Overview of Java™ Components and Applets in SAS/IntrNet™ Software

Barbara Walters, SAS Institute Inc., Cary, NC

Don Chapman, SAS Institute Inc., Cary, NC

Abstract

This paper describes the Java components and the sample applets that are part of SAS/IntrNet software. This paper describes how to use the SAS/SHARE*NET driver for JDBC, JConnect, and JTunnel to access information and services available from SAS servers, and also provides code examples to demonstrate these capabilities. Note that the code examples referenced in the text are presented in a separate section at the end of the paper.

Introduction

Java is the premier language for providing active content in Web applications. It is ideally suited to the Web because of its:

- portability across platforms,
- ability to maintain the integrity of the client machine,
- its ability to be dynamically downloaded on demand.

With the release of SAS/IntrNet software, version 1.1, SAS Institute provides production quality Java classes that access the power of SAS software through a Web browser or a Java application.

Web Technology and Thin Clients

Web browsers have revolutionized information technology by providing a simple point and click user interface to information sources located around the world. A wide variety of data formats, including HTML, image formats, audio and video formats are all presented through this single application. Users want to use a single application, i.e. the browser, to access all information sources; they are becoming more reluctant to install individual programs that access only a particular type of server or render a particular data format. Software vendors are challenged to provide access to both software services and data through the Web browser.

Because users are accustomed to accessing resources through a single application, there is increased acceptance of the idea of a *thin client* solution. A thin client solution consists of client/server software running on either thin client hardware, e.g. a network computer that has little more than a Web browser, or traditional hardware. Thin client hardware can be. a PC. In both cases programs are not permanently installed on the user's machine, but are downloaded on demand.

Thin client software is attractive because the user is not concerned with installing the particular software component; browser technology automates the process. The software often distributes work between the client machine and a remote server. Quite often the client provides support for the user interface, while most or all of the computation, data access, and other services are provided by the server running on a remote machine.

Java and ActiveX™

Both ActiveX components and Java applets provide thin client alternatives to traditional applications. The thin client user accesses a page that contains the information necessary to download the client program. The program is downloaded and starts running on the user's machine, giving the user access to the resources available on the server machine. The thin client machine requires no software other than a browser in order to access resources and services available on the network.

There are several notable differences between ActiveX components and Java applets. ActiveX components are usually written in C++ and are compiled for a specific platform. Once they are downloaded, the ActiveX component behaves as any C or C++ program and can access all resources available on the user's machine, including the local file system, other programs, and other machines on the network. The user can make the decision to not accept a particular ActiveX component and not install it on their machine, but once it is installed it has the full capabilities of any other software program.

In contrast, applets are written in Java, which is an interpreted language. This means that the applet code is portable and not restricted to running on a specific platform. In order to run a Java program, the Java Virtual Machine (JVM) must be available on the user's machine. Currently, browsers provided by

- Netscape (Navigator and Communicator),
- Microsoft (Internet Explorer) and
- Sun (HotJava)

all contain a JVM as part of their standard installation. If a user has one of those browsers installed, there is no additional software installation required to run Java applets. If the user needs to run stand-alone Java applications, a separate installation of the JVM may be required.

Java provides support for both dynamic class loading and a security manager. When a user accesses a web page that contains an applet tag, the Java classes that comprise the applet are dynamically downloaded to the user's machine and start executing. Because Java is interpreted, the JVM has the opportunity to enforce the rules specified by the security manager. It is the responsibility of the security manager to ensure that the integrity of the user's machine is maintained and that the applet does not have access to resources other than those the user has specifically granted.

What's New in Java Version 1.1

Sun Microsystems first released Java in mid-1995. Since that time, Java has undergone significant enhancements in functionality. The 1.1 version of Java was released by Sun Microsystems in the first quarter of 1997. At the time this paper was written, full production support of Java 1.1 was not available in either Netscape browsers or Internet Explorer.

In response to the delay of production-quality support for 1.1 by browser vendors, Sun has released a version of the JVM that acts as an HTML embedded object. This new product is called Java Activator. Java Activator provides support for the most recent production version of Java and ensures consistent behavior in the different browsers. This new product was released in December 1997 as part of Sun Microsystem's early access program and was not yet available in 1997 as production-quality.

The 1.1 version of Java includes support for:

- a component model, called JavaBeans™
- a standard archive format
- internationalization
- security enhancements, including "signed" classes
- object serialization
- printing support
- AWT (Abstract Window Toolkit) enhancements
- database access.

