

# The Use of Java with The SAS® System

## Clare Nicklin, Amadeus Software Ltd., Oxfordshire, UK



### ABSTRACT

As a Java experienced SAS consultant, I have spent much time in the last three years developing Java interfaces to The SAS System. Whether the style of interface has involved client-side or server-side processing, a variety of programming techniques and software has been used to complete the necessary tasks and projects. From my experience of developing such tools, I have come to realise that there is much confusion surrounding the use of Java to interface with the power of The SAS System. This paper hopes to alleviate some of that confusion by providing a number of different, potential routes into SAS from Java applications. The different ways of accessing The SAS System can prove to be confusing, but they need not be as expensive as using complicated, specialist development kits. Different tools and software products can be employed to provide the connection to SAS from stand-alone or browser-based applications. Several different SAS products are mentioned as four potential routes into SAS are discussed including their software requirements, methods, advantages, and disadvantages.

This paper is intended to be of use to both developers and managers to aid in the conception of applications which interact with The SAS System.

### INTRODUCTION

Many consider that the only way in to SAS from Java is by using SAS tools such as AppDev Studio™. This is not the case. AppDev Studio is simply a glorified Java Development Kit (JDK) developed by SAS to ease the development of Java applications to access SAS. It is very useful tool, which can help decrease an applications development time by providing ready-made data access and manipulation classes. However, it is not the only way that SAS can be used and accessed from Java. There are other SAS tools which provide an amount of access using other techniques, such as SAS® Integration Technologies. There are also non-SAS related Java standards, such as JDBC and Remote Method Invocation (RMI). Such standards need planning so that the ways in which they can be used and the other SAS components necessary for their employment can be determined. The following methods of Java usage with SAS are not meant to be an exhaustive list of possibilities, but an example of the kinds of architectures that can be used. There are other methods, as well as similar methods between client- and server-side processing that could also be considered. Generally, Java development can be split into two different types of processing, which are referred to throughout the paper: client-side and server-side processing.

#### Client- or Server-Side Processing

Client-side processing refers to the development and implementation of Java applications, applets, and additional classes which are run from the client machine (or browser) in a distributed network. The user interfaces can be highly interactive, rich-client interfaces. Whilst, this is an advantage to application usability, systems can be liable to restrictions such as the overhead involved in downloading Java classes to the client machine.

Server-side processing refers to the development and implementation of servlets, JavaServer Pages (JSPs) , and additional classes which are run from the server machine (or web server) in a distributed network. The user interfaces may be less interactive, thin-client interfaces restricted to the look and feel of HTML. Whilst, this may be considered to be disadvantageous, applications can be quick and functional while simply producing and downloading undemanding files to the local machine.

### JAVA ACCESS TO SAS

#### AppDev Studio

##### *Probable Software Requirements*

As well as needing AppDev Studio itself, licensing for the software to access SAS would also be needed such as SAS/Connect® or Integration Technologies.

Additional software is also necessary such as: a recent JDK<sup>1</sup>, which is the short-cut name for the set of Java development tools needed to build and compile Java applications (additional class libraries in the form of JAR files from AppDev Studio are placed here); a web server such as Apache<sup>2</sup>, which hosts all the web-based files, including any HTML files, JavaServer Pages (JSPs), and other Java classes necessary for the running of the application; and a servlet engine such as Tomcat<sup>3</sup>, which is necessary to interpret JSPs if server-side processing techniques are employed. AppDev Studio is currently shipped with jdk1.3.0\_01, an Apache web server, and the Tomcat servlet engine.

##### *Method*

AppDev Studio is a single interface for the development of thin client applications. It supports major web standards, on both the server and the client side, and allows simple communication with the SAS System. Applications are often developed through the webAF™ programming environment, which can be simple and effective for quick development.

webAF provides an integrated visual programming environment, which allows for the drag and drop creation of objects performing functions such as accessing SAS data sets. Therefore the submission of Base SAS programming logic and data manipulation code, and even the utilisation of remote SCL classes running on a SAS server, can be relatively simple.

