Paper 173-25

Migrating Your SAS/AF® FRAME Applications to the Web

Carl LaChapelle, SAS Institute Inc., Cary, NC
Tammy L. Gagliano, SAS Institute Inc, Chicago, IL

ABSTRACT

As the Web continues to gain in popularity for delivering critical information and offering better customer service, more and more businesses are exploring ways to exploit this technology to give them a competitive edge. Not only are new applications being designed for Web deployment but existing applications are being considered for deployment as well. Given that, just how easy is it to move your existing SAS/AF® application to the Web? Although the question may seem simple, there is no simple answer. Instead, it will take

- an understanding of what technology is available to help you accomplish this task
- a careful examination of the underlying architecture of your existing application to determine how much of that application can be reused versus rebuilt.

This paper will attempt to shed some light on both of these areas. While this paper focuses on solving these problems using Java™ that is not meant to imply that Java is always the best choice for every Web application. There have been numerous papers and documentation written on how to use some of the other technologies that the SAS® System offers. This paper will try to point you to those resources instead of duplicating that information here.

INTRODUCTION

The Web can be a bit intimidating at first, with all the talk about Web browsers and servers, application servers and middleware. Somewhere along the line, you'll also hear about HTML, CGI-based or Java-based technology, and wonder which one you should be using.

Each of the above topics could easily lend themselves to SUGI papers in and of themselves, if not entire books. The first portion of this paper provides an overview of the web-enabling technologies SAS has to offer which includes

- 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/InterNet™ software and are based on CGI technology
- Java-based technology such as servlets, JavaServer™ Pages (JSP), applets or applications.

Along with making a decision on which technology (or combination of technologies) to use, you also need to look at your existing application and how it was designed. You may find that it has not been engineered under the client/server architecture that web applications require. We will discuss what this means and how it affects the reuse of your existing application versus having to rebuild from scratch.

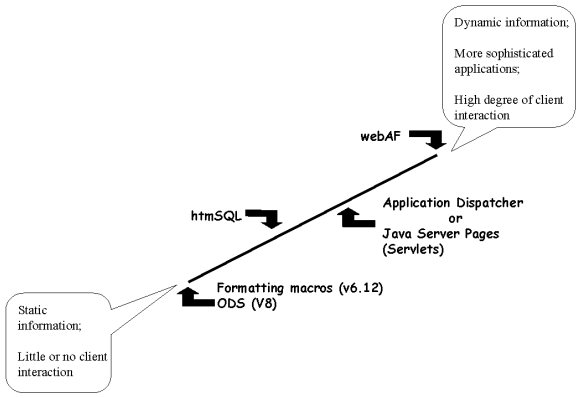
After you have chosen the technology and you are ready to begin your development, you need a solution that will enable you to take advantage of the SAS System on the server for its strength in decision support, data visualization, data mining, statistical analysis, and reporting. AppDev Studio™ (ADS) is that solution. As a complete, stand-alone development environment, this integrated suite of development tools 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.

Using webAF™ software, which is one of the products bundled with AppDev Studio, we will discuss

- how you can reuse models on the server written with SAS/AF software from within a Java client application
- the components that are available for you to use to rebuild those pieces of your application that are more appropriately placed on the client side as well as those that need to communicate back to the remote SAS server.

OVERVIEW OF WEB-ENABLING TECHNOLOGIES

The following graphic, taken from *Getting Started with AppDev Studio, First Edition*, portrays a good visual representation of the wide range of technologies that you can use. The vertical axis represents the state of the data. As your need for dynamic content generation increases, the appropriate technology or tool also changes. Likewise, the horizontal axis represents the client interactions with the data. As the need for increased client interaction grows, data manipulation can move from the server to the
client.



The following tables detail the technologies used with each available AppDev Studio component, as well as an overview of the programming skills that you need to use each component.

AppDev Studio Component	Technology		
	CGI	Java	SAS
Formatting Macros			X
ODS			X
Application Dispatcher	X		X
htmSQL	X		
webAF		X[1]	X[2]

[1] Can be used to create Java applets, applications, servlets, or JSP. [2] Required when building Java clients that access SAS on the remote server.

Your decision as to which technology to use may be based on the skills of your development team.

AppDev Studio Component	Programming Skills			
	Java	SQL	HTML	SAS Languages
Formatting Macros				Macro
ODS				SAS procedures and the DATA Step
Application Dispatcher			X	DATA Step, Macro, or SCL
htmSQL		X	X	
webAF	X		X[3]	DATA Step, Macro, and/or SCL

[3] HTML is not required if you are creating Java applets or applications and is required only if you need to modify the HTML that is generated by webAF for applets.

Subtle differences may determine which technology you choose to use in a given situation. An overview of each technology is provided below along with comparisons as to which technology might be a better fit for a given set of needs.

WEB PUBLISHING TOOLS

Many users require access to corporate information without needing to interact substantially with that information. Web publishing tools allow these users to receive information generated in SAS software as HTML-formatted pages through a Web browser. Another term for this type of application is static reporting.

The advantage that both ODS and the formatting macros have to offer is that you can leverage your current SAS software program investment without having to add additional products. The macros and/or SAS programs execute on the SAS server and automatically generate an HTML page that corresponds to your output. You can place the output from these tools on a Web server where they can be viewed by anyone with a Web browser.

