Paper 33-28

Using AppDev Studio™ and Integration Technologies for an Easy and Seamless Interface Between Java and Server-Side SAS®

Barry R. Cohen, Planning Data Systems, Inc.

ABSTRACT

The application landscape has changed substantially for SAS application developers. Today's landscape uses a distributed-object, component-based architecture that integrates SAS server capabilities into enterprise-wide applications that are n-tiered, Web-enabled, and thin-client. SAS has developed new products to help you build SAS applications in this landscape. These include the AppDev Studio product suite including especially webAF within it, and Integration Technologies. You will likely want to leverage a few key features of these products as you build Web-enabled SAS applications. From AppDev Studio, these features are SAS-aware Java classes for Java programs, SAS Custom Tags for JavaServer Pages, and SAS Components for applets. From Integration Technologies, the feature is the Application Facility. These product features allow an easy and seamless interface between your client- or server-side Java applications and your server-side SAS programs and data. This paper provides an overview of today's SAS application landscape, and discusses how to use these key SAS product features to develop applications in this landscape.

APPLICATION LANDSCAPE TODAY

The landscape has changed for application development in SAS environments. Previously, we used SAS software to build both the client and server sides of our applications, typically using SAS/AF for the user interface (UI). As an alternate, some of us previously used a non-SAS product for the client-side, (e.g. Visual Basic) to interface with server-side SAS. However, the non-SAS client approach did have significant problems regarding direct communication between the client and server applications, and was probably not as prevalent. Today's landscape uses a distributed object, component-based architecture that integrates SAS server capabilities into enterprise-wide applications that are n-tiered and are often Web-enabled and thin-client. This basically means:

- The business logic software, the business data, and the supporting software involved in the application are distributed across multiple hardware platforms (and operating systems). And multiple application software vendors are involved overall.
- The software pieces of the multiple vendors are seen as components that talk to each other, and together they comprise the full application.
- The components are objects, in the Object Oriented Programming sense, and they talk to each other across the software vendor and platform boundaries according to certain object model standards that have been established.
- There is a separation of functions within the application by hardware platform and software platform.

Figure 1 depicts the n-tiered architecture. The tiers, or layers as they are sometime called, typically are:

- Client Tier
  - In a web-enabled application, a web browser is used for user selections and presentation of results. This is accomplished through web pages served from a web server and displayed in the client browser.
  - An alternate approach is to run a Java application in an application window. In this case, the Java application can communicate directly with server-side SAS without use of a web server.
- Middle Tier
  - Web server
    - Receives requests from client browser for services.
    - Sends results, as markup language, to client browser.
  - Application server (also referred to as Java server)
    - Receives requests from the web server for Java servlets and JavaServer pages.
    - Executes these Java programs which in turn formulate and make data and compute service requests of the SAS server layer.
    - Provides some of the business logic.
    - Builds web pages with SAS results as content, and sends them to the web server for final send to client browser.
- Back Tier
  - SAS server layer for data and programs - provides business logic and data.
  - Database server layer - provides additional enterprise business data.

In this new architecture, one key difference from the past for SAS application developers is that SAS software and data reside only on the server side. Other vendor software is used on the client side, and on the other servers involved. Specifically:

- The client side is thin, using only browser software that processes (displays) markup language (HTML). These web pages may have Java programs embedded in them as applets.
- The web server uses non-SAS software. Several vendors provide this software. From the SAS application developer's perspective, it is largely a black box of pre-programmed software that is used but not developed.
- The application server, or "Java server" as it is sometimes called, uses the Java language. This application sits between the web server and your SAS-based business logic and data on the SAS server. Unlike the web server application, this application does typically involve some code development as part of the overall application.

