ABSTRACT
There are a number of technologies available from SAS Institute to integrate access to your SAS data into the development of a new or existing multi-tiered web application. These technologies are included in different SAS products and can be confusing for those evaluating them for the first time. In this paper, we outline how to proceed with building multi-tiered web applications to display your SAS data using Java and .NET frameworks. We then describe possible architectures, driver, and configuration choices, and the relative benefits of .NET versus Java. Finally, we compare the performance of alternative approaches for SAS data access from Java and .NET code.

INTRODUCTION
Imagine this scenario: you want to disseminate information from your SAS datasets across the web while allowing multiple concurrent users speedy access. You also need to build a solution that will integrate into your company’s web site. You have tried WebAF 2.0.3 and although it got you up and running quickly, it does not give you the performance you need. What is a programmer to do?

In this paper we describe strategies for building high demand multi-tiered web applications that access SAS datasets using either Microsoft’s .NET framework or Java’s J2EE. The resulting web sites must be capable of handling multiple concurrent user requests.

The paper describes two ways in which SAS can be accessed from .NET and J2EE applications using both SAS/SHARE and SAS/IOM. It provides examples of the usage of each technology. The paper also includes benchmarks of read times for datasets of varying number of observations for each combination of development platform and SAS server, followed by a discussion of the benchmark results.

MULTI-TIER WEB APPLICATIONS – WHAT ARE THEY AND WHO NEEDS THEM?
Multi-tiered web applications are made up of layers of software. A typical system will have four layers: a Web Client Tier, a Web Server Tier, a Business Logic Tier, and a Database Tier, as displayed in Figure 1.

Each one of the layers has a specific set of responsibilities. For example, the Web Client Tier (Web Browser) is responsible for displaying information to the user, collecting user input and sending it to the Web Server. The Web Server Tier handles requests from multiple clients and dispatches them to be processed by the Business Logic Tier. It also receives responses from the Business Logic Tier and dispatches them to the respective web client. The Business Tier processes individual user requests according to a set of business rules. The processing may involve invoking a database, a file server, a web service, or a different Web Server and assembling the data necessary to provide a response to the Web Client. Finally, the Database Tier is responsible for data storage and integrity.

A multi-tier web application uses this architecture to provide maximum flexibility to its software components. By separating data access from data presentation logic there is less coupling among software components. The architecture is also very scalable allowing the various components that make up each layer to be replaced as more processing power is needed. For instance, the web server and the database may be located in different hosts to boost overall performance.

An additional goal of this system architecture is to provide for the simple interoperation with other types of software systems. Multi-tier web applications are meant to interface with other web servers, web services, databases, file servers, and legacy systems by using multiple technology standards (e.g. MOM, CORBA, SOAP).

In general, multi-tiered web applications are most useful in situations where it is necessary to interoperable with other software systems, where content is integrated from a variety of sources, or
where it is necessary to deploy a highly flexible system that can be made to scale up to meet future demand.

WHAT ARE THE CHOICES?
Java’s J2EE and Microsoft’s .NET are the two most popular platforms for building multi-tiered web applications because of their industry-wide acceptance and because of the comprehensive set of services they provide.

J2EE (Java 2 Enterprise Edition) is a Sun Microsystems’s product that is based on a set of open technological standards centered around the Java programming language and its operating environment the Java Virtual Machine (JVM). As such, is not bound to a specific hardware platform which gives greater flexibility for deployment. J2EE’s open standards have fostered the creation of a large contingent of products that implement one or many of its host technologies (i.e. JSP, EJB, JMS). Some of these products are open-source efforts which means that you can get extensive libraries of Java software for free as long as you invest the time to incorporate them into your own development. J2EE is fully supported by SAS Institute with AppDev Studio and its component elements.

Microsoft’s .NET is a technology platform that strives to accomplish a number of things at once. At its simplest level, .NET is a set of frameworks that revamp the old APIs that are used to program MS Windows applications. The frameworks are supplemented with a Common Language Infrastructure (CLI) that allows programmers to use and combine code from any language that supports it such as VB.NET, C# or J#. The .NET platform is a documented standard that can potentially be ported to other non-Windows platforms.