For server-side development, webAF provides a set of JavaBeans called 'TransformationBeans' that transform SAS data into simple Web pages. TransformationBeans are used to link to webAFs existing models and represent any results on web pages in HTML. They can be built using both Java scriptlets and XML tags to generate the dynamic-content portion of a page.

##### *Example*

The basic tasks involved in the simple development of a JSP to access the sashelp.class data set through Integration Technologies are described as an example of creating a small server based application.

Firstly, the IOM (Integrated Object Model) Spawner on the server machine (or local host if in development) must be started. Secondly, a new JavaServer Pages Project must be created using the SASADS template directory. Once created 'index.jsp' opens, ready to be built upon by using the developers own HTML, XML, and/or Java and by dragging and dropping any predefined webAF objects from those provided.

Thirdly, a connection definition must be defined, shown in Figure 1, for the IOM Server configuration including the host IP address and the port number the IOM spawner is listening on. User names and passwords to the server may also be necessary.

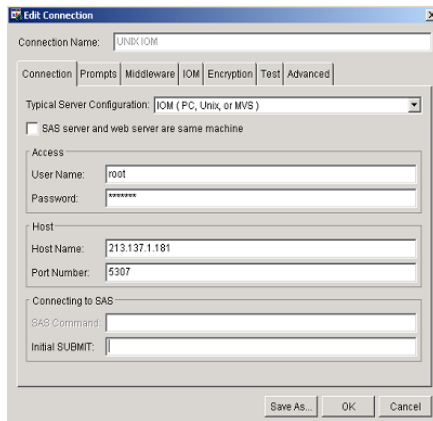


Figure 1 – Registering a Connection

Finally, some predefined objects need to be added as follows:

- from the Data Viewers tab, drag the Table object on to the screen.
- from the SAS tab drag and drop the DataSetInterface object onto the table
- a connection object is added automatically
- edit the Customiser for the DataSetInterface and enter the desired data set as sashelp.class

Using this method, the JSPs code is built in XML tag code by webAF and the source view appears as below.

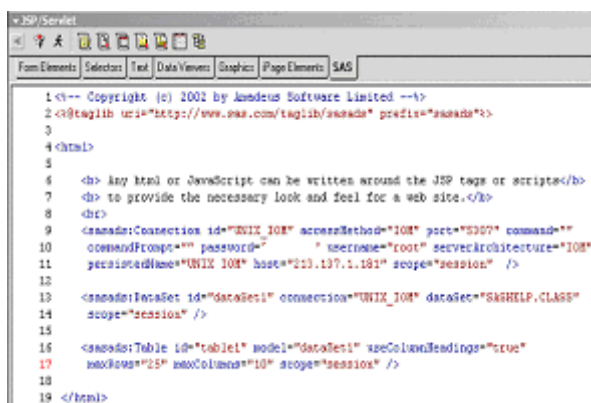
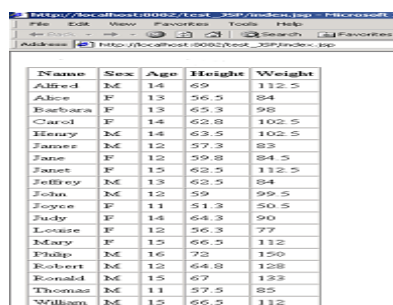


Figure 2 - webAF Automatically Creates Code

The application can then be viewed by selecting 'Execute in browser' from the Build menu, as below.



| Name    | Sex | Age | Height | Weight |
|---------|-----|-----|--------|--------|
| Alfred  | M   | 14  | 69     | 112.5  |
| Alice   | F   | 13  | 56.5   | 84     |
| Barbara | F   | 13  | 65.3   | 98     |
| Carol   | F   | 14  | 62.8   | 102.5  |
| Henry   | M   | 14  | 63.5   | 102.5  |
| James   | M   | 12  | 57.3   | 83     |
| Jane    | F   | 12  | 59.8   | 84.5   |
| Jane    | F   | 15  | 62.5   | 112.5  |
| Jeffrey | M   | 13  | 62.5   | 84     |
| John    | M   | 12  | 59     | 99.5   |
| Joyce   | F   | 11  | 51.3   | 50.5   |
| Judy    | F   | 14  | 64.3   | 90     |
| Louise  | F   | 12  | 56.3   | 77     |
| Mary    | F   | 15  | 66.5   | 112    |
| Philip  | M   | 16  | 72     | 150    |
| Robert  | M   | 12  | 64.8   | 128    |
| Donald  | M   | 15  | 67     | 133    |
| Thomas  | M   | 11  | 57.5   | 85     |
| William | M   | 15  | 66.5   | 112    |