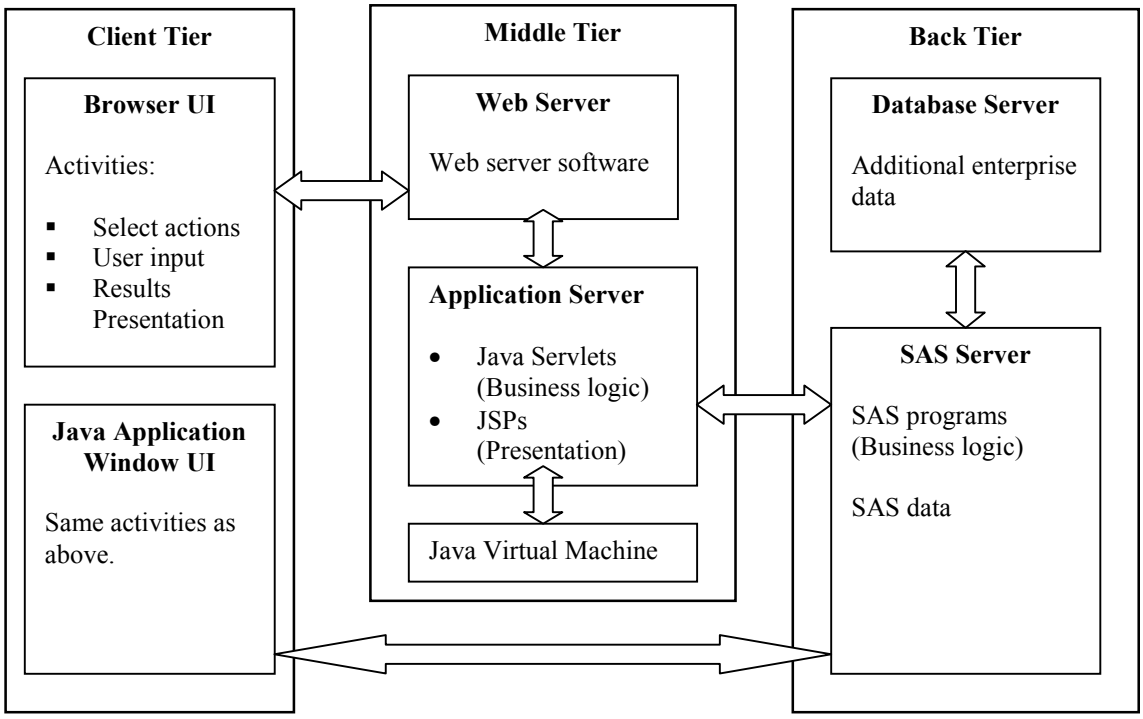

Figure 1: Application Landscape Today

The bottom line is that today, SAS application developers need to know new things. Yesterday, you could build a good client-server SAS application using Base SAS and SAS/AF, with SAS/CONNECT to handle the cross-platform connection. Today, you need to know the Base SAS, Java, and HTML languages, something about using a web server, and either SAS/CONNECT or SAS Integration Technologies to handle the cross-platform and cross-vendor object-to-object communication. And in this set up, Java and HTML, instead of SAS, are used to:

- Create the application's user interface.
- Make requests, through the web-enabled UI, for SAS compute and data services.
- Receive SAS program results, build web pages with these results as content, and pass these pages to the web server for browser presentation.

Java is new to most SAS application developers and we thus face a daunting task in trying to use Java (and HTML) in these roles within our applications. However, SAS has provided some new tools to help us. These tools are available through these new SAS products: the AppDev Studio suite, and Integration Technologies. The specific tools are:

- SAS-aware Java classes, in AppDev Studio.
- SAS Custom Tags for JavaServer Pages development, in AppDev Studio.
- SAS Components for Java applet development, in AppDev Studio.
- The Application Facility, in Integration Technologies.

In general, these tools help you to write Java programs that: (1) establish a connection to server-side SAS, (2) request SAS data and compute services, (3) receive SAS results and builds web pages with SAS results as dynamic content, and (4) build web pages that provide the application's UI in general aside from the interactions with server-side SAS.

These tools provide significant help to experienced Java programmers in this environment, because they generate much of the code that is needed, and because they handle all of the technical details of object-to-object communication between Java and SAS services. And their value to inexperienced Java programmers in this environment is even greater. In this paper, I will introduce these new tools generally, provide some specific information about what these tools do, and provide a few code examples of how they work.

APPDEV STUDIO & webAF - BACKGROUND

SAS AppDev Studio is actually a suite of SAS products used to develop applications using Java technology. It is probably fair to say that AppDev Studio was developed by SAS directly in response to today's application landscape. SAS knows that its software and data will be server-based in this landscape and will have to integrate into it from this perspective. And SAS knows that most people will access SAS servers from a web-enabled client. Thus, SAS knows that Java programs will be fielding browser requests and in turn making requests of SAS. So application developers in SAS environments will need to write Java programs

that access server-side SAS. AppDev Studio has been developed by SAS to address this need. It is used to build Java applications that run on either the client or the server side. AppDev Studio automates many steps in the Java development process. And it especially provides help for the cross-platform communication between Java and SAS.

webAF is one of the core components in the AppDev Studio suite, and it is the primary Java development product in the suite. Java technology can deliver its functionality in today's application landscape by running either server-side, as servlets or JavaServer Pages, or client-side, as applications or applets. And webAF provides all you need to create servlets, JavaServer Pages, applications, and applets. webAF is an integrated development environment (or "IDE"). Like other IDE's, webAF is a component-based visual environment with a full set of drag-and-drop tools that generate Java code. But in the webAF case, much of this generated code specifically enables access to server-side SAS software and data through SAS-aware Java classes in Java programs (applications, applets, servlets). And webAF provides help in using these Java classes, in the form of SAS Components for applet development, and in the form of SAS Custom Tags in JavaServer Pages development.

webAF - SAS CUSTOM TAGS FOR JSPs

Generally speaking, a web page has content (e.g., user data or user results of some sort), and formatting instructions for how to present that content. JavaServer Pages (JSPs) are programs that create web pages when they execute. These programs have HTML code (markup language) and embedded Java code. Typically, the HTML is static and provides the page formatting instruction. You might think of this as the template for the page. The Java code can provide additional formatting, and it typically provides the content. And since the Java code executes at run-time, it can provide dynamic content.

In general, it is a good idea to separate the development of the format from that of the content. This way: (1) a web page designer can focus on the HTML that controls overall page design, and (2) an application developer, using Java technology, can generate the dynamic content portion of the page. JSPs enable this separation of function through use of Java's component-based technology. Think of components as Java programs that perform various functions. Many components help you generate the content (i.e., your application's business logic), and some help you generate the HTML used for presentation.