Additionally, .NET introduces a number of libraries that provide server-side features like database access (ADO.NET) and page creation (ASP.NET). These services are part of Microsoft’s offerings for enterprise level computing and rely on the close integration of .NET technologies into other Microsoft server products (e.g. SQL Server 2000). .NET is also complemented by Microsoft’s push into web service technologies that strive to simplify the means of communication among large web systems.

Regardless of the choice, J2EE or .NET, both development platforms are fundamentally equivalent in terms of the services they provide for the development of multi-tier web applications. Each of them includes a comprehensive set of libraries that offers most of the services needed for application development. The main differences among J2EE and .NET tend to be along the lines of licensing and platform restrictions.

Since Java and the J2EE technologies are an open standard, source code for all the basic Java APIs is readily available. Also, there are competing implementation of many libraries allowing developers a choice of software vendor and the ability to mix and match components. It is also possible to obtain free implementation of most major APIs from the open source community. Finally, since Java is a cross-platform environment, there is also the ability to choose which platform to use for development and deployment. Another interesting feature is that it is quite easy to develop software in a platform (e.g. Windows 2000) and deploy the system in another (e.g. Solaris).

.NET is nominally an open standard but is largely controlled by Microsoft. Microsoft has made a substantial effort to remain the single supplier of technology components and tools for .NET software development. The downside is that most .NET development is likely to take place using Microsoft products and on the Windows platform. The upside of this is that since Microsoft controls the .NET standard it has managed to successfully integrate these technologies into their development tools making them both powerful and easy to use.

Either J2EE and .NET can be used to build web applications that access SAS datasets. Java’s J2EE web applications can be programmed using any of the JDBC (Java DataBase Connectivity) drivers provided by SAS Institute and .NET web applications can use any of the OLE/DB Data Providers available through SAS Institute. We will examine the OLE/DB Data Providers in the next section.

ACCESSING SAS FROM A .NET WEB APPLICATION
The SAS Institute provides three OLE/DB data providers to access SAS data from .NET programs:

- The Local data provider which enables read-only access to SAS datasets in the local machine.
- The SAS/SHARE data provider which provides access to datasets in the local server or a remote host running SAS/SHARE server.
- The SAS/IOM data provider which provides access to datasets in SAS Integrated Object Model (IOM) servers.

All of these OLE/DB data providers provide an abstraction of the SQL database query language over the SAS datasets. Once configured, all three data providers are used in the same fashion: building SQL statements and sending them to a SAS server for execution. When the SQL statements are query statements they generate result values which are returned to the client code. When the SQL statements are update, delete or insert statements, their execution returns a value indicating how many observations were affected by the statement. Additionally, the SAS/IOM data provider can execute commands that contain SAS code statements.

For the purpose of this paper we will only address the SAS/SHARE and SAS/IOM data providers. The local data provider only allows single user access to the local server and that access is limited to a read-only mode, which prevents it from becoming useful in any high-demand web application.
The SAS/SHARE data provider is meant to access a SAS/SHARE server running in either the local host or in a remote host. SAS/SHARE requires the existence of a working SAS/SHARE service.

The SAS/IOM data provider is meant to provide the client code access to a SAS IOM server. The IOM server can be running in the local host or in a remote host. If the IOM service is local, it is accessed as a COM component that launches the SAS server automatically. If the IOM server runs on a remote host, it is accessed as a DCOM remote service. The SAS/IOM data provider can also be used with the IOM Bridge to allow client code running on MS Windows to access a SAS/IOM server executing on a different (non-Windows) platform. In addition to the execution of SQL statements, the SAS/IOM data provider supports the execution of SAS statements on the server. This feature provides enormous flexibility to the types of applications that can be built with it.

