a Map of named StyleInfo objects that are used by that component. Each complex style component has an associated StyleKeysInterface which contains the valid style keys. For example, the com.sas.servlet.tbeans.form.html.ComboBoxView class has its style keys defined in com.sas.servlet.tbeans.form.html.ComboBoxViewStyleKeysInterface. To override the default you create a new StyleInfo object and use the put(String key) to set the new StyleInfo object on the map. For example, the following code is used to change the style class used for the highlighting the selected row in a ComboBoxView:

```
// Get the style map from the ComboBoxView
java.util.Map styleMap = myComboBoxView.getStyleMap();

// Create a new StyleInfo object containing the desired style settings
com.sas.servlet.tbean.StyleInfo si = new StyleInfo( "myClassid" );

// Set the new StyleInfo object on the map using the desired style key.
styleMap.put( ComboBoxViewStyleKeysInterface.COMBOBOXVIEW_LIST_HIGHLIGHT, si );
```

Changing the styleMap is required only if altering the default style name is needed. The default Cascading Style Sheet (CSS) can be modified to change the text color, background color, font, etc.

For more information about TransformationBean styles, please visit Component Style Reference page: http://support.sas.com/rnd/appdev/webAF/server/ComponentStyles/ComponentStyleIndex.htm.

COMPOSITES AND TEMPLATES

When a component contains more than one sub-component, this is referred to as a composite TransformationBean. All composite TransformationBeans implement the com.sas.servlet.tbeans.CompositeComponentInterface which has the necessary method for manipulating the sub-components within the composite. The following TransformationBeans are composite components:

- DualListSelector
- DualTreeSelector
- TreeListSelector
- TableViewComposite

The layout of the sub-components of a composite component is controlled by an external template file. Each composite component listed above has an associated default template file that is located in your <webApp>/templates directory. The layout of the composite can be customized to meet your needs. Maybe you just need to tweak the default template slightly to shift around the sub-components. Or maybe you need to add new components and completely overhaul the template to fit your requirements. Both of these can be done by either modifying the associated default template for that Web application or create a new template and applying that template to the component using the setTemplateFileName() or setTemplate() methods.

Within a template, there are named substitution parameters which are used to position the sub components of the composite within the template. The substitution parameters begin with a % and end at the next non-Java identifier. Each composite component has an associated KeysInterface which contains the valid sub-component keys used in the template. For example, the com.sas.servlet.tbeans.dualselector.html.DualListSelector class has its sub-component keys defined in com.sas.servlet.tbeans.dualselector.html.DualListSelectorKeysInterface.

To see how easy it is to change the layout of a composite, the following example will alter the layout of a DualListSelector by moving the up/down arrows from the bottom of the target listbox (Figure 3) to the right of the listbox (Figure 4).