If you program Java, you can write the code and embed the Java programs in the JSPs with scriptlet tags. If you are not a Java programmer, you can employ pre-written, reusable cross-platform components, or tags as they are called. These components or tags are sometimes called JavaBeans or EnterpriseJavaBeans. You can make powerful interactive web pages this way. SAS has written its own JavaBeans, called InformationBeans and TransformationBeans. InformationBeans concern the data model – accessing the data and generating the data-driven content. TransformationBeans provide a sort of "view" or representation of the data model using a particular markup language such as HTML. They consume data from a webAF data model and transform it into device-specific markup language. (And you can even use TransformationBeans to retrieve the data for dynamic content).

These SAS Components are part of the SAS Custom Tag Library provided with webAF. The webAF program source code window for JSP development has an available Component Palette, which is how SAS delivers its SAS Custom Tag Library. You can drag a tag from the palette and drop it on the source window, and the default drop text syntax is displayed. You can then modify the tag attributes by editing the code in the source window, or you can make the modifications without writing code by modifying settings in the tag's available Properties and Customizer pop-up windows.

When you use the webAF source window to build a JSP, the various SAS Custom Tags on the webAF Component Palette are grouped together by function. There are several functions, and multiple tags are available per function, as follows:

- Form Elements – Tags to build HTML forms and the various form elements.
- Selectors – Tags to build tree views and menu bars for selection.
- Text – Tags to provide static text, date, and time.
- Data Viewers – A tag to provide a table viewer for the rows and columns in a SAS data set.
- Graphics – Tags for graphical presentation of SAS data, including bar chart, pie chart, scatter plot.
- SAS – Tags to interact with SAS, including:
  - Connection – to open a connection from the JSP to server-side SAS
  - SubmitInterface – to submit embedded SAS code for execution
  - LibraryListInterface – to list available SAS libraries
  - SASFileListInterface – to list SAS files in a library
  - DataSetListInterface – to list SAS data sets in a library
  - CatalogListInterface – to list SAS catalogs in a library
  - CatalogEntryListInterface – to list entries in a SAS catalog
  - DataSetInterface – to open a SAS data set for various purposes
  - LevelTreeInterface – to help open a special SAS data set that populates a tree view for selection

Following are examples of how this works. These are very simplified examples. Their only purpose is to show the process you go through, without writing much Java code, to build JSPs that call in and execute pre-written Java code when the JSP executes and is compiled into a servlet. They are not meant as a tutorial on JSPs or HTML. Thus, I may skip a step or two that is not pertinent to the purpose at hand. And, thus, the code shown may be incomplete and not execute exactly as is. The process is basically the same for most of the tags you would use. Each tag will call in its own pre-written Java code at run-time.

Example 1 – HTML Form for User Interface

This is a very simple example of a JSP where there is no connection from Java to server-side SAS yet. This

example uses SAS Custom Tags just to build an HTML form to provide a simple user interface and solicit a few user selections and inputs. The JSP asks the user to make one radio box choice, one or more check box choices, and one text entry. These selections and inputs would, when submitted, then lead to further processing which is not shown here. To start, assume you have a JSP project open in webAF, and a new JSP file for the project is open in the webAF source window. You would follow these steps to create the JSP:

- Enter some text for a page heading such as:
```
<br /><br />Survey of SAS System
Usage<br /><br />
```

- Drag a Form Tag from the component palette to the source window. You will see the following JSP code appear to indicate this is a Form Tag:
```
<sasads:Form id="form1" action="" >
</sasads:Form>
```

- Modify the Form Tag: Set the 'action' attribute to indicate what other JSP should execute when the user submits the selections and inputs:
```
<sasads:Form id="form1"
action="viewSurveyResults" >
</sasads:Form>
```

- Enter some text to request a selection, and…
- Drag a Radio Tag from the component palette to inside the Form Tag. You will see the following new JSP code:
```
Which is your favorite SAS
Product?<br />
<sasads:Radio id="radio1" model="" />
```

- Modify the Radio Tag: (1) Separate the end tag. (2) Remove the 'model' attribute. (3) Add text to provide the SAS Product choices:
```
Which is your favorite SAS
Product?<br />
<sasads:Radio id="radio1" model="" >
Base SAS
AppDev Studio
SAS/STAT
SAS/CONNECT
</sasads:Radio>
```

- Enter some text to solicit which SAS Procs are run, and…
- Drag 6 Checkbox Tags from the component palette to inside the Form Tag. You will see the following new JSP code:
```
<br />Select all the SAS Procs you
are likely to run:<br />
<sasads:Checkbox id="checkbox1"
text="Checkbox" value="selected" />
<sasads:Checkbox id="checkbox2"
text="Checkbox" value="selected" />
<sasads:Checkbox id="checkbox3"
text="Checkbox" value="selected" />
<sasads:Checkbox id="checkbox4"
text="Checkbox" value="selected" />
<sasads:Checkbox id="checkbox5"
text="Checkbox" value="selected" />
<sasads:Checkbox id="checkbox6"
text="Checkbox" value="selected" />
```