You may find that creating SAS output and making it available on a web site may meet your reporting needs. You may also combine the server-side formatting capabilities of the macros or ODS with servlet or JSP technology to directly stream the results back to a client browser via a TCP/IP socket. This approach can not only leverage existing back-end processes, but also improve performance by preventing any output from being written to disk. For an example of this, see *Getting Started with AppDev Studio,* Chapter 2, *Java Technologies for the Server.*

Output Delivery System (ODS)

If your SAS/AF application does nothing more than generate output from SAS procedures and display that output for the user, you may want to consider using ODS to generate the output for you. Available in base SAS, ODS offers an easy way to produce SAS output in HTML. It enables you to produce output in a variety of formats and can be used in batch mode or interactively. ODS can save the results to an HTML file that can then be moved to your Web server. All you need to do to put the finishing touches on your application is to create a HTML page that lets them pick from the list of pre-generated reports.

ODS enhances your ability to manage both DATA step and procedure output. It enables you to manage both DATA step and procedure output and features the ability to

- combine the raw data that is produced with one or more templates to produce one or more output objects that contain the formatted results
- generate HTML files that contain the formatted results and that contain links to the results in the form of a table of contents
- generate output data sets from procedure output
- allow you to customize the procedure output by creating templates that you can use whenever you run the procedure.

For more information on ODS, see the Version 8 SAS System online help or the SAS Online Documentation CD or visit our web site at http://www.sas.com/rnd/base.

HTML Formatting Tools

These are a set of pre-written SAS macros that were made available with Version 6.12 of base SAS, before ODS came to life. They let you generate SAS output in HTML format without knowing how to code HTML. Similar to ODS, the formatting macros produce static HTML output. The macro must be run again to access any changes to the data, which means there is no dynamic support. HTML output must be manually moved to the Web server. Macros are available for converting specific items, including SAS output to HTML, SAS data sets and views to HTML tables, and PROC TABULATE output to an HTML table. The macros support cascading style sheets, dynamic HTML, and options for formatting and saving templates.

ODS may be preferable to using these macros because it offers much more flexibility and control over the results of the generated HTML. If, however, you find that you only have access to Version 6.12 of the SAS System, these macros provide a simple way to generate HTML pages from SAS output. For more information on the HTML formatting tools, see http://www.sas.com/rnd/web/format/index.html.

CGI-BASED DEVELOPMENT

Many powerful web applications are built today using Common Gateway Interface (CGI) technology, which is a programming interface that enables a Web server to communicate with an external program. CGI programs are generally small in size, written in a script or high-level language and supported by all Web Servers. The CGI program resides on the Web server to act as the interface between the Web browser and the application server (sometimes also referred to as the content server).

When a Web browser accesses the Uniform Resource Locator (URL) of a CGI program,

1. the CGI program executes.
2. the CGI program typically starts a session with a database or application server (which in our case would be a SAS session on the application server).
3. the application server then processes the information. It is responsible for generating the HTML content that gets returned to the browser by way of the Web server.

SAS/IntrNet software provides two programming tools that are based on CGI technology: the Application Dispatcher and htmSQL. It is important to note that for the purpose of this paper, SAS/IntrNet software is discussed primarily as the provider of our CGI-based technology. It is in fact a whole lot more; it is the core of SAS Institute's Web exploitation strategy offering other key pieces such as the SAS/SHARE® driver for JDBC and the SAS/CONNECT® driver. Other products in AppDev Studio - specifically our Java

technology use these drivers to access data stored in both SAS and non-SAS formats as well as to utilize the compute power of the SAS System on a remote server. For the most up-to-date information on SAS/IntrNet software, see http://www.sas.com/software/components/intrnet.html.

Application Dispatcher.

The Application Dispatcher is a set of components that enables you to use the Web browser to run any SAS program that can be executed in batch mode. The components of the Application Dispatcher are

- Application Broker, a CGI program that resides on the Web server. The Application Broker passes user input from the browser to the application server.
- Application Server, a continuously running SAS session. It can be located on the same machine as the Web server or on a different machine accessible to the Web server.
- Load Manager, an optional and separate process that can be used to enhance the distribution of Application Dispatcher resources on a network (i.e. load balancing to optimize response time)

A new copy of the Application Broker is started each time a client sends a request. The request is routed by the Load Manager to an available Application Server (you can have a pool of Application Servers). The Application Server runs a SAS or SCL program that is associated with the request and then returns the results back to the client.

When developing an application using this technology, you develop the user interface portion of the application, typically written in HTML. When executed, that HTML page passes information to the Broker – one of the pieces of information being the name of the program to be executed on the application server. This program can be a SAS or SCL program that generates the resulting HTML to be returned to the Web browser through the Application Dispatcher on the Web Server.

The Application Dispatcher does not require any CGI programming skills - it handles all the details relating to CGI, the communication of parameter values from the HTML page, and the returning of results to the users' browsers.

Somewhat advanced SAS programming skills are required when writing the portion of the application that resides on the application server. Solid experience in DATA step, macro and/or SAS Component Language (SCL) is required. Experience with designing and developing HTML forms and files is also recommended.

