

Delivering OLAP Solutions to the Web

Tammy Gagliano, SAS Institute Inc., Carmel, IN
Tony Prier, SAS Institute Inc., Kansas City, MO

ABSTRACT

What is OLAP (Online Analytical Processing) and how can it benefit your organization? Can you or should you be delivering your OLAP applications via the web? If so, what tools are available to you?

These questions and more will be answered during the presentation, which will include live demonstrations of the Java technologies available to you for web-enabling your OLAP solutions. Using tools that are part of AppDev Studio™, you can develop applet-based or servlet-based (including JavaServer Pages™) OLAP applications. Comparisons will be made regarding the pros and cons of each approach as well as how the different technologies can be used together to provide a complete solution.

INTRODUCTION

The ability to look at data from multiple dimensions, or areas of interest, and to access that data quickly and in a consistent manner is key to the success of any business intelligence application. Today, OLAP (On-Line Analytical Processing) is acknowledged as a key technology for a successful implementation of any business intelligence system and is vital for creating strategic competitive advantages for any organization.

OLAP provides analysts, managers and executives the freedom to interrogate their enterprise data. Using multidimensionality, the data is organized according to the categories that reflect the way the user thinks about the enterprise.

The SAS OLAP solution uses SAS Institute's multidimensional database (MDDb) server to provide high-performance OLAP capabilities within an integrated data warehouse environment. MDDb's package warehouse data into multidimensional data structures, which deliver data to OLAP client software upon request. The SAS OLAP Server bundles all the required server-side functionality you need for defining your OLAP data. Using the SAS OLAP Server you can:

- create MDDb's using PROC MDDb,
- register OLAP Metadata,
- create Access Control definitions,
- use Model Coordination.

The SAS OLAP server also supports Hybrid OLAP (HOLAP). HOLAP provides the ability to partition your multidimensional databases so that data can be split across multiple MDDb's that could reside on different servers. Data can also be accessed from Relational databases, SAS data sets, summary data sets, flat files, etc. This is all meta data driven and often provides a much more open and scalable solution.

With SAS OLAP Server on the back-end, you can access data from any source – in any format. Now, you are ready to choose the front-end or the client interface to the data. For that, many organizations today are turning towards thin-client solutions – interfaces that can be easily deployed through the Internet or corporate-owned intranets.

The growing use of the Web as a way to deliver client services also results in the reduction of the costs associated with upgrading applications. The role of the client interface becomes that of a viewer of information delivered by the server. Adding a new OLAP user can be as simple as e-mailing them the URL of the OLAP application.

The SAS OLAP solution is web enabled and supports a full range of client/server configurations including totally thin-client. By relying on MDDb's on the server side for storage and processing, SAS OLAP clients can be used solely as viewers of MDDb data. Users can run queries and generate reports from their browser without the need to run a SAS session on the client. The SAS OLAP solution provides:

- web publishing tools such as the SAS Output Delivery System (ODS) available as part of base SAS software in Version 7 as well as the HTML Formatting Tools available with Release 6.12 of the SAS System.
- the Application Dispatcher and htmSQL which are both part of SAS/IntrNet™ software and are based on CGI technology.
- Java-based technology such as servlets, JavaServer Pages (JSP), applets or applications.

For the remainder of this paper, we're going to focus on the Java-based technologies available in AppDev Studio and how they fit as part of the SAS OLAP solution. AppDev Studio (ADS) is a complete, stand-alone development environment. It is an integrated suite of development tools that provides the power you need to build web-enabled applications that use HTML, CGI, Java servlets and JavaServer Pages (JSP) as well as sophisticated Java applications and applets. ADS contains many OLAP specific components and interfaces to help build sophisticated yet easy-to-use OLAP documents and deploy them to the web.

APPDEV STUDIO: WEBAF™ AND WEBEIS™ SOFTWARE

webAF software

Within the AppDev Studio suite of products, webAF software is the primary development tool for Java-based applications. It helps you build applications that are lightweight, easy to manage, and instantly connect to SAS software. Support for the creation and debugging of applets, servlets and JavaServer Pages is provided. webAF software's component-based visual development environment enables easy access to SAS software from Java classes, transparent access to SAS/AF objects, access to JDBC data sources, access to remote tables and MDDb's stored on one or more SAS servers, and access to SAS compute power through remote procedure submissions.

If you are a webAF user, you will have to have some knowledge of Java as it is a Java development tool. Code is automatically generated for you as you drag-and-drop your components into your project and it also provides a simple interface for linking properties and setting event handlers to get components talking to each other.

However, you still need to have a good understanding of the underlying language in order to build a sophisticated application.

webEIS software

For those of you who are not Java experts, but still need to build OLAP applications, you should take a look at webEIS software. webEIS covers the entire spectrum of user needs – from business executives who want an easy-to-use document for viewing multidimensional data to business analysts who expect sophisticated reach through, data visualization, analysis and reporting capabilities. It is an application written in Java that makes it easy to create interactive, EIS-style documents containing charts and multidimensional tables. A webEIS document is published on the Web as a Java applet or JSP. You do not write any Java code to create these applications. They are created purely using the intuitive and powerful point-and-click, drag-and-drop interface provided by webEIS software. It is also important to note that any webEIS documents you create can easily be incorporated into a webAF project for further customization using Java.

Using webEIS, you can point to existing MDDB or HOLAP data on a remote SAS server. You can also point to an existing SAS/EIS object, which enables you to reuse functionality and behavior defined in the remote server object. Once these documents are deployed, your users can:

- navigate intuitively through volumes of data using drill down, expand/collapse or subsetting operations.
- perform analysis, and row and column calculations at run time.
- apply traffic lighting or exception highlighting to record important trends or outliers in the data.
- dynamically reach through to detail data anywhere on the network without having to know where the data resides.
- and more.

All of this functionality is built into the various MDDB components that are used in the background. It allows users to work interactively on the client from their Web browser while they are unknowingly communicating back to a SAS server where the data is stored. webEIS is implemented in a multi-user client/server mode and offers consistent rapid response to queries, regardless of database size and complexity.