- Modify the Checkbox Tags: (1) Change the value of the 'id' attribute to a meaningful name related to the SAS Proc. (2) Change the value of the 'text' attribute to the SAS Proc. (3) Change the 'value' attribute so "Yes" is provided when a checkbox is selected.
```
<br />Select all the SAS Procs you
are likely to run:<br />
<sasads:Checkbox id="summary"
text="Summary" value="Yes" />
<sasads:Checkbox id="univar"
text="Univariate" value="Yes" />
<sasads:Checkbox id="freq"
text="Freq" value="Yes" />
<sasads:Checkbox id="sort"
text="Sort" value="Yes" />
<sasads:Checkbox id="dataSets"
text="DataSets" value="Yes" />
<sasads:Checkbox id="format"
text="Format" value="Yes" />
```

- Enter some text to solicit the favorite SAS Proc, and…
- Drag a TextEntry Tag from the component palette to inside the Form Tag. You will see the following new JSP code:
```
<br /><br />Enter the name of your
favorite SAS Proc:<br />
<sasads:TextEntry id="textEntry1" />
```

- Modify the TextEntry Tags (1) Change the value of the 'id' attribute to a meaningful name. (2) Add attributes for the display size of the entry field, and the maximum entry length allowed. You will see the following JSP code:
```
<sasads:TextEntry
    id="favoriteProc"
    size="50"
    maximumLength="50"/>
```

- Drag a PushButton Tag from the component palette to inside the Form Tag:
```
<sasads:PushButton id="pushButton1"
text="PushButton" />
```

- Modify the PushButton Tag: (1) Make the 'id' attribute a meaningful name. (2) Make the button text meaningful to the user. (3) Add the 'type' attribute to make it a submit button instead of a refresh button. (4) Add two line breaks to separate the pushbutton from the TextEntry Tag above it.
```
<br /><br />
<sasads:PushButton
    id="submit"
    text="Submit Selections"
    type="submit"/>
```

The JSP is now built. In this case, you did not write any Java code, and you did not write any HTML code except for the line break
. You have referenced a series of SAS Custom Tags, specifically Form Element Tags, through the webAF GUI. When the JSP runs, each tag calls in pre-written Java code to execute that in turn generates the HTML code to complete the web page. In this case, you have basically used the SAS Custom Tags of webAF to build a user interface to an application, but you have not yet engaged the SAS-aware Java classes that talk to server-side SAS. As a SAS programmer, you might already be viewing SAS Custom Tags as somewhat like SAS macro program invocations with macro parameter value setting.

Full JSP Program for Example 1

```
<%-- Copyright (c) 2002 by Planning Data
Systems, Inc. --%>
<%@taglib
uri="http://www.sas.com/taglib/sasads"
prefix="sasads"%>
<br /><br />Survey of SAS System Usage<br
/><br />
<sasads:Form id="form1"
action="viewSurveyResults" >
Which is your favorite SAS Product?<br />
<sasads:Radio id="radio1" model="">
Base SAS
AppDev Studio
SAS/STAT
SAS/CONNECT
</sasads:Radio>
<br />Select all the SAS Procs you are
likely to run:<br />
<sasads:Checkbox id="summary"
text="Summary" value="Yes" />
<sasads:Checkbox id="univar"
text="Univariate" value="Yes" />
<sasads:Checkbox id="freq" text="Freq"
value="Yes" />
<sasads:Checkbox id="sort" text="Sort"
value="Yes" />
<sasads:Checkbox id="dataSets"
text="DataSets" value="Yes" />
<sasads:Checkbox id="format"
text="Format" value="Yes" />
<br /><br />Enter the name of your
favorite SAS Proc:<br />
<sasads:TextEntry
        id="favoriteProc"
        size="50"
        maximumLength="50"/>
<br /><br />
<sasads:PushButton
        id="submit"
        text="Submit Selections"
        type="submit"/>
</sasads:Form>
```

HTML File Generated by the JSP for Example 1

```
<br /><br />Survey of SAS System Usage<br /><br />
<form name="form1" method="POST"
action="viewSurveyResults">
Which is your favorite SAS Product?<br />
<input type="radio" name="radio1" value="Base SAS">Base
SAS</input><br />
<input type="radio" name="radio1" value="AppDev
Studio">AppDev Studio</input><br />
<input type="radio" name="radio1"
value="SAS/STAT">SAS/STAT</input><br />
<input type="radio" name="radio1"
value="SAS/CONNECT">SAS/CONNECT</input><br />
<br />Select all the SAS Procs you are likely to run:<br />
<input type="checkbox" name="summary"
value="Yes">Summary</input>
<input type="checkbox" name="univar"
value="Yes">Univariate</input>
<input type="checkbox" name="freq"
value="Yes">Freq</input>
<input type="checkbox" name="sort"
value="Yes">Sort</input>
<input type="checkbox" name="dataSets"
value="Yes">DataSets</input>
<input type="checkbox" name="format"
value="Yes">Format</input>
<br /><br />Enter the name of your favorite SAS Proc:<br />
<input type="text" name="favoriteProc" maxlength="50"
size="50" />
<br /><br />
<input type="submit" name="submit" value="Submit
Selections" />
</form>
```