CGI technology can be used to solve almost any problem. The problem that some users face after deploying CGI applications is response time. The response of a CGI program depends on how much data must be sent as well as the load on both the server and the Internet. On top of this, starting a CGI program can sometimes be slow.

The use of the Load Manager available with the Application Dispatcher can help to improve this bottleneck, however, the basic nature of pure CGI-based applications is that they require repeated interaction with servers in order to provide interactivity with the user. As an example, picture a web page that does something as simple as validating the data on an input form.

1. You enter data and press the submit button on the page.
2. The data is sent back to the server.
3. The server starts a CGI program and discovers an error. It formats an HTML page informing you of the error and sends the page back.
4. You have to use the browser's Back button to go back and try again.

While this process works, it is not very elegant and can be slow. A better solution to this type of application might be to use Java servlets which offer better performance and state management or even a Java applet where simple things like field validation can be performed on the client side, thus reducing network activity for simple tasks.

htmSQL

htmSQL is a markup language that allows the embedding of Structured Query Language (SQL) queries in a Web page. SQL is a standardized and widely used language for retrieving and updating data in relational tables and databases. The SQL procedure is SAS software's implementation of SQL.

When using htmSQL, the user interface is a Web page. The page can be as simple or as sophisticated as you want, and you can use any HTML element that your browser supports. You can embed htmSQL-specific tags and processing instructions called "directives" that enable processing of any number of SQL statements on a single page. Other directives can be used to embed results anywhere on a page.

When a user accesses or submits a page that contains htmSQL directives, htmSQL passes your SQL to a SAS/SHARE server, performs the requested updates and queries, and retrieves the result sets. The desired page is created dynamically and returned through the Web server to the browser.

A separate copy of htmSQL is run on the Web server for each request that a Web client sends. On the client side, the only software required is a Web browser; all of the processing is done by htmSQL on the Web server.

The htmSQL solution does not require any CGI programming. You do, however, need to know how to design an HTML page and understand how to use the htmSQL directives to construct standard SQL queries.

The obvious limitation to choosing this technology is that it assumes your application requirements can be accomplished through SQL, which may or may not be true. If true, this approach can be a good performer as the data access is finely tuned with direct calls being made to the underlying data source.

Scripting Languages

Before we discuss the different solutions available with Java, we need to touch on the various scripting languages out there that you might have heard about. A scripting language can be embedded in HTML to provide conditional logic and more control over the interactivity of the Web page than just using straight HTML. Because they are part of the page, they load very quickly with the single server hit required to retrieve the page. Some of the more popular ones are

- VBScript™ which looks like Visual Basic®. If you are more familiar with Visual Basic and your users will be using only Microsoft's Internet Explorer as their browser, you might choose to use this one.
- Both Internet Explorer and non-Microsoft browsers support JavaScript, which has no relationship with the Java language other than the first four letters of its name. JavaScript has since been renamed ECMAScript but most likely you will still hear people refer to it by its initial name.

Many people turn to scripting languages to be used in conjunction with other solutions mentioned earlier such as CGI or Web Publishing tools before jumping on the Java bandwagon. One reason for this is because they tend to be fairly simple languages to learn. It's the old 80/20 rule. Scripting languages, in combination with some of the other solutions, may in fact be able to solve 80% of the application needs. But for the other 20%, the most popular choice today seems to be Java.

JAVA-BASED DEVELOPMENT

Why is Java considered by some to be the premier language of choice for providing highly interactive user interfaces to the Web browser? It is a powerful language that is continuously being enhanced to provide language 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) all 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.

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 servlets is also 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 tables and MDDBs, and access to SAS compute power through procedure submissions.

JavaBeans™

Another reason that Java is so popular is because of its JavaBeans and Enterprise JavaBeans architecture. This object-oriented framework allows 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.

Regardless of whether you're developing Java applets, applications or servlets, these InformationBeans can be used. They 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. See the section labeled *SAS/AF to webAF Component Specifics* for a list of some of the more commonly used webAF components.

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.

An applet consists of many Java classes. Usually, the more complex an applet is, the more classes it depends on. Applets can depend on hundreds, even thousands, of classes and each of these classes must be available to the browser in order to execute the applet. Thus, the question for the applet developer becomes "how do I make these classes available?" There are several options:

- allow them to be individually downloaded from the Web server on demand
- allow them to be collectively downloaded from the Web server, also on demand
- pre-install them on the client machine
- some combination of the above

There are pros and cons to each approach, so let's examine each in a little more detail.

Individually downloading classes on demand is the easiest in terms of set-up, but potentially the slowest in terms of execution. Classes are simply placed in the same Web server directory as the HTML file that launches the applet and are then accessible by any client machine that can access the HTML. The performance of this approach is dependent on the number of classes, as there will be one round-trip from the client machine to the Web server for each class.

The collective-download approach can improve performance by reducing the number of these round-trips by packaging sets of related classes into archive files (i.e. JAR files). In this way, a large number of classes can be downloaded from the Web server at once. The amount of information transferred is the same, but the time to do so is significantly reduced. However again, the performance of this approach will be a function of the number of classes needed.

