Developing and Deploying Java Applications Around SAS: What they didn't tell you in class

Jeff Wright, ThotWave Technologies, Cary, NC
Greg Barnes Nelson, ThotWave Technologies, Cary, NC

ABSTRACT

Java is one of the most popular programming languages today. As its popularity continues to grow, SAS' investment in moving many of their interfaces to Java promises to make it even more prevalent. Today more than ever, SAS programmers can take advantage of Java applications for exploiting SAS data and analytic services. Many programmers that grew up on SAS are finding their way to learning Java, either through formal or informal channels. As with most languages, Java is not merely a language, but an entire ecosystem of tools and technologies. One of the biggest challenges in applying Java is the shift to the distributed n-tier architectures commonly employed for Java web applications.

In this paper, we will focus on guiding the programmer with some knowledge of both SAS and Java, through the complex deployment steps necessary to put together a working web application. We will review the execution models for standard and enterprise Java, covering virtual machines, application servers, and the various archive files used in packaging Java code for deployment. We will also survey the mechanisms available for using SAS services from the Java language and provide some helpful tips for installing these services. We will follow this by using a detailed example to show how a working application is created using the AppDev Studio development environment. Here, we will demonstrate how you can incorporate industry-standard testing and deployment strategies to migrate your code to a production environment that includes a Java application server, and SAS compute and data services (also referred to here as the Analytic/ Data Tier). We will conclude with pointers to some of the most popular tools supporting Java development today.

INTRODUCTION

Learning a new language like Java is much like learning SAS. Remember when you first learned about the Program Data Vector and various tricks like _n_, first. and last. syntax and common set and merge techniques? Of course, understanding what was happening at each iteration of the Data Step or the complicated syntax for performing a sub query in SQL didn't make you the programmer you are today. In fact, learning the "language" itself doesn't get you close to understanding the complexity of how that language can and should be used, nor does it prepare you for the complex world of "architecture".

Like SAS and the myriad of products that have become bedfellows to the Data Step, Java makes our jobs even more interesting. In your first Java class you probably learned about "objects", classes and methods. That understanding is essential for moving ahead to develop enterprise class applications that can support a large number of users and more data, but it's not enough.

In this paper, we are going to take you to that next level of mastery by helping you understand how Java and SAS can be used together. We will cover the technologies available to help you accomplish that and some recommendations on the appropriate "packaging" for the source code and related files to help you manage the application in a more robust manner.

WHAT'S COVERED IN THIS PAPER

We cover a tremendous breadth of material in this paper. Before we get started, here's the road map:

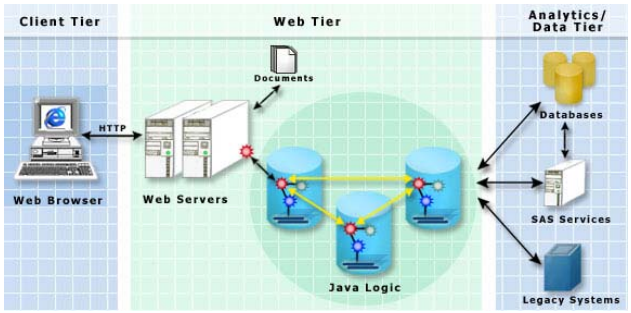
- Java Web Application Concepts
  - Client Tier
  - Web Tier
  - Analytics/Data Tier
- How Can SAS and Java Be Used Together
  - Comparison of Strengths
  - Choosing an Option for Talking to SAS
- SAS and Java Integration Details
  - Java Execution Models
  - SAS Service Details
- Packaging, Building, and Deployment
  - A Complete Example
  - Example Project Directory Structure
  - Testing Your Web Application
  - Required Infrastructure
  - Required Development Skills

JAVA WEB APPLICATION CONCEPTS

Java, as you will recall from your first course, is a programming language designed to create "applications" that run on the web. (For the purposes of this audience, that's how most people use Java.) These applications can run on the client, the server, or both. The major challenges that Java attempts to solve include:

- **Platform Independence** – Java can run on most platforms
- **Speed/ Efficiency** – because Java is not strictly an "interpreted" language, it is relatively fast and provides an efficient use of computing resources
- **"Open" standards** – many IT managers are looking for technologies that "play well with others"
- **Object Oriented** – because the issues we are trying to solve with Java can be expressed in terms of the problem (real world objects), we can express the solution in the same content – not technical representations of the solution. Also, objects give us mechanisms for using good design patterns such as Model-View separation.

Java can be run on the client, the server, or both. SAS can produce content from both Data and Proc Steps, as well as provide access to data managed by other tools. This bundling of technologies provides a rich, and subsequently complex, landscape for us as architects. In this section, we present a model architecture for Java web applications that also use SAS. Dynamic web applications use a multi-tiered architecture. These tiers represent logical layers in our application that have different responsibilities, and go by many names. One terminology we like -- Client Tier, Web Tier, and Analytics/Data Tier -- is represented in the diagram below.



CLIENT TIER

When we think about web applications, we most often visualize the web pages that display data, graphs and tables. These viewable components (also known as "presentation components") in the client tier are usually HTML pages (with Dynamic HTML/ JavaScript).

Be aware that when your browser is showing an HTML page, that doesn't necessarily mean there is a corresponding HTML file on the server. When we refer to *dynamic web pages*, we're referring to web tier logic that creates the page on the fly, usually based on data. In the Java world, dynamic web pages are created using JSP (JavaServer Pages) or servlets. JSP pages are best used for generating text-based content, often HTML or XML. Servlets are most appropriate for generating binary content, or content with variable structure. For those familiar with SAS/IntrNet or Active Server Pages (Microsoft), these are analogous technologies that produce dynamic web pages.