Figure 3 - Query Results

## Advantages

Specialist software such as AppDev Studio can dramatically decrease the time and resources needed to develop web-based applications to interface with SAS. Many of the complicated tasks involved in connecting to SAS or accessing and manipulating data are hidden from the developer, instead providing quicker, more intuitive methods.

## Disadvantages

All the software involved in developing web-based applications using AppDev Studio can be expensive. Furthermore, AppDev Studio gives little insight into the complicated process of readying applications for deployment into live environments once they are built. Benefits have to be compared against the costs in deciding on AppDev Studio's utility.

## Directly Using the IOM Server (Integration Technologies)

### Probable Software Requirements

As well as needing Integration Technologies itself, and the classes it provides, additional software would also be necessary such as a JDK, web server, and servlet engine (for server-side development).

The IOM Spawner of SAS Integration Technologies is used. External applications communicate with SAS software through this object spawner, which listens constantly for calls to SAS on a specific port. Any conversation between the application (via the web server) and the SAS software is done through the spawner.

### Method

The Java software included with SAS Integration Technologies is designed to allow the use of the functionality of the SAS IOM Server in a Java program. Using this software, you can write Java client programs that make use of the SAS IOM server almost as if it were a set of Java objects, whether that program is an applet, a stand-alone application, a servlet, or an enterprise JavaBean.

The standard installation of SAS Release 8.2 comes with all the necessary JAR files containing the Java connection classes to Integration Technologies on its Client Side Components Disk.



Figure 4 - Flow to SAS via IOM Spawner

### Example

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.

The IOM spawner must be running. The following code allows a small piece of SAS code to be submitted through Integration Technologies' WorkspaceFactory to create a small data set to the sasuser library:

```

try {
    Properties iomServerProperties = new
        Properties();
    iomServerProperties.put("host",
        "localhost");
    iomServerProperties.put("port", "5310");
    iomServerProperties.put("userName",
        "username");
    iomServerProperties.put("password",
        "password");
    Properties[] serverList =
        {iomServerProperties};

    WorkspaceFactory wFactory = new
        WorkspaceFactory(serverList,null,null);
    WorkspaceConnector connector =
        wFactory.getWorkspaceConnector(0L);
    IWorkspace workspace =
        connector.getWorkspace();

    ILanguageService sasLanguage =
        workspace.LanguageService();
    sasLanguage.Submit("data
        sasuser.testset;x=1;run;");
  }
  
```

```
wFactory.shutdown();
connector.close();

}
catch(Exception e) {
    e.printStackTrace();
}
```

Where localhost could be substituted for an IP address or a server name and the port number would reflect the port listening on that server for calls, the standard port is 5307.

### Advantages

The standard installation of SAS Release 8.2 comes with all the necessary JAR files containing the Java connection classes to Integration Technologies on its Client Side Components Disk, without having to license a costly piece of software such as AppDev Studio.

### Disadvantages

Specialist software such as webAF is very useful for quicker development using pre-existing classes that provide for application features that might otherwise take time to develop and test. Several more JAR files accompany AppDev Studio which contain such classes. Once more the benefits have to be compared against the costs.

## Java Database Connectivity (JDBC)

### Probable Software Requirements

SAS Integration Technologies or SAS/SHARE®. Additional software would be needed such as a recent JDK and a web server such as Apache. If server-side processing techniques are to be employed, a servlet engine such as Tomcat would also be needed.