Downloading on demand is elegant since no software has to be pre-installed on client machines, but another factor to consider here is that classes downloaded by a browser from a Web server are only available for the duration of the browser session. This is ok if download times are short or if individual usage of the applet is infrequent; but frequent, perhaps even daily, users of your applet are going to be less patient. In such cases, pre-installing classes on client machines is an option to consider.

Class pre-installation can eliminate the majority of applet download time. It can be done up-front or on demand. The up-front approach

is most suitable for a corporate Intranet where a system administrator can pre-install software on everyone's machine.

The on-demand approach is more flexible, as the classes do not have to be installed on a given machine until they are needed. So in this way, the applet user only experiences a significant download time the first time they use a given applet. All subsequent uses, even across browser sessions, will be very fast, as the majority of classes will already be available locally. This approach is also more dynamic, as a version number associated with the local classes can be quickly checked against a version on the Web server every time the applet is run, and thus class updates can automatically be applied. AppDev Studio provides a tool called SASNetCopy, which allows you to set up your applets using this latter approach. webAF and webEIS™ both provide a Packaging Wizard to assist you with this and the other deployment scenarios described above.

Java Servlets and JavaServer Pages™ (JSP)

At a conceptual level, servlets are just like applets except that they run in the server environment instead of the browser environment. 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 that make it simple to instantiate JavaBeans components and manipulate properties on a component from within your web page.

webAF's InformationBeans can be used from with in a JSP page in this manner. Along with these components, another set of components named TransformationBeans™ are available that make using this technology even easier.

The current set of TransformationBeans is listed in the table below. 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 bean, 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!

TransformationBeans	Description	HTML Output	JavaScript Output
Checkbox	Creates a checkbox input field	Yes	
Choicebox	Creates a drop-down list of valid selections	Yes	
Form	Creates an HTML form with optional client-side JavaScript validation code		
Hidden	Creates a hidden field	Yes	
Image	Creates a push button with an image	Yes	
Listbox	Creates a list of valid selections	Yes	
MDTable	Creates an HTML table that can display SAS MDDB data	Yes	
Password	Creates a text input field that echoes '*'	Yes	
PushButton	Creates a push button	Yes	
Radio	Creates a radio button input field	Yes	
Table	Creates an HTML table	Yes	
TextArea	Creates a text area input field	Yes	
Text	Creates a text input field	Yes	
TreeControl	Creates a tree control		Yes

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

For examples that use these TransformationBeans, see http://www.sas.com/rnd/appdev/webAF/server/examples.htm.

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.*

AppDev Studio's Middleware Server

AppDev Studio's Middleware Server adds value to applications that are based on applets or JSP technology. The ADS Middleware Server is designed to serve as a "funnel" point for all client requests to SAS and thus enable multiple clients to share a single SAS session. You can configure the ADS Middleware Server in a number of different schemes. For example, the first time a client requests a SAS session from the ADS Middleware Server, a new SAS session is created. Any subsequent clients will share the same SAS session, each being partitioned into a new task space within the existing session. Load balancing schemes control how clients are assigned to SAS sessions.

Sharing SAS sessions has distinct advantages:

- Reduced memory consumption on the server due to a reduced number of SAS sessions required to service client requests.
- Faster client startup. For clients that are sharing an existing SAS session, the startup time is greatly reduced because the server does not have to create a new SAS session. Instead, only a new task space within the existing session is needed.

For more information as to whether your application would benefit from using the ADS Middleware Server, see http://www.sas.com/rnd/appdev/doc/MiddlewareServer.htm.

CHOOSING THE BEST TECHNOLOGY

A SUGI24 paper, *SAS/IntrNet Software: A Road Map*, offered some good tips that are extremely relevant to this discussion and are included in this section.

If time to deliver is the most critical factor in deciding what technology to use, you have to take a serious look at your expertise level in the above technologies.

- First off, take some time to learn HTML if you haven't already. It is fairly easy to learn and you can learn quite a lot about it in just a few days. A minimal investment in HTML training can greatly expand your options.
- If you have limited or no knowledge of CGI, HTML, Java, and other Web technologies or if you are skilled mainly in SAS programming, then first consider using the Application Dispatcher.
- If your staff is skilled in SQL and HTML programming, then first consider htmSQL.
- JavaScript and Java have a higher learning curve; however, programmers with experience in any object-oriented programming language should have an easier time with these languages than those without object-oriented experience. The interactivity and flexibility offered by both JavaScript and Java can make the investment in learning them (or hiring staff who already know them) worthwhile.

Obviously the above cannot be your exclusive decision making point since the language you choose to web-enable your application has a direct impact on the features you have at your disposal. You must also consider the needs of the application. For example, if the application has nothing to do with generating SQL type query reports, then htmSQL is not a viable option regardless of your skill set. Similarly, just because the application involves some dynamic interaction by the user does not mean that Java is the best fit for the application if another, simpler technology can perform as needed.

As mentioned from the start of this paper, there are no easy answers as to which technology you should use for any given application. Hopefully, the contents of this paper so far have offered some suggestions or at least a basic understanding of each with some guidelines on how they can best be put to use.

For the remainder of this paper, we're going to discuss specifics on migrating your SAS/AF application with examples showing how to use the Java tools available with AppDev Studio to help you get the job done!