Heavier-weight clients, such as "rich" standalone clients, application clients, or special content formats such as Macromedia Flash or Adobe's PDF, can implement relatively more view functionality in the Client tier, and less in the Web tier (web server).

Client tier programs translate user actions and input into server requests and format server responses into some form (usually visual) that a human can use.

WEB TIER

The web tier makes application functionality available on the World Wide Web by responding to requests from the client tier. It accesses data and business functionality from the analytics/data tier, manages screen flow, and often encapsulates some user interaction control. The web tier typically interacts with client tier programs using the HTTP protocol (the standard protocol used by web browsers), and may access other tiers with a variety of protocols.

As we suggested above, the client and web tiers work together to define the user interface of a web application. For data entry and applications that require a great deal of user interaction or data capture, web clients can use JavaScript to perform sophisticated validation and error checking. Even though this JavaScript runs in the client tier (web browser), it may originate from a JSP page on the web tier (server). Developers new to web development must be aware of where the code will execute.

In most well-written web applications we find a clear separation between the logic of the application and the specifics of how information is presented in a web browser (Barnes Nelson, 2000; Simon, 1995). This is sometimes referred to as *model view separation*.

In the realm of application logic, confusion exists about how SAS can and should be used in conjunction with Java. This is where we have to make the tough decisions about what should be done with what language. For example, if we have a simple data entry application that records data for a clinical trial, there are several approaches that could be used. We would typically have some client tier logic performing checks that the data is well formed and valid before the server begins to do any work. There may server-side logic that provides cross visit validations or data range checks before actually writing the values to the database. It is these programs that are of interest to us.

The problem could be solved with Java; it could also be solved with SAS. Making this decision is what makes this problem so interesting and why good architects are worth their salaries. We aren't going to pre-dispose you to our bias about how this problem should be solved as it would also likely depend on numerous other factors, such as:

In what database does the information reside?

Will the code that performs the logic checks be used in other contexts besides the web data entry application?

Is there code that already does this or similar things?

Does the logic require the strengths of Java?

Does the logic require the strengths of SAS?

The bottom line is that this is an architectural decision that should be left to the smart folks who do this for a living. However, knowing that you have this decision to make and what the implications for how you might "talk" to SAS is exactly the kind of thing we want to focus on in the next section of this paper, but first, let's talk about the analytics/data tier.

ANALYTICS/DATA TIER

Most modern web based applications utilize data at some level as either the focus of the application (data delivered over the web) or to store information about the application or its users (security, configuration, personalization). Just as important as the web tier, is the database component that supports the architecture.

Just as with Base SAS, there are several ways to tackle the data problem. We first have to ask ourselves "where do we want to store the data" before we can ask "how do we access the data". SAS tables, MDDBs, XML, flat files and relational databases of all kinds can be used as a persistent storage for our web applications.

Note that our diagram shows Legacy Systems in this tier. Other applications are frequently a necessary source (or destination) for the data in our application.

Finally, we show SAS explicitly in this tier. SAS is ideal for sifting through large quantities of data and summarizing it to indicate aggregate results and trends. Because these types of services need direct and high-speed access to large amounts of data, we like to broaden the definition of what is normally called the data tier to include analytic services.

HOW CAN SAS AND JAVA BE USED TOGETHER

With this foundation of Java web application concepts, we can move on to tackle the question – "how can SAS and Java be used together?" The answer, of course, is a huge topic in-and-of itself, but let's try to outline some possibilities.

COMPARISON OF STRENGTHS

First let's compare the strengths of Java and SAS in the areas of presentation, application logic, and data management.

Presentation

Java is very strong in the areas of:

- Generating dynamic web content on demand
- Conditionally presenting content based on security rules or user preferences
- Acting as a controller for form submission and page

redirection

- Displaying charts and graphics that are more general in nature (bar, pie, line) or those that require client side interactivity (applets)

SAS is excellent at producing "specialty" content such as:

- STAT/ ETS/ QC charts
- Output from ODS

However, SAS is not generally appropriate for generating HTML. It's often better to separate the logic (SAS-produced numbers) from how it is presented.

Application Logic

Because Java as a language is fast and tends to scale very well (it has multi-threading built it), Java is good for:

- Generalized routines for validation
- Simple calculations
- Manipulating data in memory for preparing for presentation
- Implementing business rules
- Coordinating transaction logic

SAS, of course has a very strong foundation with its analytic capability. SAS programs/ macros are very useful when they:

- Prepare data for delivery to the web
- Perform advanced analytics
- Are used in non-web applications

Data Management

Java doesn't have a native database, but instead offers an API (Java Database Connectivity, or JDBC) that talks to databases such as Oracle and SAS. Access to databases can be done "natively" through JDBC.

SAS has a native database type that can be accessed through libname engines and used throughout the system by reference. In addition, SAS can talk to a multitude of database vendors through either native engines (Oracle) or through generalized methods (ODBC).

Since the purpose of this paper is not to compare SAS to Java, but to focus on how they can work together to build world class applications, let's turn our attention on how we can talk to SAS from Java to surface the power of SAS in web applications.

CHOOSING AN OPTION FOR TALKING TO SAS

There are several ways to connect to SAS using external clients. One that many SAS users have heard of is IOM (Integration Technologies Integrated Object Model). Enterprise Guide™ is one example of an external client that requires IOM, and this is available to Java as well. When you write Java clients, you don't always need to use IOM. The next section outlines the various methods for accessing SAS; specifically, we will discuss three:

- JDBC Access to SAS Data (using either SAS/Share or Integration Technologies)
- SAS/Connect Protocol for Analytic and Data Services
- Integration Technologies, also for Analytic and Data Services

What we will introduce next are the three most common mechanisms for SAS access from Java, while giving some guidance on when to use them.