During this presentation, we will walk you through how to build an OLAP document using webEIS and then show how easy it is to deploy the document either as an applet or JSP – without you having to write a single line of code!

JAVA TECHNOLOGY

We've talked about the tools that will enable you to deploy your OLAP applications using Java technology, but we haven't discussed why you should consider choosing Java as your development platform to begin with. Just why is it that Java is considered by some to be the premier language of choice for providing highly interactive user interfaces to the Web browser?

Java technology is continuously being enhanced to provide components and features that elegantly handle problems that are difficult in traditional programming languages such as multithreading, database access, network programming and distributed processing. It is ideally suited for the Web because it is:

- portable across platforms by virtue of it being an interpreted language. A Java Virtual Machine (JVM) must be available on the user's machine. Most browsers (e.g. Netscape and Internet Explorer) contain a JVM as part of their standard installation.
- secure through its ability to maintain the integrity of the client machine. The JVM has the opportunity to enforce the rules

specified by the security manager to ensure that the integrity of the user's machine is maintained and that the applet does not have access to resources other than those the user has specifically granted. In addition, it allows vendors to digitally sign the archive file to identify the vendor that created the JAR file. This allows the user to decide whether they "trust" the software provided by this vendor.

- considered to be a true thin client solution because of its ability to be dynamically downloaded on demand versus permanently installed on the user's machine. This eliminates the user or IS staff at your site from having to install and maintain current versions of software on each client machine.

JavaBeans™

Another reason that Java is so popular is because of its JavaBeans and Enterprise JavaBeans architectures. These object-oriented frameworks allow for the building of some very powerful components that make it easy for developers to create, deploy and manage cross-platform applications. webAF offers its own set of JavaBeans compliant components referred to as InformationBeans™. These beans allow you to tap into the enterprise data access, data warehousing, and decision-support capabilities of SAS software.

You can build sophisticated web applications that can:

- access SAS data libraries on a remote server allowing access to any data source that SAS can access through its extensive list of database engines.
- display SAS multidimensional databases in a ready-to-use OLAP viewer that has built-in functionality for drilling down through the data, subsetting, exporting the data to a spreadsheet, applying exception highlighting, adding computed columns and more.
- perform compute services by submitting SAS code on the server to perform tasks such as statistical analysis, reporting, summarization, and quality control -- just to name a few.

There are many components available with webAF that are specifically designed for generating custom OLAP applications – accessing the power of SAS on the server for both data (MDDBs) and compute resources. For example, the `com.sas.sasserver.mdtable.MultidimensionalTableV3Interface` is an InformationBean that is designed specifically to read and manipulate MDDB data.

The data supplied by this model can be displayed on the web using one of the following viewer components:

- in a Java applet using the `MultidimensionalTableView` component (found in `com.sas.mdtable`)
- or in JavaServer Pages/Servlets using the `MDTableTransformationBean™` (found in `com.sas.servlet.beans.mdtable.html`). TransformationBeans are discussed in more detail later in this paper.

Both of these viewers were designed to display MDDB data in a table format and communicate with the above model through model/view communication. Model/view communication enables a viewer (typically a visual control) to communicate with a model (typically a non-visual component) based on a set of common methods that are defined in an interface. The viewers seamlessly communicate with its attached model without you having to write additional Java code to perform tasks such as retrieving rows to display, handling updates that might be made through client-side table interaction and more.

InformationBeans virtually open the door to SAS, which enables

your web applications to take advantage of any and all functionality that SAS software provides. And using AppDev Studio, the power of having SAS on the server can be exploited without having SAS software installed on the client machine.

Java Applets

Applets lend themselves nicely for creating highly interactive user interfaces for thin-client applications. With applets, you avoid having to install an application locally on a user's machine. Instead, when an applet is executed (usually by being called from within an HTML page), the necessary files are automatically downloaded from the Web server. The applet is then loaded into memory and displayed in the browser. Typically, applets are subject to security restrictions on the client, the server, or both. Make sure that you understand any limitations that your production web environment may impose.

webAF's Project Wizard quickly steps you through creating an applet. Then you can begin building the pieces of your application using webAF's drag and drop interface to add visual and non-visual components to a window.

Even though the power of Java makes applets a popular choice among many Web application developers, applets present some deployment hurdles that you should be aware of. By "deployment" we mean the mechanism by which the applet is made available to the Web browser user.

Some of the most common complaints you hear about applets are:

- the need for the Java Plug-in which is an ActiveX control. This raises some security concerns for sites, which do not allow downloading of any ActiveX technology.
- the start-up time required to download an applet's implementation files and all required class libraries from the web server host to the client machine.
- the amount of time it takes the applet to set-up the server side environment – for example, to instantiate a SAS session and load large tables in that session that are then accessed from within the applet.

AppDev Studio developed applets require a Java VM that is JDK 1.3 or later. Popular web browsers like IE 5.0 and Netscape 4.5 do not natively support such levels of the Java VM. In response, Sun has produced the Java Plug-in to allow applets to run with the JDK level that they require in these browser environments. As a result ADS developed applets need to use the Java Plug-in in order to run in these browsers. Some sites do not allow downloading of any ActiveX controls because these controls have access to the local machine and can create havoc if coming from an unreliable source. One way to address the security issue that the Plug-in raises is to install it on a local *intranet* where the browser can be configured to allow it to be downloaded by applets as needed. This allows use of the Plug-in without allowing other ActiveX controls to be downloaded from the Internet. Another option, which avoids the ActiveX/Java Plug-in issue altogether is to consider using servlets/JSPs to deploy your application which is discussed later in this paper.

The latter two concerns from the above list fall under the 'applets are slow' category. Both tasks are performed every time the applet is invoked. When dealing with a slow network or dial-up connection, slow applet performance can severely limit their effectiveness.