APPLICATION ARCHITECTURE

Because SAS/AF software is a full-featured application development tool supported by its own comprehensive programming language, SCL, it would be impossible to offer a tool that automatically converts your full-client application to a thin-client one that's ready to go on the Web. That does not mean, necessarily, that you will have to start from scratch when building your new web-based version. How much of an existing SAS/AF application you can reuse depends primarily on the underlying architecture used to build the existing application.

We can start with two basic questions:

- Did you take advantage of SAS/AF's object-oriented nature and separate the visual pieces of your application from the portions that generate the content by defining your own classes or subclassing the ones provided with SAS/AF software?
- Or, did you design it primarily using FRAME SCL where the logic that controls the content of the visuals in your frame is intermingled with the logic that controls the behavior of the application or the user-interface?

If it's the former, then there are pieces that you may be able to reuse. But if you answer yes to the last question, then you will need to spend time mentally taking a part the pieces of your SAS/AF application. Specifically, you need to analyze what functionality is better suited for client-side processing versus what pieces should remain on the server side. This thought process aligns itself more closely with the *n*-tier architecture that the Web requires, which may be quite different from what you are used to.

Hopefully, the following topics will ease any fears you may have towards making this transition. Using webAF's interactive development environment and powerful Java component library, building web-based applications is a snap! And in some cases, without having to write a single line of Java code.

USING REMOTE SCL MODELS FROM A JAVA CLIENT

One of the most powerful features that webAF offers its ability to communicate with a remote SAS server. It is this functionality that makes it possible for you to reuse an existing model (i.e. non-visual class) written with SAS/AF software from within a Java client application. Because your SCL model has access to SAS, you can exploit virtually any functionality that the SAS System offers on that remote server. You can access any of the data sources that SAS can access as well as take advantage of the server's compute power to perform any task that the SAS System provides.

The way this works is simple.

1. You can use the InformationBean Wizard in webAF to create a Java interface that describes the model on the SAS/AF side. An interface is nothing more than a kind of contract between two objects - an agreement as to what methods are available for the client-side object to call on the server-side object. An interface can expose all or just some of the methods that exist on the server model.

   At the same time, the wizard creates what is called a remote proxy for this new interface. The remote proxy contains the Java code necessary to communicate between the client and the server. You do not need to know how to start a SAS session, instantiate the remote model, marshal data back and

forth or be concerned about communication protocols. The remote proxy that is created handles this for you.

The wizard steps you through this entire process. The end result is that a Java interface and a remote proxy is created automatically for you without you having to write any Java code.

2. You are now ready to begin using the new interface in your webAF application. The InformationBean wizard again makes this simple by asking you if you want it to add an instance of the remote interface to your project for you. When a remote object is added to a webAF project, a new Connection object is added as well. The Connection object is used to specify the properties necessary to create and/or connect to a SAS session. You supply information such as the host name, the SAS command to use to start SAS, various configuration options, etc.

   After completing the steps above, an instance of a proxy object which implements the remote interface will show up on the component view tab of the Project Navigator window which is where you manage the objects that are available in your current project.

   You can begin writing Java code and invoking methods on the remote object directly or attach your remote object as a model to a viewer component that knows how to communicate with your interface in a model/view relationship.

For more of a "how to" illustration on the above process, refer to the SUGI24 paper *Distributing SAS/AF Models with Java Clients*.

webAF already provides several interfaces and remote proxies that communicate with a SAS server to perform some of the more common tasks that you will want to do.

com.sas.sasserver.dataset.DataSetInterface
: defines methods for setting and retrieving formatted or unformatted data from a SAS data set. It supports where clause subsetting and SCL source. This interface uses the SCL DATA_M class to interact with the SAS data set.

com.sas.sasserver.MultidimensionalTableV2Interface
: defines methods for setting and retrieving formatted or unformatted data from a SAS MDDB. It supports subsetting , totals, computed values, exception highlighting (traffic lighting), sorting and more. This interface uses the SCL MDTABLE class to interact with the SAS MDDB.

com.sas.sasserver.librarylist.LibraryListInterface
: defines methods for retrieving the libraries in a SAS session, as well as specifying filters for the library list. It also supports assignment and deassignment of libraries.

com.sas.sasserver.sasfilelist.SASFileListInterface
: defines an interface to retrieve a list of SAS library members, referred to as SAS files, from a SAS session. It references SAS files of all types (e.g., files of type DATA, ACCESS, VIEW, and CATALOG).

com.sas.sasserver.catalogentrylist.CatalogEntryListInterface
: defines an interface to retrieve a list of SAS catalog entries, or SAS catalog members, from a SAS session.

com.sas.sasserver.sasfilelist.DataSetListInterface
: defines an interface to retrieve a list of SAS data sets from a SAS session.

com.sas.sasserver.dataset.DataSetInfoInterface
: defines methods for returning information about columns in a data set, such as column names, labels, formats, informats, class, length, and unique values. It does not actually return the values of the observations in the data set, or provide editing ability. For this type of functionality, the DataSetInterface class should be used.