Basic Data Access

In many cases it is useful to simply gain access to the data in SAS datasets. In this case, it is our recommendation to let Java talk to the SAS data directly using a JDBC driver. This presupposes that the data is in the necessary form for display or update purposes.

Had there been any manipulation of the data required prior to display, you'll have to make the decision – "can I do that with Java or do I have to do that with SAS?" If you can do everything you need with Java, then all you need is the JDBC driver (note there is both an IOM JDBC driver as well as a SAS/Share JDBC driver).

If on the other hand, you need the power of SAS to manipulate the data prior to display, then you should refer to the next section on exploiting the power of SAS with Java.

Exploiting the Power of SAS

Given the fact that you want SAS to perform some analytics or implement some special rules that SAS is adept at doing, you'll need to decide the necessary level of complication. Of course, the more complex, the more powerful and robust the applications tend to be (so it would seem). The choice you face at this point is between SAS/Connect and IOM.

Exactly how to decide this relies on a bit of intuition, a strong foundation in architecture and a bit of guess-work. In our minds, the best way to think about the difference between these two protocols is: when we use SAS/Connect, we pass the responsibility off to SAS to do all of the work and we gain something in return (think of this as a package or bundle) that has to be converted back into something that Java understands; when we use IOM, you can write Java client programs that make use of the SAS IOM server almost as if it were a set of Java objects.

While we don't often condone using SAS for producing the web-ready content as this violates our philosophy on the separation of model and view, we do like the power of ODS for handling large amounts of graphical and tabular content on a page. With the introduction of ODS templates and style guides in upcoming releases of SAS, we think this will become even more powerful. By using both JSP/Servlets and ODS in combination to deliver dynamic ODS content, users have a great deal of flexibility for their reporting options.

Summary of SAS Access Techniques

The following table summarizes the main considerations in choosing each of these three access techniques:

Access Methods to Talk to SAS

What you want to accomplish	Robustness Required	Effort / Complexity	How you do this
Access a SAS dataset (table viewer/ editing)	High	Intermediate	Use JDBC to access SAS/Share data sources. SAS/Connect and IOM are not required.
Submit a SAS job and retrieve the formatted output	Low	Beginner	webAF Custom Tags (using submit tag) over SAS/Connect (IOM and SAS/Share are not required)
Interface with SAS data and the SAS workspace object	High	Advanced	Utilize the IOM Workspace Factory APIs to use with your connection (IOM spawner).

SAS AND JAVA INTEGRATION DETAILS

In this section, we provide a detailed discussion of how to accomplish the integration of SAS and Java. We begin with a review of Java execution models. With this foundation, we discuss configuration details for the primary SAS APIs for Java.

JAVA EXECUTION MODELS

Now we'd like to make a dramatic transition from the blue-sky world of architecture to the subterranean mechanisms of how Java works. These details may seem obscure, but we find that developers new to Java often have the most difficulty with tool setup issues that require a good understanding of how Java code is packaged and run.

To start, Java is a compiled, object-oriented language. In an object-oriented language, the basic unit of design and programming is a class. Objects are instances of classes. Java classes are not only assigned a name, they are also assigned a *package*. Packages are useful because they allow the language to distinguish between two classes with the same name. For example, within the standard Java library is the class *java.util.List*, which is a data structure, and a class *java.awt.List*, which is a GUI component.

Package names themselves are multi-part names separated by dots. By convention, packages for the standard Java library start with "java", and other packages start with the (reversed) Internet domain of the party that supplies the package. The fully qualified name of a Java class includes the package name and the class name. For example, *com.sas.mdtable.Cell* is the fully qualified name of the Cell class in the *com.sas.mdtable* package. All of this attention on naming and packaging is designed to make it easy to bring together class libraries from several different parties without the need to worry about duplicate names.

The source code for a Java class is stored in a ".java" file, and when it's compiled you get a *class file* with a ".class" extension. If you're familiar with C programming and compilation, you might expect the class file to contain machine instructions for the computer where it was compiled. Instead, Java compiles to bytecodes that are executed by a virtual machine. The bytecodes are the same whether you compile on a Windows PC, a UNIX machine or any other computer. This is what gives Java its platform independence. You can think of the virtual machine as an interpreter, but that glosses over the fact that the high-level Java language has been compiled to low-level bytecodes, and that virtual machines have very sophisticated optimization allowing performance to approach that of native-compiled languages like C. Java is not unique in its use of a virtual machine for execution. Other similar implementations are the Common Language Runtime of Microsoft's .Net architecture and the cross-platform abilities of SAS itself.

JAR Files and CLASSPATH

Notice that each Java class gets compiled to its own class file. When you understand that a complex application (or library) will contain many classes, you'll begin looking for a way to wrap up multiple class files in a single file for easier distribution. What you're looking for is a *JAR file* (Java Archive). A JAR file is just an archive that wraps up multiple Java class files using the popular zip file format. It also contains a manifest file, which includes descriptive information about the archive.

JAR files may be created with the command line *jar* tool supplied by Sun. UNIX users will recognize that the jar tool's command line options are inspired by the *tar* command.

How does the virtual machine actually find the Java classes that make up the application we've created? This is determined by the *CLASSPATH*. The CLASSPATH is very similar to the concept of the PATH environment variable on Windows and UNIX. It is a list of directories and archive files that are searched for the classes referenced in your application. You specify the CLASSPATH when you start the virtual machine, either through the command line or by using an environment variable. Getting all the JAR files you need and getting the CLASSPATH properly defined are typical tasks that confront you when you need to run somebody else's Java application. When you get the "class not found" error, you know you have a CLASSPATH problem.

So far we've covered the concepts for how a standalone Java application runs. The Java code for the application is compiled, and ideally archived as a JAR file. This JAR file and the JAR files for all libraries used by the application are put in the deployment location that you choose. The CLASSPATH to find them is specified when you run the Java virtual machine.