USING THE SAS/IOM DATA PROVIDER – AN EXAMPLE

To read data from a dataset accessible to a SAS/IOM server using the SAS/IOM data provider, the client code must establish a connection to the service, create a command object, assign it the text of the SQL query statement that selects the needed data, execute the statement, and extract the data from the result object returned by the statement object. A code example that illustrates this sequence of steps follows. The example is written in C#, one of the languages supported by .NET and a close relative of the Java language (C# is also object oriented and has a Java-like syntax).

```csharp
// create connection to SAS/IOM data provider
string connectionString = "PROVIDER=sas.IOMProvider.1;" + "Data Source=LOCAL;";
OleDbConnection theIOMConnection = new OleDbConnection(connectionString);
theIOMConnection.Open();

// make a command to assign a libref to
// the directory containing the datasets
OleDbCommand theCommand = theIOMConnection.CreateCommand();
theCommand.CommandText = "libname mylib 'C:\SASProject\Datasets';";
theCommand.ExecuteNonQuery();

// make a SQL query to read all datasets
// in the libref 'mylib'
theCommand.CommandText = "select memname from dictionary.tables " + "where libname = 'mylib' order memname;"

OleDbDataReader reader = theCommand.ExecuteReader();

// add the datasets to the DSNameList
while(reader.Read())
{
    string libname = (string) reader["memname"];
    DSNameList.Items.Add(libname);
}
reader.Close();
theCommand.Dispose();
theIOMConnection.Close();
```

The preceding example creates a connection to the SAS/IOM server using an OleDbConnection object and a connection string. The connection string specifies the IOM data provider and the choice of a local IOM server. The program then issues a command for the SAS/IOM server to assign a libname named "mylib" with the path to the directory containing the datasets to be read. This step allows the code to send queries for data elements in datasets stored in that folder. The next step creates a new command string with a SQL query to request the names of all the datasets in "mylib". This command string is assigned to the same command object used previously. The command executes, and returns an OleDbDataReader object containing the values of all the records read. The names of the datasets are then iteratively extracted from the reader object and assigned to a list. Finally, the program closes or disposes of the reader, command and connection objects.

In a real-world application, the time cost associated with creating and disposing of a connection object is too high to repeat this process every time a group of records needs to be read. Instead, connections are pooled or they are cached in the session object associated to each user’s web session and reused multiple times.

The code above would work in a similar fashion for the SAS/SHARE data provider except that the syntax of the connection string would be different. Additionally, the SAS/SHARE data provider is incapable of issuing SAS commands such as defining a libname, so the libname definition needs to be set up in the SAS/SHARE server configuration.

BENCHMARKING THE SAS OLE/DB DATA PROVIDERS

We have two SAS OLE/DB data providers to choose from when building web applications under .NET. How do we decide which one is the right choice for our system development? One way to compare the data providers is by benchmarking them against a set of tasks. In our case, the most meaningful task to compare is reading observations from a dataset since this is an operation that is critical for most applications. We limit ourselves to benchmarking the reading performance of the data providers while ignoring the process of displaying the data over the web. This latter process depends on the web page assembly technology (e.g. ASP.NET libraries) and the delivery of content over the web (network latency) so it is not influenced by the choice of data provider.

For our benchmark, we created datasets made up of 3 variables and various numbers of observations ranging from 100 to 5000 observations each. We
then read each dataset 10 times with each SAS OLE/DB data provider, measuring the time taken from the point where the command is issued until the point where the last observation is read by the client code. The reading times for each data set were averaged and then graphed in the chart reproduced in Figure 2.

![Chart](chart.png)

Figure 2

From this simple benchmark we can conclude that reading data from SAS datasets is done considerably faster using the SAS/SHARE data provider than with the SAS/IOM data provider. This may be a consequence of the overhead imposed by the IOM object layer and the use of remote object calling technologies.

It is also clear that the difference in read performance increases as the number of observations read becomes larger. This has two practical consequences:

- If an application only reads a few observations at a time (e.g. a screenful at a time) the choice of data providers will not affect the application performance greatly.
- If an application allows users to download entire datasets at once (for example an access controlled, web-enabled repository of SAS datasets), choosing SAS/SHARE over SAS/IOM data provider will have a great impact in performance.

Using the SAS/SHARE data provider to gain efficiency does not need to be a limiting factor. It is possible to use both data providers from the same application to gain from the SAS/SHARE data provider’s efficiency while using the SAS/IOM data provider to issue SAS commands.