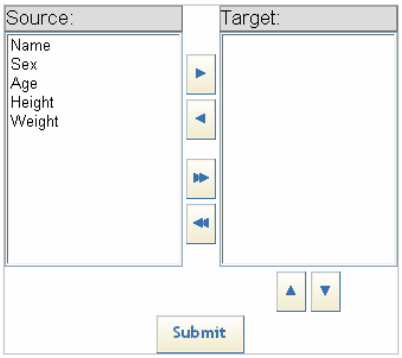


Figure 3: Default DualListSelector

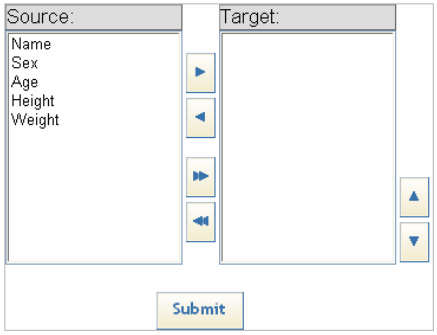


Figure 4: Altered DualListSelector

The default template for the DualListSelector would need to be modified as shown below to achieve this look:

```
<table border="0" cellspacing="0" cellpadding="0" class="%DUALSELECTOR_CONTAINER">
  <tr>
    <td class="%DUALSELECTOR_SOURCE_LABEL_AREA">%DUALSELECTOR_SOURCE_LABEL</td>
    <td> </td>
    <td class="%DUALSELECTOR_TARGET_LABEL_AREA">%DUALSELECTOR_TARGET_LABEL</td>
    <td> </td>
  </tr>
  <tr>
    <td>%DUALSELECTOR_SOURCE</td>
    <td valign="center" align="center" nowrap="nowrap">
      <table>
        <tr>
          <td>%DUALSELECTOR_RIGHT_BUTTON</td>
        </tr>
        <tr>
          <td>%DUALSELECTOR_LEFT_BUTTON</td>
        </tr>
        <tr>
              <td height="10px"></td>
        </tr>
        <tr>
          <td>%DUALSELECTOR_RIGHT_ALL_BUTTON</td>
        </tr>
        <tr>
          <td>%DUALSELECTOR_LEFT_ALL_BUTTON</td>
        </tr>
      </table>
    </td>
    <td>%DUALSELECTOR_TARGET</td>
    <td valign="bottom">
       <table>
          <tr> <td>%DUALSELECTOR_UP_BUTTON</td> </tr>
          <tr> <td>%DUALSELECTOR_DOWN_BUTTON</td> </tr>
       </table>
    </td>
  </tr>
  <tr>
    <td colspan="3"> </td>
  </tr>
  <tr>
    <td colspan="3" align="center">%DUALSELECTOR_SUBMIT_BUTTON</td>
  </tr>
</table>
```

By externalizing the layout of a composite component into a template file, you have the flexibility to alter the user interface to fit your needs without subclassing. This allows the component to retain the same functionality even though the position of the composite's sub-components has changed.

In addition to altering the template to change the UI, you can write Java code to dynamically alter the UI to fit your needs. As mentioned above, the template contains substitution parameters. These parameters are substituted with the values from a template dictionary which is populated in the following manner:

- The components map returned by the getVisibleComponents() method.
- The styles map returned by the getStylesMap() method
- User defined template dictionary which is set by the component via setTemplateDictionary() method.

In order to remove a certain component from the composite TransformationBean, you would do one of the following:

- Remove the key from the template; or
- Get the component from the components map and set visible property to false on that component; or
- Use the removeComponent(<COMPONENT_KEY>) method on the composite class.

The composite class allows you to add components to that class. Using the setComponent method, any transformation bean can be added. Once added to the class, the template must be modified to include that component. Here is an example showing how a TextEntry TransformationBean is added inside the DualListSelector.

```
DualListSelector duallist = new DualListSelector();
TextEntry myTextEntry = new TextEntry();
duallist.setComponent("CUSTOM_KEY", myTextEntry);
duallist.setTemplateFileName("myNewDualListSelectorTemplate.html");
```

The new component was added with a COMPONET_KEY of "CUSTOM_KEY", which will also need to be added to the template in order for that component to show up.

COMPONENT PROPERTY MANAGER

All TransformationBeans that use external resources like templates, images and scripts retrieve these resources based on a corresponding location: templateLocation, imageLocation and scriptLocation. These locations can either be changed on a per instance basis or globally for the entire Web application using the ComponentPropertyManager (CPM) class. The ComponentPropertyManager is used to maintain global common properties across TransformationBeans.

Only one ComponentPropertyManager is created per session per web application. The same ComponentPropertyManager is used throughout that session. A TransformationBean will use the session based ComponentPropertyManager to retrieve the default resource locations unless these locations have been overridden on a per instance basis.

The CPM can be modified to use locations specific to your Web application by using the following code:

```
// Retrieve the CPM for this session using the static getInstance method.
ComponentPropertyManager cpm = ComponentPropertyManager.getInstance(request);

//  Set the locations
cpm.setImageLocation( "<yourImagesDirectory>");
cpm.setJavaScriptLocation("<youScriptsDirectory>");
cpm.setTemplateLocation("<yourTemplatesDirectory>");
```

When initializing a new CPM, the default property values are read in from a properties file located here in the web applcition: <webApp>/WEB-INF/sas_ComponentProperties.config. The configuration file can be modified to change the defaults for those properties. The change will be applied for the entire web application. Each component used within the web application will look for the image location, script location and template location set on the component itself; and if nothing is set on the component, the default values specified on the property file will be used.

CONCLUSION

SAS AppDev Studio ™ 3.0 (ADS) is the perfect choice to help you take advantage of all that the J2EE platform has to offer. SAS AppDev Studio is a product bundle that provides support for the development of thin-client and "power-client" business intelligence applications. It includes the SAS Java Component Library which is a rich set of Java components that provide such applications with access to the full power of SAS software.

ADDITIONAL RESOURCES

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. We encourage you to access external site often site to get the latest updates: http://support.sas.com/rnd/appdev/index.htm.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the authors at:

Corey D. Benson
SAS Institute Inc.
Work Phone: (919) 531-5712
Email: Corey.Benson@sas.com

Robert Girardin
SAS Institute Inc.
Work Phone: (919) 531-2189
Email: Robert.Girardin@sas.com