J2EE Packaging

In Sun's terminology, web applications fall under the J2EE specification (Java 2 Enterprise Edition). The good news is that J2EE applications have another layer of built-in services, above and beyond the core Java library, which a developer can use without having to create them.. These services are provided by a piece of software known as an *application server*. So an enterprise application is not just deployed to a virtual machine, it is deployed to an application server. A J2EE application server is the "container" where you deploy your enterprise application. It provides middleware services such as a web server, transaction management, and distributed communication.

In summary, the application server is the glue that connects URLs in somebody's browser to some piece of Java code you wrote.

The bad news (if you're starting to get tired of definitions) is that J2EE defines several new types of archive files. Like the JAR file, these new archives use the zip format. One is the *WAR* file, or Web Application Archive. A WAR file contains the files that make up a dynamic web application, including JSP pages, HTML pages, images, JAR files, and other file types. There is a defined directory structure for the files that make up a WAR file, and the WAR file must contain a deployment descriptor named *web.xml*. Just as the name suggests, the deployment descriptor is an XML file that tells how the application will be deployed, defining such things as what URLs the application will use and what security restrictions are placed on pages in the web site.

Here are the rules for directories within a WAR file, as defined by the J2EE specification:

- *WEB-INF*: This directory and all sub-directories are for non-web content. Users cannot see this directory in their browser. It holds the deployment descriptor *web.xml*.
- *WEB-INF/classes*: This directory is automatically added to the CLASSPATH. It is for Java class files and other resource files loaded from the CLASSPATH.
- *WEB-INF/lib*: This directory is for Java libraries. Any JAR file in this directory is automatically added to the CLASSPATH.
- *<other directories>*: All other directories in the WAR file, including the top-level directory, are for web content. This can be dynamic content (JSP pages) or static files (images, HTML, PDF documents, etc.).

The highest level of packaging you can have for a J2EE application is the *EAR* file (Enterprise Application Archive). An EAR file can contain dynamic web pages in WAR files, and Enterprise JavaBeans and class libraries in JAR files. Like a WAR file, there is an application level deployment descriptor *application.xml*.

A developer with a thorough knowledge of J2EE can create the deployment descriptors and archives for deployment to an application server. The same jar tool can create WAR and EAR files as JAR files. Commercial development environments and application servers that provide GUI tools for these steps simplify this process. The benefits of working out the packaging are realized when the application is released. An application that is packaged as a WAR or EAR file can easily be moved from a test

server to a production server.

SAS SERVICE DETAILS

With this understanding of Java, virtual machines, and JAR files, let's return to the SAS APIs for Java and examine the details of how they are used. As we have learned, there are numerous ways to talk to SAS. Which one we choose depends on what we want to do. Using the scenarios described previously, we have a simple heuristic:

- If you want to access data stored in SAS, use JDBC. There are both SAS/Share and IOM JDBC drivers available
- If you want to access the power of SAS, you have two options: SAS/Connect or IOM (Integration Technologies)

Of course, each of these requires that you have the relevant SAS products and a desire to learn enough about the technology to be more than dangerous – but efficient and effective. Users of AppDev Studio software will find much of these decisions "hidden" from you (but don't be confused, these methods are all "accessible" in the software).

For example, in AppDev Studio, most users will find it easy to develop applications that use SAS/Connect with or without the MiddleWare server. JDBC access is not so obvious and IOM is not obvious at all[1]. But don't lose faith. With a little help, a slight shove and lots of patience, you'll do fine.

SAS/Share

SAS/Share is a SAS product that allows us to talk to data (stored in SAS datasets) without being concerned that someone else is touching a record at the same time. We might want to use SAS/Share if our applications requires concurrent update access to SAS datasets. Our data entry application described earlier is a great example. You don't want to edit a record at the same time someone else is modifying the record as you will likely get locked out, or worse, get unexpected results in your database.

SAS datasets managed by SAS/Share can be accessed either by a SAS program (Data Step or Proc Step) or via Java. Since we are focusing on Java as the client of interest, let's focus on that approach. SAS/Share is a required product when accessing SAS datasets in update mode using the JDBC driver. [2]

JDBC provides Java developers with access to databases using an object-orientated mechanism. A JDBC program consists of three main elements:

The Java client code

- Written by you, the developer
- All calls to the database must conform to the JDBC API

The JDBC driver manager

- Supplied by the core Java library
- Provides a link between your Java application and the Driver

The JDBC driver

- Supplied by vendor or third party, appropriate to the database (in this case: SAS)
- Converts JDBC-compliant calls to native database calls

To access the SAS/Share data sources via JDBC, you must have the JDBC driver available to your application. The SAS/Share driver consists of two JAR files (note: we will discuss JAR files in more detail later) which are provided with AppDev Studio:

- connect.jar
- netutil.jar

They must be either included in the WEB-INF/lib directory of your web application, or installed into a common location on the CLASSPATH for your application server.

The other requirement is that SAS/Share must be running. This is achieved by *proc server*. Here is an example program to start SAS/Share:

```
proc server
     serverid=shr1
     authenticate=opt
     log=query;
run;
endsas;
```

Similarly, the service is stopped using *proc operate*. On Windows, it is also possible to configure SAS/Share as a Windows service. When using TCP transport, the share *serverid* corresponds to a service defined in \WINDOWS\system32\drivers\etc\drivers (Windows) or /etc/services (UNIX). Consult the SAS/Share documentation for full details on configuring and running SAS/Share.

SAS/Share is a separately licensed product that must be present on your target deployment server or servers.

It is also worth noting that there is an alternative JDBC supplied by SAS that uses IOM (see below) for connectivity rather than SAS/Share. Establishing a connection is different than with standard JDBC; the *java.sql.Connection* is made available through the IOM workspace. Once you have have obtained the Connection, this JDBC driver is used just as the SAS/Share driver. To obtain safe, concurrent access to datasets, you may find it necessary to include SAS/Share in your architecture even if you're using the IOM JDBC driver for access.