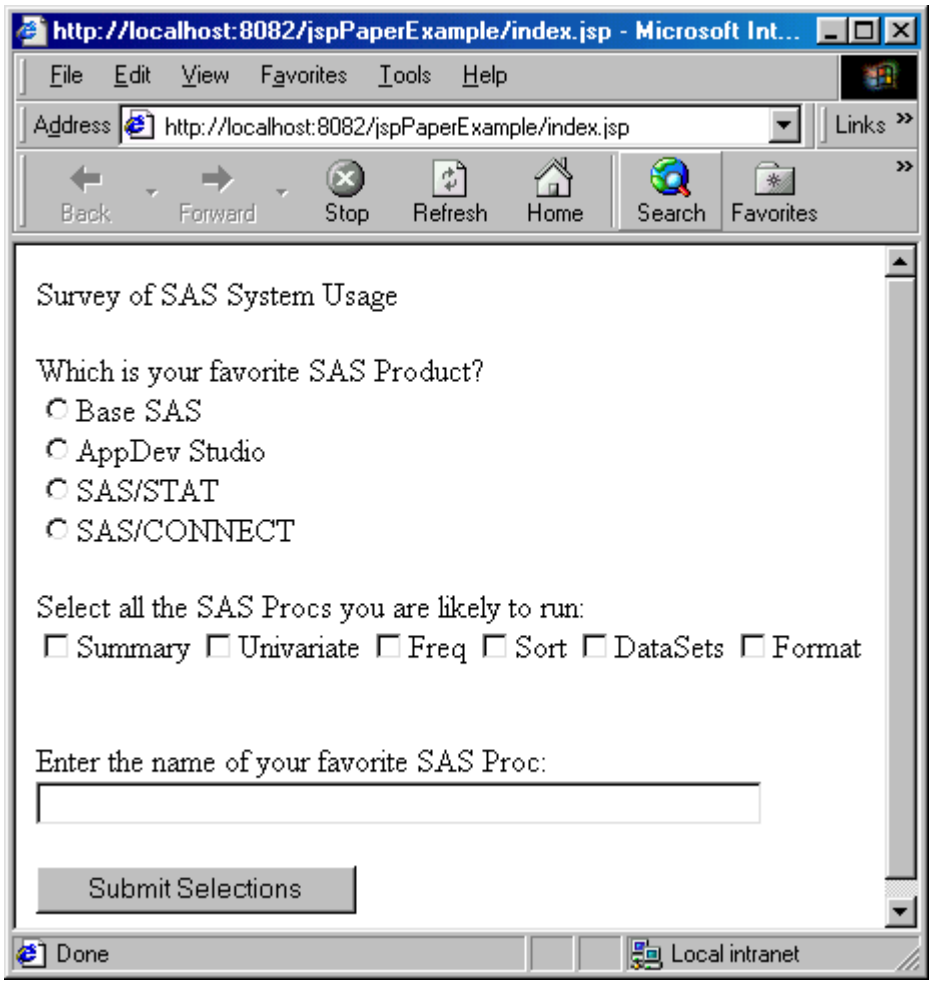

Figure 2: Web Page Result for Example 1

Example 2 – Connect to SAS Server, Submit SAS Code

This is an example of building a JSP that makes a connection to a SAS server, submits some Base SAS code to execute, and receives and displays the Listing results. Suppose you have been collecting in a data set the worldwide results of the SAS System Usage survey from Example 1. You will now run a Proc PRINT to display some worldwide survey results. This JSP will use a Connection Tag to open a connection to server-side SAS, and then use a SubmitInterface Tag to submit SAS code across that connection and receive results. To start, assume you have the same JSP project open in webAF that was used in Example 1. And you have added a hypertext link on the first JSP of the project to view worldwide survey results. This new JSP is called when the user clicks on the "View Worldwide Survey Results" hypertext link. You would follow these steps to create the JSP:

- Drag the Connection Tag from the component palette to the source window. You will see the following JSP code appear to indicate this is a Connection Tag:

```
<sasads:Connection id="connection1"
scope="session" />
```

- Drag the SubmitInterface Tag from the component palette to source window. A "Remote Connection" dialog box pops up. Select "Use Existing Connection", highlight the connect name "connection1", and press OK. You will see the following new JSP code:

```
<sasads:Submit id="submit1"
connection="connection1">
</sasads:Submit>
```

- Modify the SubmitInterface Tag: (1) Add the 'display' attribute to direct the Proc Print output to the web page. (2) Enter the SAS code to be submitted. You will see the following JSP code:

```
<sasads:Submit id="submit1"
 connection="connection1"
 display="OUTPUT">
  libname saserver 'D:/My
Documents/Sugis/sugi28/data';
  options nodate nonumber;
```

```
  proc print
data=saserver.wwsasSurveyResults(obs=
3);
  title 'Partial Listing of Worldwide
SAS Usage Survey';
  run;
</sasads:Submit>
```

The JSP is now built. You have referenced two SAS Custom Tags, specifically the Connection (to SAS) Tag and the SubmitInterface Tag, through the webAF GUI. When this web page is called for, this JSP program will be compiled into a Java servlet and executed. The servlet source code (the .java file) will have a substantial amount of Java code that pertains to the connection to SAS and the interface for submitting SAS code. This Java code is some of the SAS-aware Java classes available to you with webAF. You, of course, did not write any of this Java code. The two SAS Custom Tags provide it. Once again, you might think of these tags as somewhat like SAS macro program invocation with macro parameter value setting.