Detailed information about the contents of the 1.1 release can be found at the Java Web site:

http://java.sun.com

JavaBeans

JavaBeans is a specification that allows vendors to create Java components that can interact with components developed by other vendors. Many vendors are developing components that provide support for sophisticated user interfaces as well as components that access databases or compute servers.

Archive Files and Security

The standard Java archive (JAR) support addresses both download time issues and security issues. JAR support allows Java programmers to bundle Java classes together, which greatly reduces download time. In addition, it allows vendors to digitally sign the archive file. JAR files can contain digital signatures identifying the vendor that created the JAR file.

This digital signature, when used with the new security API, allows the user to decide whether they "trust" the software provided by this vendor. "Trusting" the vendor conditionally grants the vendor's programs access to the resources that belong to the user. The 1.1 version of Java enables signed applets. Future releases of Java will include more enhancements to the security API.

AWT Enhancements

In previous Java releases, AWT components would frequently not render or not behave in a consistent manner on different hardware/software architectures. The AWT enhancements allow creation of sophisticated components that render themselves consistently, regardless of platform. Other enhancements also include a new event model, improved cursor support and an API for print support of AWT components.

Future releases of Java will include a robust set of visual components as part of the Java Foundation Classes (JFC) JFC. Since these components will be included in a standard release of Java, this will greatly reduce download time.

JDBC ™

The Java Database Connectivity API (JDBC) is included in the 1.1 version. Similar to ODBC, it provides a standard interface for accessing databases through SQL (Standard Query Language) statements. The JDBC driver manager classes are included with the standard Java 1.1 release.

These new enhancements to Java enable vendors to provide full-feature applets and applications that take advantage of client/server computing. SAS/IntrNet software fully supports the JDBC API.

SAS/IntrNet Java Components

SAS/IntrNet software includes two Java packages:

- the SAS/SHARE*NET driver for JDBC
- JConnect.

All of the classes in these packages are written entirely in Java and can be dynamically downloaded from a Web server.

In addition, there is a Common Gateway Interface (CGI) program, JTunnel, which helps alleviate some of the configuration limitations of Java applets.

The JDBC driver allows Java clients, either applets or applications, to access data in SAS data sets or SAS views that are served from a SAS/SHARE server or a SAS/CONNECT server. A Java client that uses JDBC allows users to filter data, perform simple transformations of the data, and update data.

JConnect is a set of Java classes that allow a Java client to submit SAS statements or procedures and retrieve the results that were generated from those statements. The results can be textual output, graphics or new data sets.

These two packages allow Java clients to access much of the functionality of SAS. The Java client can be either an application or an applet. Before describing the components in detail, it is important that the reader understand the different types of SAS servers.

Overview of SAS Servers

The Java components interact with two different types of SAS servers:

- SAS/SHARE servers
- SAS/CONNECT servers.

All communication from a Java client to the SAS server is done through a socket connection. This means that the server machine must have a TCP/IP protocol stack installed. The JTunnel feature allows the Java client to use HTTP (HyperText Transfer Protocol) but, ultimately, those requests are sent on a socket to the SAS server.

Important differences between these two servers are described in the following sections.

SAS/SHARE Server

A SAS/SHARE server allows multiple clients to access data simultaneously. Clients can be

- SAS clients
- Java clients
- htmSQL
- or other clients implemented using the SAS SQL Library for C.

The SAS/SHARE*NET driver for JDBC uses the SQL services provided by the SAS/SHARE server.

An administrator is responsible for configuring, starting, and stopping the SAS/SHARE server. The administrator configures the port number where the server will receive client requests, sets the data access permissions for the server and sets the run mode of the server. The data access permissions determine what data on the server is accessible to clients that connect to the server. The run mode for a server can be set to either "secured" or "unsecured".

The default run mode for the server is "unsecured" mode. This mode allows any client to request services from the server. The client can request any data that is accessible to the server. Usually the administrator defines a set of libraries that are available from the server. However, a client can request that a particular library be defined for their use. This is described later in the section "Data Access through JDBC."

The server can also be run in "secured" mode which requires the clients to provide a user ID and password before the client can make requests of the server. The user ID must be a valid identity for the host where the SAS/SHARE server is running. With this configuration, the client can only access the data to which both the server and the client have permission. In Version 6 of SAS software, the user ID and password can be encrypted before being sent to the server. Encryption support is only provided for user ID and password; the data is returned to the client unencrypted. SAS software, Version 7 will provide encryption support for data.

After it is started, the server continues to run whether or not any clients are actively using it. Because the server is already running, a client can connect quickly to the server and send requests. When the client is finished it disconnects from the server. Disconnecting from a server does not stop the server. It continues to run until the administrator shuts it down. When the SAS/SHARE server is no longer needed, the administrator can shut it down.