SAS/Connect

SAS/Connect was the first method that SAS came out with for distributed access to resources within the entire SAS system. Essentially, this method involves a Java client talking to a SAS server using SAS/Connect – a proprietary protocol that allows SAS components to talk to one another on different machines.

AppDev Studio comes with the webAF *custom tags* for JSP that use SAS/Connect (or IOM) to submit blocks of SAS code and create views of SAS results. These tags are embedded in a JSP page and have the appearance of HTML tags, but they initiate SAS operations on the server before the resulting HTML page is returned to the client's browser.

One pitfall to be aware of in using webAF and SAS/Connect is the configuration of the Java and SAS servers that will be the target of your deployment. When you install AppDev Studio, it makes all of the modifications required to your development workstation. However, when you deploy to a server environment, you will have to make these changes yourself.

On the Java side, your application must have access to the following JAR files from AppDev Studio:

- activation.jar

[1] Note: If one simply wants to use the IOM *protocol* and is content with the existing set of ADS components, you can use the com.sas.rmi.Connection component.

[2] Note: SAS tables can be accessed via the JDBC-to-ODBC Bridge in addition to the JDBC driver. This however, reduces your ability to deploy your application on multiple platforms (a strength of both SAS and Java)

- antlr.jar
- brgorb.jar
- connect.jar
- iomdriver.jar
- iomprx.jar
- mail.jar
- netutil.jar
- queryAF.jar
- remobj.jar
- webAF.jar
- webAFServerPages.jar

SAS recommends that all but the last of these be installed in the <java_home>/jre/lib/ext library, which is the standard extensions location (where <java_home> is the base directory of your J2SE SDK installation). JAR files placed in this directory are automatically added to the CLASSPATH of all applications. The last JAR file, webAFServerPages.jar, contains the webAF custom JSP tags, and is best deployed in the WEB-INF/lib directory of your web application.

If addition, you must include the *tld* file *sasads.tld* for the custom tags in your application. This file is part of Sun's JSP spec for custom tags, and contains metadata about the SAS webAF tags. The location of the *tld* file is normally defined in the web application's web.xml deployment descriptor.

On the SAS side, you first need to install updates to your SAS server software. These updates are supplied with AppDev Studio as a *cpo* transport file and come with with installation directions.

Next you need a SAS/Connect server. On Windows, this is can either be a Windows service or a SAS/Connect spawner that listens for SAS/Connect requests on a specific port number.

For example, on Windows, we start the SAS/Connect Spawner by invoking the following command:

```
C:\SASV82\spawner.exe -c tcp -telnet 2323
```

On UNIX, there is no need to have SAS/Connect running since we usually reference this through our connection. We simply provide the command that should be used to start SAS when we connect, for example:

```
/sas/sas -dmr -comamid tcp -noterminal -cleanup
```

We can test the connection using a simple JSP page or through the test facility in AppDev Studio.

Once SAS/Connect is running, we can use our Java client (program) to access SAS through the *sas.com* api (see: http://support.sas.com/rnd/appdev/webAF/api.htm).

If you use the SAS/Connect protocol, you must license SAS/Connect and SAS/IntrNet on the server side. The webAF libraries come with AppDev Studio, and you should be certain to obtain the latest updates from the SAS support web site.

SAS Middleware Server

You may (but are not required) choose to utilize the AppDev Studio Middleware Server (MWS). The MiddleWare Server enables your application to share SAS sessions (on the server) with other users (rather than having to start a new SAS session each time you connect to data) while adding performance enhancements such as preloading SAS sessions and remote SAS model classes. One of the factors in deciding to use the MiddleWare server will be whether or not you need to take advantage of the user's "WORK" datasets, which are session-specific. If you do, you may not be able to share sessions.

Integration Technologies (IOM Object Spawner)

Integration Technologies – or IOM as it is sometimes referred – is a set of APIs that allow you to talk to SAS as if it were a set of Java objects. IOM is a much more robust method and consequently more complicated when it comes to developing and administering.

IOM introduces us to the Workspace Factory. This provides us with program level access to all of the power of SAS through Java. The SAS workspace is the highest-level component in the IOM object hierarchy, and connecting to a workspace object is the first step in using an IOM server. The *WorkspaceFactory* class provides methods for creating and connecting to a SAS workspace on an IOM server.

On the Java side, IOM is a Java library and requires the following JAR files, supplied with Integration Technologies:

- brgorb.jar
- iomdriver.jar
- iomprx.jar
- netutil.jar

These *jar* files must either be included in the WEB-INF/lib directory of your web application, or installed into a common location on the CLASSPATH for your application server.

On the SAS side, you must have access to an IOM server. The IOM server is a program that is started using the *objspawn* command on either UNIX or Windows and requires a configuration file for its parameters.

On Windows, we would start the object spawner with a command like:

```
c:\sasv8\inttech\sasexe\objspawn -configFile
objspawn.cfg
```

The configuration file contains instructions on where the server is, what port number we should connect on and what protocol should be used to access the server.

Like the SAS/Connect protocol, there are volumes of documentation on IOM and how to write Java clients to utilize the full power of SAS. These can be found at: http://support.sas.com/rnd/itech/doc/dist-obj/javaclnt/javaprog/index.html

If you use the IOM protocol you must license SAS/Integration Technologies on the server side[3]. Up to now, we have referred to using SAS IOM as a preferred method of talking to SAS from non-SAS clients and we have intentionally been non-specific about how we can use IOM. The fact is that IOM can be used both as a protocol and as a set of programmatic interfaces. We really are referring to the latter in this paper as the former requires the use of com.sas.rmi.Connection object and the remote object class factory (ROCF) which plugs in the correct protocol stubs underneath the ADS components. Because of its reliance on SCL as a server side component, we don't usually recommend the its use (ROCF), but rather, we mean using IMO as a set of programmatic interfaces.