Additionally, the SAS OLE/DB code can also be used from stand-alone MS Windows applications without major changes. This provides a quick way to prototype and test the code that accesses SAS datasets. When the code is working, it can be moved to the web application.

ACCESSING SAS FROM A JAVA WEB APPLICATION

SAS Institute provides several different technologies to access SAS from a Java program. For instance, AppDev Studio includes WebAF and WebEIS which can be used to quickly build Java applications. WebAF includes a Java taglib that can be included in Java Server Pages (JSPs) to present SAS data. However, these technologies, as included in AppDev Studio 2.x, have shown slow performance.

An alternative SAS product to access SAS data from Java is the SAS JDBC (Java DataBase Connectivity) drivers. These drivers operate in a similar fashion to the SAS OLE/DB drivers discussed above by providing an object layer that abstracts the invocation of SQL commands and queries. This means that JDBC has object abstractions for a server connection, a command, and a set of results among others.

JDBC is a widely popular standard used by Java programmers. The obvious advantage of the SAS JDBC drivers to SAS users is its great efficiency in accessing SAS data.

There are three SAS JDBC drivers:

- SAS/Connect JDBC driver.
- SAS/SHARE JDBC driver.
- SAS/IOM JDBC driver.

In this paper we study only the SAS/SHARE and SAS/IOM JDBC drivers to establish a reasonable comparison with their OLE/DB data provider equivalents.

The SAS/SHARE JDBC driver allows SAS client code access to a SAS/SHARE driver either on the local host or a remote host. The SAS/IOM JDBC driver provides access to a SAS/IOM server but requires the use of an additional component the IOM Java bridge. The SAS/IOM JDBC driver can issue SAS commands just like the SAS/SHARE data provider.

Although the SAS JDBC drivers can be downloaded directly from the SAS Institute web site, the drivers only work with the Java Development Kit (JDK) version 1.3.0_01 which comes included with AppDev Studio. Therefore, you must license AppDev Studio to build applications that use the SAS JDBC drivers.

USING THE SAS/SHARE JDBC DRIVER – AN EXAMPLE

To read data from a SAS dataset in a SAS/SHARE server using the SAS/SHARE JDBC driver, the client code must perform the following steps:

- Create a connection object to access the SAS/SHARE server
- Create a statement object from the connection object
- Execute a SQL query using the statement object. The SQL query string specifies the data requested and the name of the dataset where it is
stored. The execution of the query returns a result set object that contains the requested data.

- Retrieve the data from the result set object.

The following example is a piece of Java code that reads all the observations from a dataset called “test0100” located in the libname “mylib”.

```java
// instantiate SAS/Share JDBC driver
Class dc =
Class.forName("com.sas.net.sharenet.ShareNetDriver");
java.sql.Driver driver = (Driver) dc.newInstance();
// specify libref for my datasets
Properties properties = new Properties();
properties.put("librefs", "MYLIB 'C:\SAS Project\Datasets'");
// create connection object
String url = "jdbc:sharenet://127.0.0.1:5420";
java.sql.Connection shareConnection =
driver.connect(url, properties);
// create a statement object
Statement statement = theConnection.createStatement();
// execute query
ResultSet resultSet = statement.executeQuery("select * from mylib.test0100");
// read results
while(resultSet.next()){
    double sValue1 = resultSet.getDouble("index");
    String sValue2 = resultSet.getString("id");
    double dValue = resultSet.getDouble("age");
}
resultSet.close();
statement.close();
shareConnection.close();
```

The code instantiates the SAS/SHARE JDBC driver and creates a new connection object using the driver instance. The connection is created using a URL that specifies the location of the SAS/SHARE server, and a properties object that can include a number of parameters. In this case, the properties object includes the definition of a libname (“mylib”) that will point to the folder containing our datasets.

With the connection object, we create a statement object which is then used to execute a SQL query. The query string specifies the selection of all the observations from the dataset “test0100” located in the library “mylib”. The execution of the query returns a result set object which contains the data from all the observations read. The next statements iterate over all the records in the result set reading them into other variables.