To help overcome these problems, AppDev Studio provides a number of techniques that you can use. Where applet download times are a problem, SASNetCopy or JSASNetCopy along with the Java Plug-in's applet caching feature can be used to install the applet classes and any required extension classes onto the

client machine. Both of these technologies use a zero-administration, auto-install approach. Using either technology, the applet's required class libraries can be downloaded automatically the first time that the applet HTML is referenced and then cached on the client machine for subsequent re-use. Updated class libraries are made available by simply installing them on the web server and modifying a server-side configuration file. Then, the next time a user accesses the applet HTML, the updated class libraries will automatically be downloaded and cached on the client machine.

Where SAS startup time is an issue, the AppDev Studio Middleware Server's SAS session preloading and MDDB or class preloading feature can be used such that the required SAS support is available to a client virtually instantaneously. This feature can be used for servlet/JSP environments as well.

For more details on these techniques, refer to the following papers available from the AppDev Studio Developer's Site:

- Improving Applet Performance: SASNetCopy and JSASNetCopy:
<http://www.sas.com/rnd/appdev/Tools/applet-performance.htm>
- AppDev Studio Middleware Working with User Profiles:
<http://www.sas.com/rnd/appdev/doc/MWSprofiles.htm>

Java Servlets and JavaServer Pages™ (JSP)

While the techniques described above will help speed up applet startup times, applet technology may still present some problems that lend themselves towards a look at servlet or JavaServer Page technology instead. For example, suppose you have an applet that needs a connection to a SAS session on a remote server in order to take advantage of data stored there or the compute power of a SAS server. If you are deploying your solution in a more restrictive network environment (for example through firewalls, proxy servers, or using secure HTTP), you may have to undertake considerable effort to get applet solutions to work. In such a network environment, you should also consider using a server-side Java solution such as JavaServer Pages or Java servlets. These solutions simply return HTML back to the client device while the code runs on the application server machine. Both the server-side program and the SAS server reside behind the firewall.

At a conceptual level, servlets are just like applets except that they run in the server environment instead of the browser environment.

In the firewall scenario mentioned above, hosting the processing on the server eliminates the problems with communications from the client machine to the SAS server. Additionally HTTPS is fully supported since the only requests being passed between the client and the web server are HTTP requests.

JavaServer Pages are actually an extension of the Java Servlet API. However, developing an application based on JSP technology does not require in-depth knowledge of how servlets work. JSP technology makes it easier to build web pages with dynamically generated content through the use of Java's component-based technology. It separates the user interface from the application logic, which enables:

- the page designer to focus on writing the HTML that controls the overall page design.
- the application developer, using JSP tags (or scriptlets), to generate the dynamic content portion of the page.

Java is the native scripting language for JSP, which means you can develop platform-independent applications due to Java's "Write Once, Run Anywhere" characteristic. In the simplest

terms, a JSP page is simply an HTML page with embedded Java code. If you are comfortable writing Java code, you can embed Java programs directly in the JSP page using scriptlet tags.

If you're not a programmer, you can take advantage of reusable, cross-platform components (JavaBeans or Enterprise JavaBeans) with JSP-specific XML tags (e.g. USEBEAN tag) that make it simple to instantiate JavaBeans components and manipulate properties on a component from within your web page.

Through the use of this component-based logic, page developers are able to develop sophisticated, interactive web-based applications with very little Java programming knowledge. JavaServer Pages provide other benefits as well. Execution of JSP pages is simple and fast because they can be executed on any Java-enabled Web server, application server, or operating system. This differs from other technologies that have specific server requirements. For example, Microsoft's Active Server Pages™ technology is dependent on other Microsoft technology (such as COM).

JSP technology holds advantages over traditional CGI-based solutions, which have shown limitations with respect to scalability. With each CGI request, a new process on the server is launched. When multiple users access the program concurrently, these processes can quickly consume all of the web server's available resources and can bring the application to a halt. When a JSP page is first called, if it does not yet exist, it is compiled into a Java servlet class and stored in the server memory. A Java servlet is a Java-based program that runs on the server as opposed to an applet, which runs on the browser. This enables very fast responses for subsequent calls to that page (and avoids the CGI-bin problem of spawning a new process for each HTTP request, or the runtime parsing required by server-side includes).

Finally, JSP differs from other technologies because it utilizes reusable components and tags, instead of relying heavily upon scripting within the page itself. Through its use of servlet technology and Java server-side processing, it offers:

- scalability for complex, dynamic web pages
- a true thin-client deployment strategy (with an even smaller footprint than applets which require the Java classes to be downloaded to the client)
- persistence due to Java's true session management capabilities.

However, like CGI, the graphical user-interface (GUI) portion of the application is somewhat limited to what the HTML form elements can provide. For more detail on comparing this technology to CGI or Applets, refer to *Getting Started with AppDev Studio, First Edition*.

In addition to what the servlet/JSP platform provides, AppDev Studio offers additional functionality, which makes the development of web pages in the servlet/JSP environment even easier. Those features are:

- TransformationBeans (available for both servlets and JSPs)
- Custom Tag Library for the TransformationBeans and key InformationBeans (custom tag technology available for use within JSPs only)

TransformationBeans

TransformationBeans are a set of specialized JavaBeans included with webAF. These beans are designed to consume data from an existing webAF model (e.g., using the DataSetInterface model that retrieves data from a SAS data set) and transform it into HTML to display on the Web page (e.g., using the Table TransformationBean, which displays the data in

an HTML table).

When using webAF's InformationBeans and TransformationBeans together, not only does the page author have access to the power of SAS on a remote server but they also spend less time writing HTML. The TransformationBeans do all the work!