SAS OLAP Server

The focus of this paper has been on accessing SAS datasets and compute services offered through SAS. If you are using *webEIS* and a version 8 SAS server, you will also need SAS OLAP Server. The SAS OLAP Server contains critical components that are used when you implement HOLAP solutions.

[3] IOM protocol is only supported if you run against SAS 8.2

PACKAGING, BUILDING AND DEPLOYMENT

At this point, we have covered Java web applications and SAS APIs for Java. We have looked at concepts and configuration details. In this section we tie it all together, using an example to discuss packaging, building and deployment.

A COMPLETE EXAMPLE

To illustrate the principles discussed in this paper, we created a sample J2EE web application that uses all three Java/SAS connectivity approaches. The main web page is an index page with links to four JSP pages:

- *CustomTagExample.jsp*: This page contains webAF custom tags that connect to SAS, submit a block of SAS code, and display results in both Bar graph and table form.
- *CustomDataViewer.jsp*: This page contains webAF custom tags to create a custom table view of a SAS dataset.
- *JDBCConnection.jsp*: This page instantiates a Java object, which performs a query against SAS/Share using JDBC. The results are displayed in a table.
- *IOMPackageSample.jsp*: This page contains a form, which allows code to be submitted. When the form is submitted, a Java servlet connects to an IOM server, creates several artifacts (PDF, RTF, SAS Dataset and Excel file), packages them up and sends them back to the client for rendering.

In our example, we followed the typical practice of encapsulating blocks of Java logic into standalone Java classes or servlets. In this case, the Java logic consisted of data access code that manipulates the JDBC and IOM APIs. Although it is possible to embed arbitrary amounts of Java code into a JSP page, it is best to leave JSP files focused on presentation concerns.

The details of how the web tier communicates with the analytics/data tier involve settings such as host names, ports, and other communication parameters. Multi-tiered applications need a way to specify *environment-specific* customizations. In our basic example, we put these customizations into a Java properties file that is read at run-time. The *java.util.Properties* class is frequently used by Java applications to externalize configuration details into a file that contains setting in the form *name=value*. Another approach to the problem of environment-specific customizations would be to use a directory service.

We like to create an automated build process for our projects. Ideally the build process should be scripted in a way that doesn't require a GUI development environment. Using this approach, an administrator that is not a Java developer can perform the build, and builds can be scheduled as a batch process to check for integration problems. A very popular tool for Java build automation is *Ant* (http://ant.apache.org). Ant has some similarity to the make utilities that have been used for years in C and C++ development. It not only knows how to run the Java compiler and jar tool, it is capable of doing all file manipulation activities required for a full-featured build system. Ant uses an XML file that defines the build targets and steps. Our basic *build.xml* file contains the following targets:

- *compile*: Compile Java sources
- *war*: Package web content as a WAR archive
- *ear*: Package the entire application as an EAR archive
- *clean*: Delete all compiler output and archives, to reset the project to a clean state

We created and tested our sample application on Windows workstations. Since we were using all three SAS services discussed in this paper, we created Windows BAT scripts to run SAS/Share, the SAS/Connect spawner, and the IOM object spawner. For production deployment on Windows, you would typically configure these as Windows services. For production deployment on UNIX, you would write UNIX scripts to launch the appropriate services, and integrate them with the *init* process so they would be started at automatically.

EXAMPLE PROJECT DIRECTORY STRUCTURE

We used the following directory structure for our application. This structure not only includes Java source code, but also library files, web content, SAS data sets, and administrative/build scripts.

Web Application Directory Structure

Directory	Description
bin	Scripts for starting SAS services, plus IOM config
data	Sample datasets
devlib	Java JAR files not deployed with the WAR file, either because they are installed to a different location or because they are only needed for compilation, not runtime.
logs	Log files for SAS services, created by scripts in bin directory
make	Location of Ant *build.xml* instructions
make/descriptors	Holds deployment descriptor files not included in the web directory, namely *application.xml*
make/dist	Location where *build.xml* puts WAR and EAR files
src	Java source code
web	Web content (JSP files, images, etc)
web/WEB-INF	Standard directory for web application metadata not accessible to user browsers. Location for *web.xml* deployment descriptor and *sasads.tld* tag descriptor.
web/WEB-INF/ classes	Ant build puts compiled Java classes here. Also the location for the *DataViewer.properties* file.
web/WEB-INF/lib	JAR files deployed with the web application, such as *webAFServerPages.jar*

TESTING YOUR WEB APPLICATION

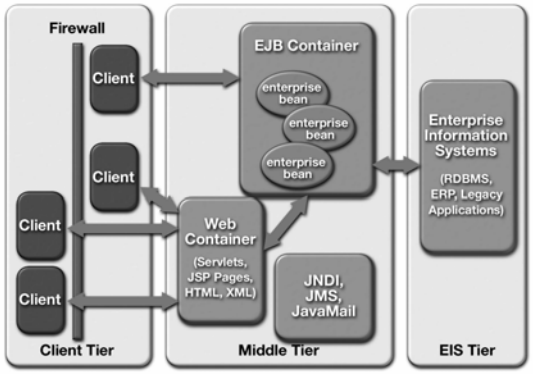
Once you deploy your applications, it is important that you test all of the components within it. This is best accomplished by planning a test deployment early in your project to make sure you have a clear understanding of all the server configuration issues. The following suggestions have been compiled to help you think about the kinds of things that can (and usually will) go wrong:

- If your Web application cannot locate the tag extension information, insure that the following is true
  - the *webAFServerPages.jar* is included in the application's WEB-INF/lib directory
  - the *web.xml* file contains the appropriate reference to the *sasads.tld* file
  - the *sasads.tld* is stored in the location specified in *web.xml*.
- Make sure all JAR files required by the SAS API you have chosen are installed in the appropriate location.
- If you are using the AppDev Studio Java components to connect to SAS, you will need to apply the AppDev Studio server-side catalog updates to the SAS server.
- Make sure you use the correct directory path for any images, styles or other includes that are referenced in your pages. You may need to change the references in your code before you deploy your JSP pages and servlets so they point to the correct relative or absolute paths.