An example of using the DataSetInterface would be in an application where you want to display a remote SAS data set in a table in the browser. Using webAF to build an applet, you would

1. Create the visual control that you want to use to display the data. You can create objects easily using the Component Palette in webAF. The first icon on the Data Viewers tab represents the TableView component, which knows how to display two- dimensional data. Create an instance of this component by dragging from this icon and dropping it onto the frame that's open in your project.
2. Next, we need to create an instance of the DataSetInterface model and connect it to the table so that the table will display the contents of whatever data set we point the model to. From the SAS tab, drag the DataSetInterface icon and drop it on top of the TableView component in your frame.
3. The Remote Connection dialog will display asking you if you want to create a remote connection to SAS. In this window, you can create a new connection or use a predefined connection to get to a remote SAS server. A Connection object will be created for you and is necessary for your application since you will need to connect to SAS on a remote server to get the data.
4. The final step is to tell the DataSetInterface model which remote data set you want to use. You can do this several ways but a quick way is to open the Customizer for the TableView object. Select the Data Set tab and specify the SAS data set you want to use. If you use the ellipsis button to the right of the Data set name field, webAF will establish a connection to the remote server automatically for you and display a dialog that let's you choose from the available SAS data sets on the server.
5. To test your application, select Build from the menu and then one of the Execute options. You can see your applet in the Applet Viewer supplied with webAF or in the chosen browser.

The above applet was built without typing a single line of Java code. The required Java code was generated for you automatically as you interacted with the build environment and created your frame. The relationship established between the TableView and DataSetInterface object is referred to as model/view.

<u>Model/View Communication</u>

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. In the example above, the TableView component can display information from a SAS data set because the DataSetInterface model implements the interface, which the TableView requires to communicate with its model. The viewer can call any of the methods defined in the interface and is guaranteed that the model has implemented these methods to perform the necessary functionality. As a result, the table knows how to work with its attached model without you having to write Java code to perform tasks such as retrieving the rows to display, handling updates that might be made through client-side table interaction and more.

If you've been using Version 7 or higher of SAS/AF software, this process is almost identical to establishing model/view relationships between SAS/AF components. You define the methods in an interface using the new Interface Editor. This information gets stored in a new catalog entry type named INTRFACE. Using the Class Editor, you then register on the model component that it *supports* the interface and you must also implement all of the methods defined in the interface. On the components that are to function as viewers for these models, you register that they *require* the interface. There are many model/view relationships that can be established using the standard set of components that come with SAS/AF software. Refer to the SUGI24 paper, *SAS Component Object Model (SCOM) in Version 7 of SAS/AF Software* for more

details on this subject.

If you've been using Version 6, model/view communication existed, however, it was not as visible to you. An example of where model/view was used is with the Data Table and Data Form objects. These objects were created to save you the process of having to physically create a viewer and then a model and write the necessary SCL to handle the communication between the two. The viewer used behind-the-scenes is either the Table Editor or Form Editor objects. The model automatically attached to the viewer is the Data Set Data Model class (or DATA_M). In Version 6, we did not have a separate catalog entry where the interface was defined. It was an 'implied' contract between the two at that point. In Version 7, we were able to add true interface support through the INTRFACE entry. And in Version 8.1, new Table Viewer and Form Viewer components have been added that can be attached to either a SASDataSet or SCLList model for true model/view communication.

Within webAF, there are many components that are enabled for this type of communication. For example, a ListBox component can use a LibraryListInterface component as its model to automatically display a list of libraries available on the remote server. You can just as easily display a list of SAS data sets using the DataSetListInterface in a ComboBox control.

JSP technology also takes advantage of this model/view separation but in a slightly different way. The same model can be used but with a different type of viewer. The viewer, in this case, would be one of the TransformationBeans. For the same example outlined above, you'd attach the DataSetInterface object to the Table TransformationBean from within the Java code in your .jsp program.

```
<html>
<head>
   <title>Table Bean</title>
</head>

<body>
<%
   // Setup the connection to SAS
   com.sas.rmi.Rocf rocf = new
      com.sas.rmi.Rocf();
   com.sas.rmi.Connection connection = new
      com.sas.rmi.Connection();
   connection.setHost("localhost");

   // Create a new DataSetInterface model.
   com.sas.sasserver.dataset.DataSetInterface
      dataSet =
   (com.sas.sasserver.dataset.DataSetInterface)
   com.sas.servlet.util.Util.newInstance
   (rocf, connection,
   com.sas.sasserver.dataset.DataSetInterface.clas
   s);

   // Set the value for the dataset to display
   dataSet.setDataSet("SASUSER.HOUSES");

   // Create a new table object
   com.sas.servlet.beans.TableInterface
        table = new
           com.sas.servlet.beans.html.Table();

   // Set the model
   table.setModelInterface(dataSet);

   // Output the table
   table.write(out);

   // Close the table and the SAS connection
      dataSet.close();
      rocf.stop();
%>
</body>
</html>
```

Note: The above code may not execute exactly as printed due to some line wrapping that occurred to fit the format of this paper.

The example uses a connection to SAS so that the DataSetInterface object has a means to access SAS data sets. The DataSetInterface has a setDataSet() method that is set to SASUSER.HOUSES. This interface is the model through which the table bean can be populated. The table bean's write() method simply outputs the HTML containing the table information.