The table below, lists the TransformationBeans you will find useful when building an OLAP application:

Package: com.sas.servlet.beans.mddbtable.html	
TransformationBean	Description
MDBar	creates an HTML Bar chart image that represents data stored in a MDDB table
MDCombination	creates an HTML Combination chart image that represents data stored in a MDDB table
MDDrillPath	displays text that corresponds to the current drill path within an MDDB table
MDExportToExcel	exports MDDB data to MS Excel (using CSV file format)
MDFinder	generates HTML to represent a selector that allows a user to specify a text string to search for in a selected Column of an MDDB table.
MDNavigationBar	generates HTML visuals which allow a user to scroll within an MDTTable Transformation Bean
MDPie	creates an HTML Pie chart image that represents data stored in a Multidimensional Database.
MDQuerySelector	generates HTML and JavaScript to represent a selector that allows a user to dynamically change Rows, Columns, Measures and Statistics displayed in an MDTTable TransformationBean
MDScatter	creates an HTML Scatter chart image that represents data stored in a MDDB
MDSegmentedBar	creates an HTML SegmentedBar chart image that represents data stored in a MDDB
MDSelectorMenuItem	used to populate a Menu item in the MenuBar TransformationBean.
MDSortSelector	generates HTML and JavaScript to represent a selector that allows a user to change the sorting criteria for MDDB data displayed in a MDTTable TransformationBean
MDSubsetSelector	generates HTML and JavaScript to represent a selector that allows a user to change the subset criteria for

	MDDB data displayed in a MDTable TransformationBean
MDTable	generates HTML tables to view data stored in a MDDB (attached to the MultidimensionalTableV3Interface InformationBean)
MDTopBottomSelector	generates HTML and JavaScript to represent a selector that allows a user to subset according to top N/bottom N criteria for MDDB data displayed in a MDTable TransformationBean
MDTotalsSelector	generates HTML and JavaScript to represent a selector that allows a user to specify totals to be displayed in a MDTable TransformationBean

Package: com.sas.servlet.beans.html	
TransformationBean	Description
MenuBar	generates HTML that builds a menubar which can be populated with Menu items and MDSelectorMenuItem items which launch the various TransformationBean selectors described in the above table
Menu	used in conjunction with the MenuBar and MDSelectorMenuItem components to populate a MenuBar

Note: All HTML beans adhere to functionality available in HTML 3.2.

There are many more TransformationBeans available in the com.sas.servlet.beans.html package than what is listed in the above table. To see a complete list along with examples, use the following link to go to the *Server Side Examples* page on the AppDev Studio Developer's Site:
<http://www.sas.com/rnd/appdev/webAF/server/examples.htm>

At first glance, it might seem that TransformationBeans are simply a set of convenience objects that help a Java developer implement an HTML page. Compare the HTML code necessary to place a form input element such as a check box on a page

```
<input type="checkbox" name="box1"
      value="checked" checked>Label
```

with the Java code that implements a CheckBox TransformationBean

```
<%
com.sas.servlet.beans.html.Checkbox
checkbox = new
com.sas.servlet.beans.html.Checkbox(
"box1", "Label", true, "checked");
checkbox.write(out);
%>
```

The simple HTML code is straightforward and easy to read. However, the TransformationBean makes it easy to implement

dynamic content. If you wanted to simply check or uncheck the form control based on the value of a boolean variable named **status** defined in your JSP, the code without the TransformationBean looks like

```
<% if (status) { %>
<input type="checkbox" name="checkbox1"
      value="checked" checked>Label
<% } else { %>
<input type="checkbox" name="checkbox1"
      value="checked">Label
<% }%>
```

Using the TransformationBean, the JSP code contains

```
<%
com.sas.servlet.beans.html.Checkbox
checkbox = new
com.sas.servlet.beans.html.Checkbox(
"checkbox1", "Label", status,
"checked");
checkbox.write(out);
%>
```

The use of the bean not only aids in dynamic content generation, but it is also easy to read and debug.

The above TransformationBean example was taken from a paper found on the AppDev Studio Developer's Site called *Why use TransformationBeans?* To read the complete article use the following link:

<http://www.sas.com/rnd/appdev/webAF/server/whybeans.htm>

SAS Custom Tag Library

Along with the standard scriptlet and JSP specific tags, the JavaServer Pages specification also supports something called *custom tags*. A custom tag is a user-defined JSP language element.

Custom tags are usually distributed in the form of a *tag library*, which defines a set of related custom tags and contains the objects that implement the tags.

webAF offers an extensive tag library that corresponds to the TransformationBeans and key InformationBeans that you would want to use in your JSP applications. You do not have to be a Java expert to use these custom tags. The tags are HTML- or XML-like. You simply specify values for the attributes on a given tag statement in your JSP.

Continuing with the Checkbox TransformationBean example shown above, the equivalent custom tag code would look as follows:

```
<sasads:Checkbox id="checkbox1"
  text="Label" selected="<%= status %>"
  value="checked" />
```

In the section labeled *USING WEBAF TO BUILD A JSP-BASED OLAP APPLICATION* found later in this paper, you will see an OLAP example of the custom tag code that gets generated as you drag and drop components from the webAF toolbar onto your JSP project. The custom tag source code that gets generated is much easier to read and maintain versus scriptlet or standard Java code. As a result, you'll find that using custom tags increases your productivity because of the encapsulation they provide for more than often very complex tasks.

For more details on the SAS Custom Tag Library and what it has to offer, go to the following papers found on the AppDev Studio

Developer's Site:

- WebAF and the SAS Custom Tag Library
<http://www.sas.com/rnd/appdev/webAF/server/customtags.htm>
- SAS Custom Tags: Overview
<http://www.sas.com/rnd/appdev/webAF/taglib.htm>.

LETTING WEBEIS GENERATE YOUR OLAP APPLICATION FOR YOU

One of the easiest and quickest ways for you to design an OLAP document and deploy it to the web is to use webEIS software. Using webEIS, you do not need to write a single line of code. It provides you with a friendly user-interface to perform all the tasks you need such as

- creating OLAP documents that have one or more sections to them.
- choosing a remote data source.
- adding components to the sections within your document.
- rearranging the components and change their properties to give them the exact behavior and look and feel that you desire.
- deploying the finished document as either an applet or JSP.