### Method

JDBC allows programmers to connect to, update, and query databases using the Structured Query Language (SQL). Standard SQL statements such as SELECT, to obtain data, and CREATE, UPDATE, INSERT, and DELETE, to modify data, are supported. JDBC allows third-party drivers to connect to specific databases i.e. database vendors provide their own drivers registered with and used by the JDBC driver manager.

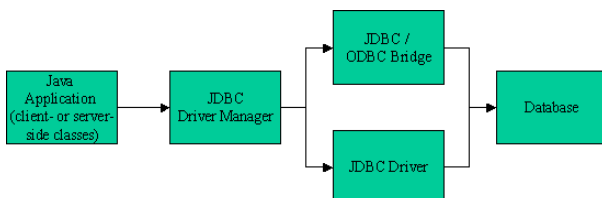


Figure 5 - Flow to SAS via JDBC

Figure 5 above shows how the process flow can occur for database connectivity.

Some databases cannot be connected to directly from JDBC as no direct driver is supplied. Such databases rely on the ODBC driver to communicate with a database using a bridge.

Many JDBC drivers are freely available from some database vendors. However, the drivers for SAS can be found attached to other software such as SAS/SHARE. The SAS/SHARE driver for JDBC enables applications to access and update any SAS data that is available through SAS/SHARE, from Java programs. The SAS/SHARE driver for JDBC can be used to create Java applets, applications, and servlets. The process of activating a flow between a client and the JDBC driver could occur through other standards mentioned in this paper such as HTTP or RMI.

### Example

The following example displays how to use the SAS/SHARE driver for JDBC. Firstly, make sure that the SAS/SHARE server is defined correctly (identify the port number it is listening on from the services file) and make sure the server is running. Secondly, set up the Java code. The JDBC driver classes accept a URL that identifies the SAS/SHARE server. The URL

is in the form  
jdbc:sharenet://hostname:portnumber.

'Hostname' is the name of the machine where the SAS/SHARE server is running and 'portnumber' is the port the SAS/SHARE server is configured to use. Many properties can be specified as connection arguments, such as username and password, but shall not be covered here.

The code below uses the JDBC driver classes to establish a connection to the SAS/SHARE server. After the connection is established, the driver provides SQL access to the SAS/SHARE server. The code sends SQL statements to the server and retrieve the results generated by those statements.

```
try {
    // Register JDBC Driver in jdbc riverManager
    java.sql.Driver driver = (java.sql.
        Driver)Class.forName("com.sas.net.
            sharenet.ShareNetDriver").newInstance();
    // Create connection
    java.sql.Connection connection = driver.
        connect("jdbc:sharenet://localhost:5010",
            null);

    java.sql.Statement st =
        connection.createStatement();
    java.sql.ResultSet rs =
        st.executeQuery("select * from
            sashelp.class");

    // Result set manipulation to get required
    // data.....
}
catch(Exception e) {
    e.printStackTrace();
}
```

An entire application, applet, or server-side application can be built around this code. Once the results set is retrieved from any SQL query passed through, it can be manipulated for display in many different ways. Localhost can be substituted for a server name or IP address.

### Advantages

With the correct drivers, JDBC usage can be relatively straightforward. SAS users often already have access to the necessary software, which can then be used from Java applications. They also often already have knowledge of SQL, used for the database querying, without having to learn any further coding techniques. As JDBC is an industry standard, examples of its use can be relatively easy to find.

### Disadvantages

Unfortunately, the drivers for SAS are not freely accessible and come wrapped up with other potentially expensive software. If business users do not already have need for and use SAS/SHARE, it can be an expensive piece of software for the drivers alone.

## SAS Batch Processing with RMI

### Probable Software Requirements

The SAS System on a server machine alone would be sufficient to use RMI technology for batch processing SAS programs as shown in the example below. A JDK needs to be installed on both the machines aiming to communicate for the various Java classes to run and use the necessary packages such as the 'java.rmi' package. A web server such as Apache would also be necessary.

This example assumes that both the client and server objects are written in Java (otherwise CORBA would have to be used to interface between different programming languages). It is also assumed that programs such as the object server and rmiregistry can be constantly running on the server machine. This server object must be 'alive' when a service is requested and be reachable through TCP/IP.