Finally, the result set, statement and connection objects are closed.

Since the SAS/IOM and SAS/SHARE JDBC drivers implement the same API, the code to access a dataset using the SAS/IOM JDBC driver is basically identical except that the set of properties necessary to open the connection object are different.

**BENCHMARKING THE SAS JDBC DRIVERS**

How do the SAS/IOM and SAS/SHARE JDBC drivers compare over the same benchmarks used on the SAS OLE/DB data providers? Figure 3 displays the relative performance of both JDBC drivers in our benchmarks.

As with the SAS OLE/DB data providers, the SAS/SHARE JDBC driver reads observations from a SAS dataset much faster than the SAS/IOM JDBC driver. Also, the difference is more pronounced as the number of observations being read increases. The conclusions from the SAS OLE/DB data provider benchmark also apply here: applications that read a few observations at a time may not notice the difference between drivers, and applications that download large numbers of observations at once will perform substantially better using the SAS/SHARE driver. Likewise, you can use both drivers together to gain from SAS/SHARE’s performance and SAS/IOM’s extra features.

**BENCHMARKING THE OLE/DB DATA PROVIDERS VS. THE JDBC DRIVERS**

A comparison of the benchmark data from the OLE/DB data providers and the JDBC drivers is presented in Figure 4:

The first thing to notice is that the SAS/IOM JDBC driver is substantially faster than the OLE/DB SAS/IOM data provider even though JDBC requires the use of an extra piece of software, the Java Spawner. For datasets with over 1000 observations,
the best performance is obtained using the SAS/SHARE OLE/DB data provider but if we focus on datasets with less than 1000 observations things look a bit different. See Figure 5:

![Figure 5](image)

It is clear that for small numbers of observations (<200) both Java JDBC drivers provide the best performance. However, it is important to note that raw performance alone should not be the only guide in choosing a platform for multi-tier web application development. Performance needs to be understood in the context of the business needs of each application scenario. Even more important, technology adoption has its own costs.

In the case of choosing between J2EE and .NET, there are number of factors to consider. Among them:

- Is either technology already familiar to the organization? Is there a large investment in either platform?
- What will be the licensing issues involved in picking either platform. If choosing Java, the licensing of technologies can be relatively inexpensive or pricey depending on the combination of vendors.
- What platform will be used for deployment?
- How hard will it be to configure the application? Multi-tier web applications are notable for their complex configuration issues. Throwing the SAS data providers, drivers, and services in the mix can only complicate this challenge.

CONCLUSIONS

It is possible to build high performance multi-tier web applications to access SAS datasets either using the J2EE or the .NET platform. SAS Institute provides access from both platforms to the SAS/SHARE and SAS/IOM servers using the corresponding JDBC drivers and OLE/DB data providers.

Our benchmarks indicate that the SAS/SHARE data provider and the SAS/SHARE JDBC driver provide faster read access to SAS datasets overall than their SAS/IOM counterparts.

This feature makes the SAS/SHARE JDBC driver and OLE/DB data provider especially useful in applications that read very large numbers of observations at once. However, the differences in read times among all JDBC drivers and data providers become smaller when reading a small number of observations.

The ability of the SAS/IOM JDBC driver and data provider to execute SAS code on a server gives them an edge when those features are necessary. It is possible to combine the performance of the SAS/SHARE driver and data provider with the SAS code execution feature of the SAS/IOM driver and data provider by using both types of JDBC driver or both types of OLE/DB data provider in the same application.

COPYRIGHT INFORMATION

SAS is a registered trademark of SAS Institute, Inc. in the USA and other countries. ® Indicates USA registration.

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

REFERENCES

The SAS OLE/DB cookbook. Contains info on SAS/IOM and SAS/SHARE data providers:

SAS/SHARE driver for JDBC:

Developing Java clients for SAS/IOM:

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:
MGC Data Services
5051 Millay Court
Carlsbad, CA 92008
(760) 804-5746
Ricardo@mgcdata.com
Miriam@mgcdata.com
www.mgcdata.com