SAS/CONNECT Server

A SAS/CONNECT server allows a client to send requests for remote execution of SAS statements and procedures, as well as requests for data. Data requests are made using JDBC method calls; only SQL access is provided. Unlike the SAS/SHARE server, a SAS/CONNECT server is started on behalf of an individual client and is shut down when that client is finished requesting services. Before a client can request services, it must first request that a SAS/CONNECT server be started.

For Java clients, the machine where SAS/CONNECT software is installed must have either a telnet daemon running or a copy of the SAS/CONNECT spawner running. For this paper, we refer to the telnet daemon or the spawner as the *bootstrap program*.

The Java client must configure the JConnect classes to identify the location of the bootstrap program. JConnect uses a "host" parameter to identify the machine where the SAS/CONNECT server is run; host is either the IP name or address of the server machine. It uses a "port" parameter to identify the port on which the bootstrap program receives requests. The default port value for either a telnet daemon or the SAS spawner is 23. If no port property is specified, the JConnect classes defaults to port 23.

The Java client sends a request to the bootstrap program to start a SAS/CONNECT server for its use. Because the server is started for an individual user, most hosts require the client to provide a user ID and password. This ID determines the identity used to start the SAS/CONNECT. Because this initial request must start the SAS/CONNECT server, it takes longer to complete than a request to the SAS/SHARE server. After the bootstrap program has successfully started the SAS/CONNECT server, the Java client can connect to the server and start sending requests.

SAS/CONNECT software provides scripts that are used by a SAS client to start the remote SAS/CONNECT server. JConnect does not use these scripts. Instead, the JConnect classes accept a property object that describes the prompts sent from the bootstrap program and the appropriate responses that should be returned. For example, the first prompt sent from the bootstrap program might be the string "login:". The first response should be the user ID used for the SAS/CONNECT server. A description of these properties is covered in depth in the JConnect documentation.

Data Access through JDBC™

The SAS/SHARE*NET driver for JDBC provides SQL access to data available in SAS data sets, SAS views or data that resides in a foreign database such as Oracle.

Through JDBC, Java clients can:

- define SAS libraries
- create or delete tables
- insert or delete columns
- read or update data
- perform simple transformations on data.

JDBC defines methods that provide information about the database itself. For example, there are methods to determine the libraries available to the server or the types of data supported by the server.

In order to use JDBC, the Java client must first create an instance of the JDBC driver. The JDBC driver manager uses the JDBC URL to determine what driver to use. By calling the connect method of the driver, the client can establish a connection to the server. The connect method uses the JDBC URL to determine where the server is running and which port it is using to receive requests. The URL takes the form:

jdbc:driver://host:port

The driver keyword is replaced with the driver name. To use the SAS/SHARE*NET driver replace "jdbc:driver" with "jdbc:sharenet". The host keyword is replaced with the IP name or address of the machine where the server is running. The port keyword is replaced with the port number that is used by the server. For example, SAS Institute has a SAS/SHARE server running on a machine available on the Internet. Its URL is:

jdbc:sharenet://192.35.86.10:5010

The connect method can also accept the user ID and password used to access the server (if one is required). If a user ID and password are required and have not been passed as either parameters to the connect method or included in the property object, the connect method throws an SQL exception that contains the message

"Userid/Password not accepted by server."

If the Java program catches the SQL exception and detects the above text in the exception message, it can prompt for the user to input a user ID and password and retry the connection. Other SQL exceptions may be thrown if the server is not available or if the host or port number were not specified correctly.

Other properties can be used by the driver to configure the connection. The *dbms* property can be set to tell the driver to access a foreign database (such as Oracle) utilizing the SAS/ACCESS software. The *libref* property is used to configure additional libraries.

A Connection object is returned from successful execution of the connect method. From the Connection object, the program can obtain DatabaseMetaData objects or Statement objects. The DatabaseMetaData objects have methods that return information about the database. For example, a

program can obtain a list of SAS libraries (SQL schemas) available at the server by calling the getSchemas() method of the DatabaseMetaData object. It can also obtain a list of data sets by calling the getTables() method of the DatabaseMetaData object. The code segment in Figure 1 shows how to create a connection to a SAS/SHARE server.

```
/*
 Establish a connection to the SAS/SHARE server
 running on the machine named server
 listening on port 5010
*/

java.sql.Driver driver = null;
try {
  driver = (java.sql.Driver)Class.forName
    ("com.sas.net.sharenet.ShareNetDriver").
    newInstance();
} catch (Exception e) {
  System.out.println("Class not found");
}

try {
  java.sql.Connection connection =
    driver.connect("jdbc:sharenet://server:5010",
    null);

  System.out.println("Connection complete");
} catch (SQLException sqlException) {

  String eMessage = sqlException.getMessage();
  if (eMessage.indexOf("Userid/Password not accepted
      by server.") != -1) {

/* Pop-up a logon dialog box to prompt user for a
   user ID and password and retry the connection
   using that information
*/
} else {
    /* Some other error occurred */
    System.out.println("Exception thrown: " +
      eMessage);
  }
}
```