Full JSP Program for Example 2

```
<%-- Copyright (c) 2002 by Planning Data
Systems, Inc. --%>
<%@taglib
uri="http://www.sas.com/taglib/sasads"
prefix="sasads"%>
<sasads:Connection id="connection1"
scope="session" />
<sasads:Submit id="submit1"
 connection="connection1"
 display="OUTPUT">
  libname saserver 'D:/My
Documents/Sugis/sugi28/data';
  options nodate nonumber;
  proc print
data=saserver.wwsasSurveyResults(obs=3);
  title 'Partial Listing of Worldwide SAS
Usage Survey';
  run;
</sasads:Submit>
```

HTML File Generated by the JSP for Example 2

<pre>
Partial Listing of Worldwide SAS Usage Survey

Obs favprod summary univar freq sort datasets format favproc

1 AppDev Studio No Yes Yes No No No Contents
2 Base SAS Yes Yes No No Yes Yes Univariate
3 SAS/CONNECT Yes No No Yes No No Tabulate
</pre>

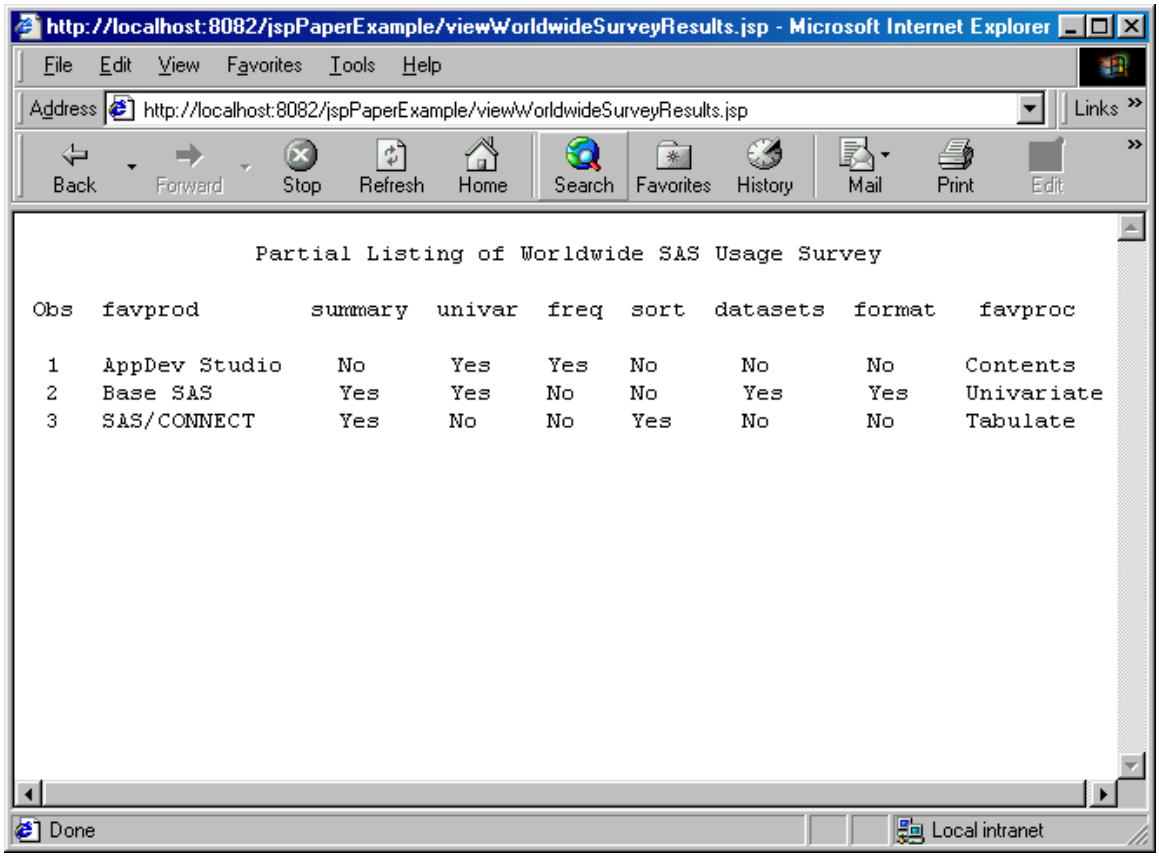

Figure 3: Web Page Result for Example 2

SAS Custom Tags Summary

These two examples are fairly trivial. When you develop a robust application, you will need to go beyond the Java code provided by the SAS Custom Tags and write your own Java code; however, even these trivial examples do indicate that:

- webAF provides a substantial amount of pre-written Java code (referred to as SAS-aware Java classes) to handle many of the basic tasks involved in building the client (or presentation layer) in n-tiered, web-enabled SAS applications.
- The basics covered include standard user interface and standard results presentations in the browser, connection from Java to server-side SAS, access to SAS compute and data services, and receipt of SAS results from these services.
- The pre-written Java code is readily accessible through the webAF GUI in the form of SAS Custom Tags.

webAF – SAS COMPONENTS FOR APPLETS

Another approach to building the client tier of the n-tiered application is to use client-side Java technology, (applets and applications), instead of server-side Java technology (servlets and JSPs). Java applications are stand-alone applications that run in a client application window, as would any application such as Microsoft Word or Excel. Java applets are Java applications that run within a web page displayed within a browser window. I will focus herein on Java applets, (and not applications), as the client-side Java technology in this paper, because I am focusing on web-enabled SAS applications in our new application landscape.