- If your application has problems locating resources you have specified, verify that the correct URL is provided to access those resources. If the URL begins with a "/", then it is a resource that can be found in a directory relative to the application's root directory. If the URL does not begin with a "/", then the resource can be found relative to the current JSP page. This means that you must check the URLs that are interpreted by a Web browser, including links to other pages, HTML <form> actions, image and style sheet links, to verify that the correct path is specified.
- If you create an EAR file that you want to distribute to multiple combinations of operating systems, Web servers, applications servers, and SAS servers, make sure that you test that Web application in all intended environments.

REQUIRED INFRASTRUCTURE

Let's recap the required technical infrastructure to host our web application. In many of the scenarios outlined above, we have made some fundamental assumptions about what was "running" so everything will function properly. In the diagram below (taken from Sun Microsystems web site) we show the components that typify a multi-tiered web application.



Client Tier

In the client tier, the only technical infrastructure is the web browser. For an intranet application, you will frequently be targeting a corporate standard web browser. For a public Internet application, you should develop a profile of browsers and browser settings that you will support. There is a tradeoff here between supporting the largest possible audience, including older browsers, and the richer experience possible with up to date browsers, especially with regards to Dynamic HTML and Cascading Style Sheets (CSS). The good news is that up to date browsers, such as Internet Explorer 5.5 and newer, Netscape 7.0 and newer, and Mozilla 1.0 and newer, have a much greater compatibility than in the early days of the web.

Web Tier

In the web tier, you will need to be sure that Java is installed. The Java2 Standard Edition Software Development Kit (J2SE SDK) can be downloaded from http://java.sun.com. You will also need an application server. There are a number of commercial application servers such as BEA WebLogic and IBM WebSphere. It is also possible to use open source alternatives such as Tomcat (http://jakarta.apache.org/tomcat/) and JBoss (http://www.jboss.org). You may also choose to use a standalone web server such as Microsoft IIS or Apache. Most application servers give you the ability to operate using a built-in web server, or to act as a plug-in to IIS or Apache. Finally, in the web tier, you must have the Java libraries required to use SAS, and possibly JDBC drivers for direct access to any database present in the third tier. Recall from previous discussions that these libraries will be in the form of JAR files, and that they can be included in the WEB-INF/lib directory of a WAR file for deployment.

You must verify support for the correct version number of the JDK before selecting an application server. The SAS custom tag library and other SAS libraries require support for the Servlet 2.2 and JSP 1.1 specification using Java 1.2.2 or higher. In addition, the SAS server that you access from your Web applications must include the AppDev Studio Server-Side updates.

In the web tier, you will find the following types of files:

- HTML
- JavaScript
- Images
- Documents (Doc, RTF, PDF)
- JavaServer Pages (JSPs)

In addition, you will also find directories for the Java server side components. These include:

- Application descriptor files (web.xml)
- JAR, WAR and EAR files
- Custom Tag definitions and libraries
- External libraries (such as SAS' APIs for data access, compute services and IOM libraries)

A fairly typical directory structure for a web application that utilizes SAS APIs has already been shown in this paper (Section: Example Project Directory Structure) along with a description of what should be contained in each directory.

Analytics/ Data Tier

In the analytics/data tier, the specific infrastructure you need will depend on your architecture choices. Given the topic of this paper, we expect Base SAS software to be included in this tier. On top of this are the services required for Java access; some combination of SAS/Share, Connect Spawner, and IOM Object Spawner. If you are using webAF with SAS/Connect, you will need to apply the AppDev studio updates to SAS. If your application uses a database such as Oracle, DB2, or Microsoft SQL Server in conjunction with SAS, that is part of this tier and typically introduces a need for SAS/Access.

In summary, here are a few best practices around technical infrastructure:

- Although we haven't specified any versions of software in this discussion of infrastructure, that doesn't mean that versions are unimportant. On the contrary, establishing a working configuration requires carefully making note of required versions of all software. All members of the development and support team should understand the selected versions of tools. Documenting and managing the specific versions of infrastructure products is part of the configuration management for the application.
- Be sure to understand and document the location of all log files. Typically, you will have server logs for the application server, for SAS, and for your database. These will be invaluable in trouble-shooting production problems. Also insure that the log files or log directories have a clean-up mechanism in place so that you disk space is not exhausted by logs!
- Clearly there are a number of moving parts in a multi-tiered application. To deliver the availability that users expect, it is useful to initiate proactive monitoring of all the server-based processes that must be running and healthy for the application to function.
- Security is beyond the scope of this paper, but the architecture and technical infrastructure should be carefully planned to support the requirements of the application.

Typical Application Infrastructure

Client Tier	Web Tier	Analytics/ Data Tier
Web Browser	J2SE SDK Application Server Web Server SAS Java Libraries DBMS JDBC driver	Base SAS Software SAS/Share Connect Spawner IOM Object Spawner DBMS of choice SAS/Access

Required Development Skills

In the same way, it is useful to profile the development skills required for building a web application using SAS and Java. A representative mix of skills is listed in table below.