Figure 1: Establish a connection to the server

A Statement object is returned from calling the getStatement() method of the Connection object. The Statement object provides methods to execute SQL statements or queries. For example, a program uses the execute() method to define a new library for its own use. The code segment in Figure 2 shows how to define the library "temp" that is located in the directory /usr/temp.

```
/*
   Assign a new library.
*/

try {
  java.sql.Statement statement =
    connection.createStatement();
  boolean result = statement.execute("libname
    temp '/usr/temp';");
} catch (SQLException e) {
  System.out.println("Exception thrown: "
    + e.getMessage();;
}
```

Figure 2: Assign a library

The executeQuery() method of the Statement object accepts an SQL select statement as a parameter. It returns a ResultSet object that contains the rows and columns that meet the criteria of the select statement.

The ResultSet object provides methods to access its contents. The next() method moves the pointer of the ResultSet object to the next record. When a ResultSet object is first created, the pointer is positioned to an invalid record. The next() method must be called to position the pointer to a valid record. After obtaining a valid record, the program can obtain the data from any column either by referring to the column by its index number or by its name.

The contents of the column are obtained through "get" methods. All data returned from the SAS/SHARE server can be obtained as a character string, regardless of the original type of the data. For example, the getString(1) method will return the data available in column 1 as a Java String. If the column contains numeric data, the methods getFloat() or getInt() can be used.

The data is sent from the server as the client program requests it. For example, if the program opened a dataset that contains 10 megabytes of information, only a small fraction of the data is actually downloaded to the client. As the client requests more data, additional requests are sent to the server to obtain additional data.

The code segment in Figure 3 shows how to create a ResultSet object and extract the data for column 1.

```
/*
   Create a ResultSet.
*/

String column;
try {
  java.sql.ResultSet resultset =
    statement.executeQuery("select * from
    temp.table");

while (resultset.next() == true) {
    column = resultset.getString(1);
    System.out.println("Column 1 = " + column);
  }
  resultset.close();
} catch (SQLException sqlException) {
  System.out.println("Exception thrown: " +
    sqlException.getMessage());
}
```

Figure 3: Create a ResultSet

Some JDBC drivers support cursors, which allow a program to "mark" a position in the ResultSet and return to that position. A ResultSet object obtained from the SAS/SHARE*NET driver for JDBC does not support cursors. The result set must be read starting from the first record, proceeding through each of the records. If the first 5 records have been read, there is no way to reread the first record without creating a new ResultSet object.

The data at the server can be modified using the executeUpdate() method of the Statement object. This method accepts an SQL UPDATE statement and updates ALL rows that match the update criteria. The executeUpdate() method returns the number of rows that were actually updated. There is no way to lock a single record and then update it using the SAS/SHARE*NET driver for JDBC.

If the program requires a guarantee that only a single record be updated and guarantee that data is not overwritten, the UPDATE statement must include a where clause that contains a unique index as well as the data as it was originally read. When the where clause includes the previously described information, the executeUpdate() method will return a count of zero if the data has been changed from the time it was originally read or if some other client has the record currently locked. The unique index guarantees that only a single record is updated.

The code segment in Figure 4 shows how to update all the rows in the salary data set that have values greater than 40000 in the salary column.

```
/*
   Update salary in all the rows where the value in
   the salary column is greater than 40000
*/

int numRowsUpdated = 0;

try {
  numRowsUpdated = statement.executeUpdate("update
    EMPDB.SALARY set salary = salary * 1.05 where
    salary > 40000");
} catch (SQLException sqlException) {

System.out.println("Exception thrown: " +
    sqlException.getMessage());
}
```

Figure 4: Update a dataset

Access to SAS Procedures and Output through JConnect

JConnect classes are responsible for both starting and shutting down a remote SAS/CONNECT server. As described in “Overview of SAS Servers,” the JConnect classes communicate with a bootstrap program to start the SAS/CONNECT server. The JConnect client classes (either TelnetConnectClient or TunneledConnectClient) require the IP name or address of the remote server machine and the port number the bootstrap program is listening on. These classes receive a series of prompts from the bootstrap program and send responses to these prompts in order to start the SAS/CONNECT server. These prompts and responses are passed to the JConnect classes in a java.util.Properties object.