Saving the document as a JSP application is new with Version 2 of AppDev Studio. Instead of choosing **File->Save as Applet...** from the webEIS menu,

1. Select **File -> Save As JSP...**
2. In the Save as JSP window, note the name of the JSP and the path where all dependent files will be saved. By default, JSPs are named using the name of the current document. They are saved to the default Web server location if one has been specified; otherwise, they are saved to the current document directory.

When you save a document as a JSP, several JSP files are created:

- a main JSP file
 - one JSP file for each section
 - any associated resource dependencies (for example, images that were specified with absolute file paths) are also saved.
3. (Optional) Click the **Package files as...** check box if a Web server has not been specified. All files are then saved as a .zip file. You can easily move the .zip file to another location, such as a Web server.

The EIS document itself is still available and is saved in a file with a .eis extension. This document can be edited later within webEIS. Additional applets and/or JSPs can be generated from it at any time.

Some developers like to have more control over their application or add additional behavior than maybe what a user-interface like webEIS provides. For example, an OLAP document may be only a small part of a much bigger application. The application may need to surface additional reports against relational data or submit other tasks such as more sophisticated data analysis or forecasting. Or, some developers simply like to write their own code instead of letting an application generate it for them. If you're one of these people, you may prefer to use webAF software to develop your OLAP applications.

The underlying OLAP components are the same between the two products. webEIS uses the MultidimensionalTableV3Interface to access MDDB data from a remote SAS server. You would use the same model in webAF regardless of whether you're building applets or servlets. The difference between the two tools

obviously being that in webAF, you will be writing the code to make the various components communicate with each other and to control the overall behavior of the application. The remaining two sections of this paper focus on using webAF to build your OLAP solutions.

USING WEBAF TO BUILD A JSP-BASED OLAP APPLICATION

With webAF, you can choose to create either a JSP or servlet project. It's not unlike the steps you go through when building an applet. webAF offers various components that are available from its component palette. You drag and drop those components onto the project you are creating. Some code gets generated automatically for you. You can use various wizards, dialogs and properties windows to specify options and control some of the behavior. But more often than not, you will end up going directly to the code and writing some yourself to complete the application!

First, let's take a look at building a JSP-based OLAP application using webAF. We'll take a look at the custom tag code that gets generated and how to manipulate that code. In the next section, we'll show the same example from a servlet perspective. Again, the underlying components used are the same – only the implementation differs. One uses custom tags and scriptlets mixed with HTML elements... the other uses pure Java code.

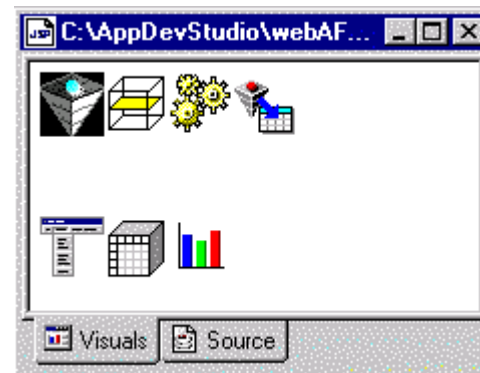
To begin creating a JSP project, select **File→New** from the webAF menu. When the New window appears,

1. select JavaServer Pages Project from the Projects tab
2. type sugi26 in the Project Name field
3. select OK
4. select Finish in the Project Wizard – JSP options window.

Change the component palette from JSP/Servlet to MDDB JSP/Servlet. Now we need to drag the six components needed for this OLAP application onto the project. Select the Visuals tab in your webAF project frame area, then drag a

- MultidimensionalTableV3Interface from the SAS tab (a Connection object will automatically be created along with this component since it needs to make a remote connection to SAS to get the data),
- MDCommandProcessor from the Data Viewers tab,
- MDExportToExcel from the MDDB Tasks palette,
- MenuBar from the Selectors tab,
- MDTable from the Data Viewers tab,
- and MDBar from the Graphics tab.

The Visuals tab within webAF should look like the following after dropping the components.



Open the customizer for MDModel1. On the Data Source tab,

1. type sugidemo in the Metabase field

2. type SUGIDEMO . CBMDDB in the Database field.

On the Query tab,

1. type Product Hierarchy for Add New Item and select the Add button for Rows
2. type Time Hierarchy for Add New Item and select the Add button for Columns.

On the Measures tab type,

1. Actual for Measure and SUM for statistic
2. Forecast for Measure and SUM for statistic
3. Difference for Measure and SUM for statistic.

On the Totals tab type,

1. Product Group for Level
2. Total for Label
3. true for state.

Close the MDModel1 customizer.

Open the property sheet for MDEXportToExcel1, type

1. index.jsp in the formAction field
2. exportForm in the formName field
3. false in the render field.

Setting render to false will prevent the tag from writing itself out at the location of the custom tag. The tag will be rendered later when it is referenced as a menu item in the Menu bar. Close the Property Sheet Window.

Open the customizer for menuBar1. On the Menu Bar tab,

1. select MenuBar in the Tree
2. select SELECTOR_EXPAND for Menu Type
3. type | in the separator field
4. select Menu in the Tree
5. type Subset in the Label field
6. type /assets/subset.gif in the Image field
7. type Subset in the Alternative Text field
8. expand the Subset Node in the Tree
9. select MDSelectorMenuItem in the tree
10. type MDModel1 in the Model field
11. select SUBSET_SELECTOR from Selector Type choice box
12. select MenuBar in the Tree
13. select New→Child
14. type Export to Excel in the Label field
15. type /assets/export.gif in the Image field
16. type Export To Excel in the Alternative Text field
17. type document.exportForm.submit(); in the Custom Action field
18. select New→Child
19. select MDSelectorMenuItem from the Menu Item Type choice box
20. type MDEXportToExcel1 in the Selector field

Close the customizer for menuBar1.

Open the customizer for MDTable1. On the MDTable tab,

1. type MDModel1 in the Model field
2. set the border width to 1
3. type MDCommandProcessor1 in the CommandProcessor field.

On the NavBar tab type,

1. /assets/double_left_03b.gif in the First

field

2. /assets/left_03b.gif in the Previous field
3. /assets/right_03b.gif in the Next field
4. /assets/double_right_03b.gif in the Last field.