SAS/AF TO WEBAF COMPONENT SPECIFICS

If you've used SAS/AF software, Version 7 or higher to build your frame applications, working with webAF components will be similar to how you're used to working with SAS/AF components. Instead of using SCL to write your programs, you use Java. A component has a set of properties that you can access. It has

- Properties (referred to as attributes in SAS/AF software) that you can set through the Properties window or a Customizer window in webAF
- Methods that you can use to set these properties and control its behavior
- Events that you can trap in an event handler method that you define
- A set of required or supported interfaces that it supports

The following table lists the more commonly used SAS/AF visual components and which component(s) offer similar functionality when building webAF applets or applications:

SAS/AF Component	webAF Component
Catalog Entry Viewer	Not available
Chart Control	com.sas.graphics.chart.Bar, com.sas.graphics.chart.SegmentedBar, com.sas.graphics.chart.Combination
Check Box Control	com.sas.awt.Checkbox, com.sas.visuals.GraphicalCheckbox
Combo Box Control	com.sas.visuals.ComboBox, com.sas.awt.Choice
Container Box Control	com.sas.visuals.BorderedPanel, com.sas.visuals.BorderedContainer, com.sas.awt.Component, com.sas.awt.Canvas,
Critical Success Factor Control	com.sas.graphics.RangeView
Desktop Icon Control	com.sas.visuals.PushButton
Dual Selector Control (Experimental)	com.sas.visuals.DualSelector
External File Viewer Control	Not available
Form Viewer Control	Not available
Graph Output Control	Not available
Graphic Text Control	com.sas.visuals.LabelView, com.sas.awt.Label, com.sas.visuals.Marquee
Histogram Control	com.sas.graphics.chart.Combination
Image Viewer Control	com.sas.visuals.ImageView, com.sas.visuals.ImageAnimation
List Box Control	com.sas.awt.ListBox, com.sas.visuals.UpDownListBox
Map Control	com.sas.graphics.chart.bean.MapChart

Pie Control	com.sas.graphics.chart.Pie
Progress Bar Control	Not available
Push Button Control	com.sas.visuals.PushButton, com.sas.awt.Button, com.sas.visuals.ToggleButton
Radio Box Control	com.sas.visuals.RadioBox, com.sas.visuals.RadioButton, com.sas.visuals.SelectionGroup
Scatter Control	com.sas.graphics.chart.Scatter
Scrollbar Control	com.sas.awt.Scrollbar, com.sas.visuals.Scrollbar
Spin Box Control	com.sas.visuals.SpinBox, com.sas.visuals.SpinButton
Tab Layout Object	com.sas.visuals.TabbedView
Table Editor (V6)	com.sas.table.TableView, com.sas.mdtable,MultidimensionalTableView
Table Viewer Control (Version 8.1)	com.sas.table.TableView
Text Entry Control	com.sas.awt.TextField
Text Label Control	com.sas.awt.Label, com.sas.visuals.LabelView
Text Pad Control	com.sas.awt.TextArea
Tree View Control (Experimental)	com.sas.visuals.TreeView

Just like with SAS/AF software, there are non-visual components you can use within webAF as well. These components can be used from within a Java applet or JavaServer Page.

SAS/AF Component	webAF component
Catalog Entry List Model	com.sas.sasserver.catalogentrylist.CatalogEntryListInterface
Catalog List Model	com.sas.sasserver.sasfilelist.SASFileListInterface
Color List Model	com.sas.models.DefaultColorList
Data Set List Model	com.sas.sasserver.sasfilelist.DataSetListInterface
External File List Model	com.sas.io.FileList
Library List Model	com.sas.sasserver.librarylist.LibraryListInterface
LIST Entry List Model	com.sas.sasserver.catalogentrylist.CatalogEntryListInterface
Range Model	com.sas.models.RangeCollection
SAS Data Set Model	com.sas.sasserver.dataset.DataSetInterface
SCL List Model	Some of the functionality provided in com.sas.collection.hlist.HListInterface
SLIST Entry List Model	com.sas.sasserver.catalogentrylist.CatalogEntryListInterface
Variable List Model	com.sas.sasserver.dataset.DataSetInfoInterface
Variable Values List Model	Not available

Additional webAF components that you will find useful:

Visual components	
com.sas.awt.ScrollPane	Similar to BorderedPanel container except it provides a scrollable container area
com.sas.table.NavigationBar	Selector for navigating through a data set
com.sas.apps.webEIS.viewer.bean.ReportViewer	Displays a webEIS report in a webAF applet/application
com.sas.visuals.Marquee	Marquee with text and/or image

Non-visual components	
com.sas.io.PrinterList	List of system printers
com.sas.models.DefaultFontList	List of standard AWT fonts
com.sas.util.CurrentDate	Current date
com.sas.visuals.AutoSizing.GridLayout	Aligns components in a grid with automatic sizing based on the components preferred size
com.sas.models.SimpleTable	2D model for data on the client side that can be used with TableView
Page: 9 com.sas.sasserver.mdtable.MultidimensionalTableInterface, com.sas.sasserver.mdtable.MultidimensionalTableV2Interface	an interface to SAS multidimensional data on a remote server
Com.sas.collection.*	Collection classes for holding data. Can be used as models for many viewers such as the ListBox
Com.sas.sasserver.format.FormatInterface	Allows data to be formatted by the SAS server

Both SAS/AF and webAF software components support an object model that adheres to simple communication rules, which in turn make it easy to share information between components. Those communication rules being

- model/view, which was discussed earlier
- property linking (referred to as attribute linking in SAS/AF software new to Version 7), which enables a component to change one of its properties when the value of another property on the same or different component is changed
- events and event handlers.