One common reason for using client-side Java is to create highly interactive user interfaces. Server-side Java is more limited in this regard because each time the user interacts with the active web page, the required processing must be done on the web server and Java server, even when the interaction does not involve calling new web pages and server-side SAS services. In contrast, with applets, the Java code for this interaction is downloaded to the client when the page is loaded to the browser, and it is then executed more quickly on the client when the interaction occurs. There are drawbacks to this approach, compared to server-side Java technology. Specifically, (1) web pages with applets load more slowly; (2) there can be minor browser compatibility issue regarding required Java plug-ins; and (3) there can be security restrictions on the client, or server, or both because the applet must be loaded to a client outside the firewall. Server-side Java technology does seem to be more in favor today, because it uses a thinner client, which means simpler deployment and more overall application security when the client sits outside the firewall. But applets are a viable component of the new application landscape because the more powerful interfaces they provide do offset the deployment difficulties in many instances. Thus, it is worthwhile to look at how SAS supports applet development through the webAF product of AppDev Studio.

In the context of our n-tiered, web-enabled, component-based SAS application landscape, web pages with Java applets have to provide the same functionality as do servlets and JSPs. That is, the Java code in the applets will have to: (1) provide general user interface interactivity; (2) request server-side SAS data and compute services; (3) receive the results of those services; and (4) present the results in web page display. Regarding the non-visual aspects of this work, the same pre-written, SAS-aware Java classes that are available for these purposes when using webAF to develop the Java code for servlets are also applicable and available when using webAF to develop applets. And regarding the visual aspects, Java classes that visually render in the applet space are used instead of the SAS Custom Tags for JSPs, because the Custom Tags generate only HTML interfaces while the applet is a Java interface. And as before, these classes are designed to facilitate application development by reducing the amount of Java code that a developer must write in order to implement common Java functions.

When you build an applet project with webAF, you will typically have an HTML file for the web page to be displayed, and a Java program file (or ".java" file) for the applet to run in the web page. The HTML code will include a "call" to the Java applet. webAF provides a "frame" window for visual display of your applet during development. And webAF provides a Component Palette for this window, just as it does for the source code window for JSPs. As you build an applet project in webAF, you can drag and drop visual and non-visual SAS Components onto your frame. So the SAS Components used for applet development have basically the identical role as do the SAS Custom Tags for JSP development. As you drag and drop SAS Components onto your applet frame, you are adding the Java code for the SAS-aware Java classes to your applet program (.java file). And, in the case of applets, you are also adding needed HTML code to your associated HTML file for the web page in which the applet will run.

Let's look a bit further at the development process. Suppose you had a very simple applet project. It has one web page with three visual widgets on it. On the top left is a list box for the user to select the name of a SAS library. On the bottom left is a list box for the user to select the name of a data set among all data sets in the selected library. This list box is populated with data set names once the library is selected. On the right is a table viewer used to display the rows and columns of the selected SAS data set. The steps in the applet development process in webAF would be roughly as follows. Assume the applet project has been opened in webAF and the frame window is displayed:

1. Drag the SAS Component for the ListBox Form Element to the top left side of the frame.
2. Drag the SAS Component for the ListBox Form Element to the bottom left side of the frame.
3. Drag the SAS Component for the TableView to the right side of the frame.
4. Create the model/view attachment needed to populate the top left ListBox:
   - Drag the non-visual SAS Component for the LibraryListInterface model and drop it on the top left ListBox view.
   - A SAS Connection component will be created when you do this first model/view attachment.

The Connection controls how the applet communicates with SAS.

5. Create the model/view attachment needed to populate the bottom left ListBox:
   - Drag the non-visual SAS Component for the DataSetListInterface model and drop it on the bottom left ListBox view.
6. Create the model/view attachment needed to populate the TableView:
   - Drag the non-visual SAS Component for the DataSetInterface model and drop it on the TableView view.
7. Establish property links between the components:
   - listBox1: property=selectedItem, sendToItem=dataSetListInterface1, (to display the list of data sets for the selected library)
   - listBox2: property=selectedItem, sendToItem=dataSetInterface1, (to display the data for the selected data set)

When you first open the applet project in webAF, a ".html" file is automatically created to launch the applet(s) to be created. The generated code in this file includes HTML and JavaScript, and it includes what is needed to open the web page and run the applet. A ".java" file is also created with initial Java code that is needed for the applet.