Development Skills

Client Tier	Web Tier	Analytics/ Data Tier
HTML JavaScript CSS Graphics	Java Object-Oriented Design JSP Servlets JDBC	Base SAS SAS Macro SQL SCL (if SAS models are used) Data Modeling

CONCLUSION

This paper has been a high-level tour through many topics relevant to the creation of web applications using SAS and Java. Our intention was not to give you a cookbook for all of the technologies we've touched upon. Instead, we wanted to survey the important considerations so you know what questions must be answered when constructing a working application. Because the breadth of material we've covered is so wide, we've focused on the integration and deployment aspects of these technologies. We expect that the nuts and bolts of programming in SAS and Java are within the grasp of our audience, the working developer.

Other topics that you may wish to explore as you go deeper into this subject are:

- Security
- Directory Services
- Automated testing using tools such as JUnit (http://www.junit.org)
- Performance and load testing tools such as Apache's JMeter (http://jakarta.apache.org/jmeter)
- Model-View-Controller frameworks such as Apache Struts (http://jakarta.apache.org/struts/index.html)

APPDEV STUDIO 3.0

Since this paper has been written just prior to the release of AppDev Studio 3.0 (to be released with SAS 9.1), it is worth noting the changes that we will see with that new release.

- With the Servlet 2.3 and JSP 1.2 specifications, web application containers are moving away from dependence upon the extensions directory. The recommendation is to deploy fully self contained web applications based upon the notion that this improves the robustness of each application (they cannot stomp on one another's jars) and the fact that disk space on the server is cheap. As a result, starting with AppDev Studio 3.0, SAS will default to building fully self contained web applications and there will be no recommendation to store SAS jars in the Java extension directory. This has a number of benefits, such as eliminating one extra deployment step, protecting AppDev Studio from incompatible downloads into the extensions directory (e.g., version conflicts with an XML parser or SAS jar), and providing a version based jar repository for applets so that different applets on the same machine can run with different sets of underlying jars.
- SAS has renamed their runtime jars such that they all are prefixed with "sas.", and repackaged many of the classes to provide more granular sets of jars to varying solutions. Therefore, the recommended set of jars be changing.
- In version 3.0, webAF uses Ant internally for builds and supports user customization of the Ant build script. All Ant tasks are exposed in the build menu so that they may be driven directly from the webAF GUI. Or, the build scripts can support batch builds and packaging.
- Tag library deployment is significantly easier in Servlet 2.3 / JSP 1.2. The TLD file is stored inside of the tag library jar file and automatically discovered by the container. This eliminates many of the problems associated with mapping the taglib URI in the web.xml. SAS will follow this practice with our custom tag libraries in ADS 3.0.
- In AppDev Studio 3.0, SAS allows the creation of web application projects instead of JSP or Servlet projects. Templated content is supported, as are JSTL and Struts. We also support development of portlets for the SAS Information Delivery Portal. They build .war files for webapp deployment with all necessary SAS servlet mappings (e.g., ContentServlet, MDQueryServlet, etc.) pre-defined.

ACKNOWLEDGMENTS

The authors would like to sincerely thank several people for their guidance and thoughtful review of this manuscript. Specifically, we would like to thank Rich Main, Steve Jenisch, Pat Herbert, Julian Anderson and Mary Bednarski.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.
Please feel free to contact the authors at:

Jeff Wright or Greg Barnes Nelson

jwright@thotwave.com or greg@thotwave.com

2054 Kildaire Farm Rd, #322

Cary, NC 27511

800.584-2819 – Phone/ Fax

Bibliography and Recommended Reading

The classic advice on learning web development is: surf the web and "view source" on web pages that you like. There are many, many resources on the web for learning web development. Here are a few goodies...

Web Development with Java Server Pages, Duane Fields et al. (Manning)

Databases on the Web: Designing and Programming for Network Access Patricia Ju (Out of print, but does a nice job on fundamentals of web architecture and separation of layers for web applications)

XML and SAS: An Advanced Tutorial. Barnes Nelson, G. (2000). SAS Users Group International, Indianapolis, IN, SAS Institute.

The Art and Science of Smalltalk: An Introduction to Object-Oriented Programming Using VisualWorks. Simon, L. (1995). London, UK, Prentice-Hall.

Beginning Java Databases: JDBC, SQL, J2EE, EJB, JSP, XML (by Kevin Mukhar, et al.)

Thinking in Java (by Bruce Eckel) http://www.bruceeckel.com/

Java in a Nutshell, Flanagan, D. O'Reilly & Associates, 1999, ISBN 1-56592-487-8

Core Java 2, Volume I - Fundamentals, Horstmann, C.S. and Cornell, G. Prentice Hall, 1999, ISBN 0-13-081933-6

Core Java 2, Volume II - Advanced Features, Horstmann, C.S. and Cornell, G. Prentice Hall, 2000, ISBN 0-13-081934-4

Sun Microsystem's Java tutorial (http://java.sun.com/docs/books/tutorial/)

JSP or Servlets - Which architecture is right for you? http://www.adtmag.com/java/article.asp?id=354&mon=3&yr=2001

Understanding JavaServer Pages Model 2 architecture http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

Developing Data-Driven Applications Using JDBC and Java Servlet/JSP. Chad Ferguson and Sandra Brey (2003) Technologies, SUGI 28 (http://www2.sas.com/proceedings/sugi28/049-28.pdf)

"Web Application Deployment Tips and Tricks" http://support.sas.com/rnd/appdev/webAF/server/deployingapps.htm

"Remote Access Troubleshooting Guide" http://support.sas.com/rnd/appdev/doc/RemoteAccessTSG.htm

"webAF Reference" http://support.sas.com/rnd/appdev/webAF/reference.htm

"SAS/Share Driver for JDBC" http://support.sas.com/rnd/web/intrnet/java/jdbc/index.html

"SAS Integration Technologies, Developing Client Applications" http://support.sas.com/rnd/itech/doc/dist-obj/index.html