Close the customizer for MDTable1.

All the properties of the MDTable are not displayed in the customizer. To view all the attributes available in the custom tag open the help window for the MDTable custom tag. To open the help window,

1. select MDTable1 in the Project Navigator,
2. use the right mouse button to display a pop-menu
3. select help from the pop-menu.

Switch to the Source tab in webAF project. Edit the MDTable1 custom Tag directly to specify additional attributes. New attributes are in **bold**.

```
<sasads:MDTable id="MDTable1" maxRows="25"
  maxColumns="10" scope="session"
  borderWidth="1"
  commandProcessor="MDCommandProcessor1"
  model="MDModel1"
  detailDataStyleSheet=
    "/assets/sasads.css"
  upArrowImage="/assets/up_03b.gif"
  rightArrowImage="/assets/right_03b.gif"
  leftArrowImage="/assets/left_03b.gif" >
```

```
<sasads:MDNavigationBar
  doubleLeftArrowImage=
    "/assets/double_left_03b.gif"
  doubleRightArrowImage=
    "/assets/double_right_03b.gif"
  leftArrowImage="/assets/left_03b.gif"
  rightArrowImage=
    "/assets/right_03b.gif"
  disabledDoubleLeftArrowImage=
    "/assets/double_left_03g.gif"
  disabledDoubleRightArrowImage=
    "/assets/double_right_03g.gif"
  disabledLeftArrowImage=
    "/assets/left_03g.gif"
  disabledRightArrowImage=
    "/assets/right_03g.gif" />
```

```
</sasads:MDTable>
```

Open the property sheet for MDBar1, specify

1. MDCommandProcessor1 for the commandProcessor property
2. MDModel1 for the Model property.

Close the property sheet.

To complete the JSP page we need to add a few HTML tags to complete the HTML. The complete index.jsp file is shown on the following page. The added HTML tags are shown in bold.

```

<%-- Copyright (c) 2001 by SAS Institute Inc., Cary, NC 27513 --%>
<%@taglib uri="http://www.sas.com/taglib/sasads" prefix="sasads"%>

<sasads:Connection id="D2159_s_PC" serverArchitecture="PC" persistedName="D2159's PC"
  command="sas.exe -dmr -comamid tcp -noterminal -cleanup" host="D2159"
  scope="session" />

<sasads:MDModel id="MDModel1" connection="D2159_s_PC" scope="session" metabase="sugidemo"
  database="SUGIDEMO.CBMDDDB" >
  <sasads:MDRowAxis >Product Hierarchy</sasads:MDRowAxis>
  <sasads:MDColumnAxis >Time Hierarchy</sasads:MDColumnAxis>
  <sasads:MDMeasure measure="Actual" selectedStatistics="SUM" />
  <sasads:MDMeasure measure="Forecast" selectedStatistics="SUM" />
  <sasads:MDMeasure measure="Difference" selectedStatistics="SUM" />
  <sasads:MDTotal level="Product Group" label="Total" state="true" />
</sasads:MDModel>

<sasads:MDCommandProcessor id="MDCommandProcessor1" scope="session" />

<sasads:MDEXportToExcel id="MDEXportToExcel1" formName="exportForm" formAction="index.jsp"
  render="false"/>
<html>
<head>
<link rel="stylesheet" type="text/css" href="/assets/sasads.css">
</head>
<body>
<sasads:MenuBar id="menuBar1" menuType="SELECTOR_EXPAND" separator="|" >
  <sasads:Menu label="Subset" image="/assets/subset.gif" alternateText="Subset" >
    <sasads:MDSelectorMenuItem model="MDModel1" selectorType="SUBSET_SELECTOR" />
  </sasads:Menu>
  <sasads:Menu label="Export To Excel" image="/assets/export.gif" alternateText="Export
to Excel" customAction="document.exportForm.submit();" >
    <sasads:MDSelectorMenuItem selector="MDEXportToExcel1" />
  </sasads:Menu>
</sasads:MenuBar>

<sasads:MDTable id="MDTable1" model="MDModel1" commandProcessor="MDCommandProcessor1"
  maxRows="25" maxColumns="10" scope="session" borderWidth="1"
  detailDataStyleSheet="/assets/sasads.css" upArrowImage="/assets/up_03b.gif"
  rightArrowImage="/assets/right_03b.gif" leftArrowImage="/assets/left_03b.gif">
  <sasads:MDNavigationBar doubleLeftArrowImage="/assets/double_left_03b.gif"
    doubleRightArrowImage="/assets/double_right_03b.gif"
    leftArrowImage="/assets/left_03b.gif"
    rightArrowImage="/assets/right_03b.gif"
    disabledDoubleLeftArrowImage="/assets/double_left_03g.gif"
    disabledDoubleRightArrowImage="/assets/double_right_03g.gif"
    disabledLeftArrowImage="/assets/left_03g.gif"
    disabledRightArrowImage="/assets/right_03g.gif" />
</sasads:MDTable>

<sasads:MDBar id="MDBar1" model="MDModel1" commandProcessor="MDCommandProcessor1"
  imageLocation="/assets/" scope="session" />
</body>
</html>

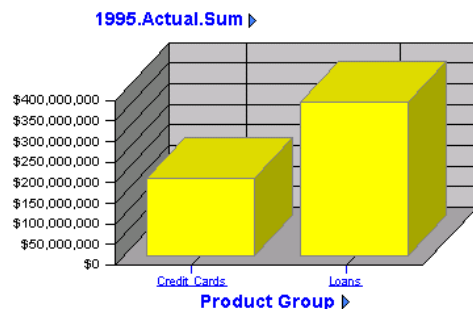
```

To test the JSP page from within webAF:

1. Make sure your web server is running by selecting Tools->Services->Start Java Web Server.
2. Select Build->Execute in Browser.

The resulting page should appear in the browser as shown.