### Method

RMI is a standard used to communicate between two Java Virtual Machines via standard network protocols such as TCP/IP. For example a Java application running on a local

machine collects information from a user using a graphical user interface, sends that information to a server for processing, which then returns a response.

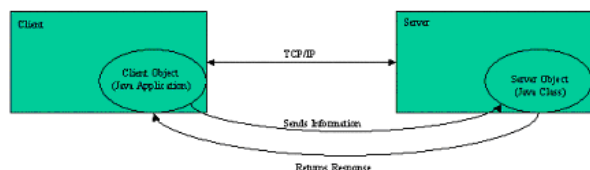


Figure 6 - Example Architecture Using RMI

For example, a local Java application (client object) allows the selection of a SAS program and passes that selection to the remote/server object which is programmed to run and process that code. The server object then sends a response (or error) back to the client which displays the returned information. RMI therefore ships parameters to another machine, runs a method on the remote machine (which was called from the client), and ships back a return value or exception.

#### Example

While providing code for all the different classes involved in establishing an RMI connection both on the client and the server machines would be too much to display here, the standards involved and many coding examples can be found in any good Java reference book. Once the framework is developed using such a reference, standard Java and SAS knowledge can then adapt the application for use with SAS.

The figure below shows an example of RMI built creating a small, Java, stand alone application to allow a user to remote submit SAS programs to a server machine. These programs could be pre-written programs selected from a list or manually written in a text area (displayed in the first tab). The log and output of the submitted program are then retrieved and displayed in further tabs on the application.



Figure 7 - Java Application to Remotely Run SAS Code

#### Advantages

RMI technology is a standard Java technology and as such is supplied with any recent JDK. This means that connections can be made to server machines running SAS technology with no additional cost. SAS programs can be run in batch, and the server based classes can even access results and send them back to the client as sets of objects.

#### Disadvantages

RMI can take time to research and set up correctly. There are several classes involved to establish the necessary connections between the client and server machines. Each class needs to be set up and placed correctly for the connections to work. The process can be a little time-consuming to begin with. However, once the stub classes have been generated and positioned correctly and the skeleton of the process is in place, any Java development can then proceed at a normal pace.

This example would not be recommended for large amounts of data as passing large sets of results back to the client can be resource intensive. Other methods and additional software can cope better with accessing large sets of data. However, if the server based SAS processing occurs to such an extent that only small summaries are needed to be displayed using the application, then this method could be quite effective.

#### Other Routes

Beyond the four methods assessed in this paper, other routes

into SAS exist using other specialist SAS software, such as SAS/IntrNet<sup>®</sup>, or new technologies, such as web services. Web services is a platform independent based technology based on a set of industry standards which provide basic functions via Internet protocols. This means that such solutions would be relatively inexpensive. Very generally, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response.

Furthermore, methods such as those discussed above can be used collectively. For example, RMI technology could be used to run SAS code to manipulate data and create results data sets while JDBC could then be used to access and retrieve those results. AppDev Studio in fact utilises technologies such as JDBC and RMI to build much of its operability.

#### CONCLUSION

Therefore, the use of specialist tools such as AppDev Studio is not the only way to use SAS from Java based applications. There are other routes provided by SAS as well as Java standards that can be adapted which can utilise SAS. It is important to note that there are alternatives, even if some of these may use SAS in more simplistic or roundabout ways.

#### TRADEMARK CITATION

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.

® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

#### CONTACT INFORMATION

We very much welcome your feedback, comments, and questions on this paper. Contact the author at:

Amadeus Software Ltd.  
Orchard Farm  
Witney Lane  
Leafield  
Oxfordshire  
OX29 9PG  
United Kingdom

[www.amadeus.co.uk](http://www.amadeus.co.uk)

[clare.nicklin@amadeus.co.uk](mailto:clare.nicklin@amadeus.co.uk)

<sup>1</sup> <http://java.sun.com>

<sup>2</sup> <http://httpd.apache.org>

<sup>3</sup> <http://jakarta.apache.org>