The above can be done easily through the build-time environment within webAF without writing any Java code.

MDDB RELATED APPLICATIONS

This paper has focused primarily on webAF and the functionality it provides towards web-enabling your SAS/AF applications. If your application revolves primarily around displaying and interacting with MDDB data sources or if you have existing SAS/EIS applications that you want to deploy on the web, you should look into using the following solutions:

- the com.sas.sasserver.mdtable.MultidimensionalTableV2Interface model is a Java component available with webAF software and is designed specifically to read and manipulate MDDB data. The data supplied by this model can be displayed in a Java applet using the com.sas.table.TableView component or in JavaServer Pages using the MDTable TransformationBean. Both of these viewer components were designed to display MDDB in a table format and communicate with the above model through model/view communication.
- MDDB Report Viewer, which is a component of SAS/IntrNet software. The MDDB Report Viewer is a finished application based on CGI-technology, which enables you to interact with MDDB reports and graphs in a Web environment. For more information on the MDDB Report Viewer, see http://www.sas.com/rnd/web/mddbapp.html.
- webEIS, which is another product bundled with AppDev Studio. It is a Java application 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. You do not write any Java code, however, to create these applets. 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 an 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. webEIS lets you work interactively with your created document in an applet environment. Using this tool, you can navigate through your data using drilldown or expand/collapse, you can apply traffic lighting, add computed columns, specify a subset, and more. All of this is done interactively on the client in the Web browser while communicating back to a SAS server where the data is stored.

For more information on deploying SAS/EIS applications to the Web, refer to the SUGI24 paper, *Deploying Existing SAS/EIS and SAS/AF Software Applications to the Web*. This paper contains information mostly with respect to migrating SAS/EIS applications. It is also more focused on using CGI-based technology with SAS/IntrNet than with Java technology such as webEIS.

CONCLUSION

When moving an existing SAS/AF application to the Web, there are pieces that can easily be reused. This helps to protect the time and investment you have already made with existing applications. We also provide the tool that makes it easier for you to build the client pieces whether that is using CGI-based or Java-based technology. Most importantly, all of the solutions we provide allow you to easily connect to and communicate with a remote server running the SAS System. Once connected to SAS, the tasks your Web application can perform are virtually limitless!

After reading this paper and seeing all of the options you have available to you for web-enabled reporting or application development, you will hopefully see that SAS is truly the right choice for delivering strategic business information to your intranet or to the Internet.

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

The SAS Institute offers a broad curriculum of instructor-based courses to help you use SAS software to meet your development goals with AppDev Studio. Instructor-based training allows you the flexibility to attend courses in training facilities across the United States and in other countries. Check with your local SAS office for availability.

Courses cover a wide range of Web applications development, including:

- *SAS Web Tools: Overview of SAS Web Technology*
- *SAS Web Tools: Running SAS Applications on the Web*
- *SAS Web Tools: Static and Dynamic Solutions Using SAS/IntrNet Software*
- *SAS Web Tools: Understanding Java in webAF Applications*
- *SAS Web Tools: Accessing MDDB Data Using webEIS Software*

In addition, Institute 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.

Thinking in Java, written by Bruce Eckel, Prentice-Hall Inc., 1998

Because Web technology changes quickly to keep up with user demand that it perform better and better, some of the following documentation may have dated information as to what is now available with specific releases of the software. However, they still provide excellent information and were used as references when writing this paper:

Distributing SAS/AF Models with Java Clients, written by Chris Bailey and Karl Moss for SUGI24

Deploying Existing SAS/EIS and SAS/AF Software Applications to the Web, written by Dave Horne, Don Henderson, and Vince DelGobbo for SUGI24

SAS Component Object Model (SCOM) in Version 7 of SAS/AF Software, written by Tammy Gagliano and Sue Her for SUGI24

SAS/IntrNet Software: A Roadmap, written by Donald Henderson for SUGI23

Java for Competitive Advantage, written by Chip Kelly for SUGI23

Overview of Java Components and Applets in SAS/IntrNet Software, written by Barbara Walters and Don Chapman for SUGI23

ACKNOWLEDGMENTS

Many thanks go out to the folks that helped review and provide content for this paper: Glen Walker, Ryan Norris, Tony Prier, Scott Leslie, Rich Main, Marty Tomasi, Jennifer Appetta, Jean Moorefield, and David Malkovsky.

CONTACT INFORMATION

If you have any questions or comments, please feel free to contact us.

Carl LaChapelle
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Work Phone: (919) 677-8000, extension 7712
Email: Carl.LaChapelle@sas.com

Tammy Gagliano
SAS Institute Inc.
Work Phone: (630) 724-9496
Email: Tammy.Gagliano@sas.com