Subset Export To Excel				
Year	1995			
Product Group	Actual Sum	Forecast Sum	Difference Sum	Actual Sum
Credit Cards	\$189,841,079	\$210,916,460	\$-21,075,381	\$193,941,557
Loans	\$375,606,799	\$278,075,052	\$97,531,747	\$383,780,698
Total	\$565,447,878	\$488,991,512	\$76,456,366	\$577,722,255



DEPLOYING YOUR OLAP APPLICATION USING SERVLETS AND JAVASERVER PAGES

In most Web project development scenarios, multiple roles and responsibilities will exist. For example, an individual who designs HTML pages fills the role of a Web designer, while someone who writes Java code might fill a software development role. With the Web technologies available in J2EE -- namely JSP and servlets - not only can you appropriately separate the development roles, you can also maintain a separation between your business logic and presentation code.

We have already demonstrated how you can use JSP technology for presentation purposes. Let's examine how one might separate some elements of the business logic from that presentation. To accomplish this, you can identify the components you need, then decide which technology to use to implement those components based on the role they play:

- Java servlets are well-suited to manage application flow and business logic evaluation. You can use servlets to provide a single point of entry into the Web application by performing such actions as intercepting HTTP requests that arrive from the client and simplifying security management.
- Presentation JavaServer Pages generate HTML (or other mark-up), and have as their main purpose the presentation of dynamic content.

The application can flow from the servlet to the presentation JSPs because the servlet can simply delegate or "forward" the incoming request -- plus any additional information returned by the business logic processing -- to the JSP. This separation, then, is analogous to the Model-View design pattern, where the front-end servlet functions as the model and the presentation JSP functions as the view. This approach is often presented as a "best practice" in implementing Web applications using J2EE.

Consider the OLAP application example, beginning with its Connection component. If you were to deploy the application within an organization that had many users, simple tasks such as connecting to SAS become much more costly because so many

users could be accessing the application at the same time, which increases the load on the SAS server. To remedy this, your team would likely employ a server load balancing technology such as AppDev Studio Middleware. Or, you might direct users with specific roles or from specific departments to other SAS servers. The logic to implement the Connection component under such constraints is no longer appropriate within the JSP page. It would require embedding conditional Java scriptlets within the JSP or adding a significant number of attributes to the Connection custom tag. Instead, you can instantiate the Connection component from within a servlet. You've now begun to separate your business logic from the presentation code.

When determining how to separate the components of your application, you can apply a simple test: Does this component contribute to the rendering of the response? If it does, then you can leave it in the JSP. Otherwise, it is appropriate to move it into the servlet. The MDModel in our OLAP application fits this criteria. For example, by coding the metabase and multidimensional database information directly on the JSP as attributes of a custom tag (as we did in the previous demonstration), you place a reference to key information about your organization within the presentation. In addition, this metabase information may be data-driven itself -- or, based on your business logic, you provide different metabases for different user roles or scenarios -- and the presentation simply displays it.

In general, it is best to leave a Web designer with the responsibility of coding attributes on tags that simply render information. Any information related to organizational data or an organizational process is more appropriately included in the non-visual servlet.

Some other things to consider:

- webAF supports the development of both servlets and JSPs from within a single project. You can use File > New and specify a HTTP Servlet if you want to add a servlet file to a project.
- To forward a request from a servlet to a JSP, you use the code shown at the top of the following page.

Example of forwarding request from a servlet to a JSP:

```
RequestDispatcher dispatcher=getServletContext().getRequestDispatcher("jsp-file.jsp");
dispatcher.forward(request, response);
```

where *jsp-file.jsp* is the name of the presentation JSP.

The remainder of this section illustrates how the example we've been building can more appropriately be separated into servlets and JSPs. We've separated the business logic and application flow components (Connection, MultidimensionalTableV3Interface, CommandProcessor) and placed them in a servlet. The presentation component (MDTable) is placed in the JSP. To complete the example, you would also need to add the MDNavigationBar, MDBar, and Menubar custom tags to the JSP since they also represent rendering of visuals that are part of the application.

Servlet Example:

```
/* Copyright (c) 2001 by SAS Institute Inc., Cary, NC 27513 */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.sas.rmi.Connection;
import com.sas.rmi.Rocf;
import com.sas.servlet.util.BoundConnection;
import com.sas.sasserver.mdtable.MultidimensionalTableV3Interface;
import com.sas.servlet.beans.mddbtable.html.*;
import com.sas.servlet.beans.mddbtable.MDCommandProcessor;
import com.sas.servlet.util.Util;
import com.sas.servlet.beans.mddbtable.commands.*;

public class servletSugi26
    extends javax.servlet.http.HttpServlet
{
    /**
     * Respond to the Post message.
     */
    public void doPost(javax.servlet.http.HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response)
        throws javax.servlet.ServletException, java.io.IOException
    {
        //Get the session object for this user
        HttpSession session = request.getSession();
        if (session != null)
        {
            //Declare the Objects that will have Session Scope
            MultidimensionalTableV3Interface MDModel=null;
            MDCommandProcessor MDCommandProcessor1;

            //Get the connection object
            Connection olapConnection = (Connection)session.getValue("olapConnection");

            //If olapConnection is null then the user has not connected to the server.
            //Establish a connection and create the objects needed for application and put
            //the objects in the users session so they can be retrieved by future requests.
            if (olapConnection==null)
            {
                //Create the Connection Object, set the host to localhost, and
                //store it as a session object.
                olapConnection = new Connection();
                olapConnection.setHost("localhost");
                session.putValue("olapConnection",olapConnection);

                //Make this a bound connection so the SAS session will be
                //terminated when the http session times out.
                BoundConnection bc = new BoundConnection(olapConnection);
                session.putValue("bc",bc);

                //Create the Rocf Object and store it as a session Object
                Rocf rocf = new Rocf();
                session.putValue("rocf",rocf);

                //Create the MDCommandProcessor Object and store it in the session
            }
        }
    }
}
```

```

MDCCommandProcessor1 = new MDCCommandProcessor();
session.putValue("MDCCommandProcessor1",MDCCommandProcessor1);

//Create the MDModel Object.
MDModel =(MultidimensionalTableV3Interface) Util.newInstance(rocf,
                    olapConnection, MultidimensionalTableV3Interface.class);

if (MDModel != null)
{
    //Store the MDModel as a session object
    session.putValue("MDModel",MDModel);

    //Create String arrays to initialize the MDModel.
    String rows[]={"Product Hierarchy"};
    String cols[]={"Time Hierarchy"};
    String measures[]={"Actual","Forecast","Difference"};
    String stats[]={"SUM"};
    try
    {
        //Set up the MDModel
        MDModel.setMetabase("sugidemo");
        MDModel.setDatabase("SUGIDEMO.CBMDDDB");
        MDModel.setRowAxis(rows);
        MDModel.setColumnAxis(cols);
        MDModel.setSelectedMeasures(measures);
        MDModel.setSelectedStatistics("Actual",stats);
        MDModel.setSelectedStatistics("Forecast",stats);
        MDModel.setSelectedStatistics("Difference",stats);
    }
    catch(com.sas.table.TableException te)
    {
        //If there is a problem, forward to an error page.
        RequestDispatcher errPage =
        getServletContext().getRequestDispatcher("/error.jsp");
        errPage.forward(request, response);
    }
}
else
{
    //If the Connection has already been made retrieve the session objects.
    MDModel = (MultidimensionalTableV3Interface)session.getValue("MDModel");
    MDCCommandProcessor1 =
        (MDCCommandProcessor)session.getValue("MDCCommandProcessor1");

    RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher("/{ProjName}/presentation.jsp");
    dispatcher.forward(request, response);
}
}

/**
 * Respond to the Get message.
 */
public void doGet(javax.servlet.http.HttpServletRequest request,
    javax.servlet.http.HttpServletResponse response)
    throws javax.servlet.ServletException, java.io.IOException
{
    // Note: Add User DO_GET code here
    doPost(request,response);
}
}