As steps 1-3 above are done, Java code is added to the applet's ".java" file concerning rendering the visual widgets inside the applet space. Then, as steps 4-6 are done, additional Java code is added to the ".java" file. This code involves objects instantiated from the SAS-aware Java classes pre-written by SAS. The objects are used in your applet program to: (a) establish a connection to server-side SAS, (b) request services such as to open a SAS session and read the available libraries, (c) receive the results of the request (i.e., a list of libraries), and (d) display the results in the list box. Finally, in step 7, the pop-up Properties window for each view component is used to set property values that modify the Java source code when the applet program is compiled. This modified code will cause the second list box to be populated with a list of data sets once the user selects a library in the first list box. And it will cause the TableView to be populated with the actual data from the data set, once the data set is selected. And you accomplished all this using the webAF SAS Components for applet development, without writing any Java code.

This is a substantial amount of Java application development work you are able to do, without writing any Java code yourself, for your n-tier, web-enabled SAS application. You will still need to be able to program in Java if you are building a more robust applet project; however, this simple example does suggest the substantial boost provided by webAF to your development effort through the use of SAS Components.

INTEGRATION TECHNOLOGIES – APPLICATION FACILITY

So far, I have focused on the webAF product within the AppDev Studio product suite. We have seen that webAF is the primary tool in the suite for Java program development in the new application landscape. Another product in the suite is SAS Integration Technologies (SAS/IT). webAF, and more specifically the Java programs you write with webAF, actually leverage the capabilities provided by SAS/IT.

SAS/IT provides an infrastructure that makes SAS compute and data servers available to other components in an enterprise-wide application. In other works, SAS/IT allows SAS to integrate seamlessly into the n-tier, web-enabled, distributed-object, component-based architecture we have been discussing. There are four major components to SAS/IT: Integrated Object Model (IOM), Application Facility, Lightweight Directory Access Protocol, and Publish/Subscribe. The first two have the most direct bearing upon our focus in this paper.

Integrated Object Model

The Integrated Object Model (IOM) provides an industry standard way for software components in an n-tier, distributed-object, component-based architecture to interact with each other across the various software vendor boundaries and platform boundaries. The interaction is based upon object-oriented programming methods. Each vendor writes object interfaces to its product, and then other vendors can write objects that work with those object interfaces. SAS has built a series of IOM object interfaces to various server-side SAS features such as the procedural scripting language, data, file system, results content, and formatting services. This enables you to use industry-standard languages, programming tools, and communication protocols to develop client programs that access these services on a SAS IOM server. SAS IOM interfaces can be scripted from a variety of languages such as: VB, VBA, VBScript, Java, C++, Delphi, PowerBuilder.

AppDev Studio (i.e., webAF) takes advantage of the SAS/IT infrastructure by providing access to Integrated Object Model (IOM) servers. This enables you to write Java programs (applets, stand-alone applications, servlets, and JavaServer Pages) that interact with an IOM SAS server.

Application Facility

SAS refers to its IOM object interfaces to server-side SAS as the SAS/IT "Application Facility". The object interfaces of the Application Facility within SAS IOM provide access to this SAS functionality:

- SAS Workspace - Methods to start a SAS session on a server or a local machine.
- FileService - Methods to create, access, and manage SAS filerefs and read or write files on the server's host file system.
- DataService - Methods to create, access, and manage SAS librefs and read or write SAS library members (e.g., data sets) on the server's host file system.
- LanguageService - Methods to submit SAS procedural scripting language statements (to execute DATA and PROC steps) and to retrieve the Log and List outputs. And, methods to monitor program progress including error conditions (through the LanguageService event interface).
- Utilities - Methods to access and create result packages, formats, and host information. For example, the ResultPackageService, which has

methods to retrieve output formats other than the Log and List, such as HTML.

In essence, SAS/IT, through IOM, allows server-side SAS to work with other applications (such as Java) in a distributed object, component-based manner. And SAS has used this capability in webAF to make our application development in today's landscape easier. Specifically, SAS has built a set of Java classes, (objects if you will), according to this standard, to provide access to various SAS services with which we will need to interact in our applications. We have seen this above in the discussion about SAS Custom Tags for JSPs, and in the discussion about SAS Components for applets. Both SAS Components for applets and SAS Custom Tags for JSPs engage precisely these pre-written Java classes.

So, when you write the client tier of your application using Java technology, and you develop the Java programs using webAF, you are leveraging the work SAS has already done with Java to interact with server-side SAS using the SAS/IT Application Facility. If you want to develop your own Java classes to interact with SAS, you can use the SAS/IT Application Facility for this. And, you can also use the SAS/IT Application Facility if you develop your client tier using another language. You can integrate your client tier and the backend SAS tier into the application if: (1) your other language allows you to write IOM object interfaces, and (2) you use SAS/IT to enable an IOM SAS server.

ACKNOWLEDGEMENTS

The author would like to thank Rich Main of the SAS Institute staff who gave generously of his time to review this paper and provide helpful feedback. However, any errors that remain are mine.

CONTACT INFORMATION

Barry R. Cohen, President
Planning Data Systems, Inc.
700 Ardmore Ave. #512
Ardmore, PA 19003
610 649 8701 (voice)
610 649 8408 (fax)
cohenbar@bellatlantic.net
SAS Alliance Consulting Partner