```

The request gets forwarded to the JSP.

JSP EXAMPLE:

```

<!-- Copyright (c) 2001 by SAS Institute Inc., Cary, NC 27513 -->
<%taglib uri="http://www.sas.com/taglib/sasads" prefix="sasads"%>

<html>

```

```
<head>
<title>OLAP Presentation JSP</title>
<!-- Include sasads style sheet for default style classes -->
<link rel="stylesheet" type="text/css" href="/assets/sasads.css">
</head>
<body>

<h2>OLAP Presentation Example</h2>

<!--Create the MDCommandProcessor, Connection and MDModel -->
<!--Locate the MDCommandProcessor tag at the top of the page-->
<!--because the execution of other tags depend on the executed command-->
<sasads:MDCommandProcessor scope="session" id="MDCommandProcessor1" />
<sasads:Connection ref="olapConnection" scope="session" />
<sasads:MDModel ref="MDModel" scope="session" />

<!--Create and write out the MDTable -->
<sasads:MDTable id="MDTable1" borderWidth="1" cellSpacing="0"
    maxColumns="8" maxRows="20" cellPadding="1"
    detailDataStyleSheet="/assets/sasads.css" commandProcessor="MDCommandProcessor1"
    scope="session" model="MDModel" />

</body>
</html>
```

CONCLUSION

AppDev Studio gives application developers a simple way to build OLAP applications that leverage the broad power of SAS on the server. It delivers powerful, easy-to-use OLAP reporting capabilities through

- WebEIS software, a rich 100% Java-based application, which enables you to build your OLAP applications without having to write any Java code. Documents built with webEIS can be easily deployed as applets or JSPs by simply making a choice off of a menu!
- webAF software which is a stand-alone, integrated Java development environment. Through its drag-and-drop interface you can quickly build applications, applets, servlets or JSPs. webAF also provides a rich class library that includes the foundation components for building OLAP solutions. These components are dedicated to delivering OLAP data on the web and utilizing SAS on the back end for both computation power and data storage.

By offering this all of this technology under one umbrella, AppDev Studio gives you the tools to build the right solution for your enterprise.

ADDITIONAL RESOURCES AVAILABLE

AppDev Studio Developer's Web Site

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 your local Web server when you install AppDev Studio. You can always access the most current version of this site at www.sas.com/rnd/appdev/.

Training

SAS Institute offers a broad curriculum of instructor-based courses to help you use SAS software to meet your development goals with AppDev Studio. Courses cover a wide range of Web applications development, including:

- *SAS Web Tools: Accessing MDDB Data Using webEIS Software*
- *SAS Web Tools: Developing JSP Applications Using webAF Software*
- *SAS Web Tools: Understanding Java in webAF Applications*
- *SAS Web Tools: Overview of SAS Web Technology*
- *SAS Web Tools: Running SAS Applications on the Web*

Instructor-based training allows you the flexibility to attend courses in training facilities across the United States and in other countries. In addition, SAS staff can conduct on-site training. For more information on these and other courses, visit the SAS Training site at www.sas.com/training.

REFERENCES

SAS Institute Inc. (2000), *Getting Started with AppDev Studio, First Edition*, Cary, NC: SAS Institute Inc.

OLAP Tools and Techniques within the SAS System, A SAS White Paper written by John McIntyre, Mark Moorman, and Johnny Williams, SAS Institute Inc.

ACKNOWLEDGMENTS

Much of the content of this paper was pulled from various papers that are available on the AppDev Studio Developer's Site. We tried to give credit in the appropriate sections for those contributions, but we would like to thank all of the authors for the valuable information and useful examples that are available from that web site.

We'd like to personally thank several SAS developers for either review and/or writing of material for this paper: Angela Allen, Corey Benson, Scott Leslie, Rich Main and Marty Tomasi.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. You can contact the authors at:

Tammy Gagliano
SAS Institute Inc.
Work Phone: (317) 569-9598
Email: Tammy.Gagliano@sas.com

Tony Prier
SAS Institute Inc.
Work Phone: (913) 491-1166, x1364
Email: Tony.Prier@sas.com