After the SAS/CONNECT server has been started, the client program can send SAS statements to the server for execution. The results of the procedure can be output (text), datasets or a file. All of these can be retrieved and displayed by the Java client.

After a set of SAS statements have been submitted to the SAS/CONNECT server, the SAS log and output lines are returned to the client. The methods getListLines() and getLogLines() return the information sent from the server. Because the log and output are returned as ASCII text, they are be easily displayed in a text window. The code segment in Figure 5 is taken from the sample JConnect applet included with SAS/IntrNet software.

```
/* Get the output from PROC CONTENTS. The
   connection object has already been created
   as a com.sas.net.connect.TelnetConnectClient
   object
*/

try {
  connection.rsubmit("proc contents data=tmp.data;
                      run;");
  String lines = connection.getListLines();
  TextArea textWindow.setText(lines);

} catch (JConnectException exception) {
  System.out.println("JConnect Exception thrown: " +
    exception.getMessage());
}
```

Figure 5: Create and retrieve output

A new data set can be created as a result of the statements executed on the SAS/CONNECT server. This data set can be accessed using the JDBC Connection object. The JDBC Connection object is retrieved using methods available in the JConnect client classes.

Graphics produced by the SAS/CONNECT server can be presented in a Java program: either as a GIF image or as a VRML (Virtual Reality Markup Language) file. Either of these formats can be presented in a Java client program.

SAS/IntrNet software includes the GIF image driver as one of the Web Publishing tools. This driver can be used with graphics procedures to create GIF files. Output generated by graphics procedures can use the GIF image driver to produce a GIF file. PROC GREPLAY converts existing GRSEG's to a GIF file.

Included in SAS/IntrNet is a new graphics procedure, PROC GGRAF, which produces three dimensional, clickable graphs. These graphs are stored as VRML files. SAS/IntrNet includes a VRML browser written entirely in Java that can be used to render the VRML graphs. Unlike GIF images, the VRML graph can save information about the contents of the graph and produce clickable graphs. For example, a bar chart created in VRML can include information about the variables selected to produce a bar. The variables and their values can be retrieved by the Java client and used to generate another graph displaying more information about the variables.

The previous discussion described how to generate GIF files and VRML files on the server. In order to render them in a Java client program, the files must be downloaded from the server to the client. This download is accomplished using PROC DOWNLOAD. When the client submits PROC DOWNLOAD to the SAS/CONNECT server, the server responds by sending the file as a stream of bytes. This byte stream is stored in memory by the Java Connect client. The Java client requests this byte stream from the JConnect client classes by calling the method getDownloadData(fileName).

The code segment in Figure 6 shows how to create a GIF file from PROC GREPLAY and display it in the Java client.

```
/*
  GIF output wanted, so use the GIF graphics
  driver. Library and catalog are the location
  of the existing GRSEG and entry is the catalog
  entry name.
*/

code = new String(“filename _GRAFOUT
                    '/temp/tempfile';
                  goptions dev=gif
                  gsfname=_GRAFOUT
                  gsfmode=replace;
                  title;
                  proc greplay nofs
                  igout=library.catalog;
                  replay entry;
                  quit;
                  proc download

infile=_GRAFOUT
                  outfile=”_GRAFOUT”;
                  run;");
try {
  connection.rsubmit(code);
} catch (JConnectException exception) {
  System.out.println("Exception thrown: " +
    exception.getMessage());
}
byte[] downloadBytes = getDownloadData(connection,
  "_GRAFOUT");
```

Figure 6: Convert a GRSEG to GIF

The code segment in Figure 7 shows how to create a VRML file from PROC GGRAF and display it in the Java VRML browser.

```
/*
  Build 3d Pie Chart code for PROC GGRAF
*/

String code =
    new String("filename _GRAFOUT
                '/temp/tempfile';
               goptions gsfname= _GRAFOUT
               gsfmode=replace;
               proc ggraf INFO NOFORMAT
               data = library.temp;
               pie3d salary /;
               proc download
               infile=_GRAFOUT
               outfile="_GRAFOUT";
               run;");

try {
  connection.rsubmit(code);
} catch (JConnectException exception) {
  System.out.println("Exception thrown: "
        + exception.getMessage());
}

/*
  Get the downloaded file (VRML) and give
  it to our VRML Browser
*/

byte[] downloadBytes =
    getDownloadData(connection, "_GRAFOUT");
ByteArrayInputStream vrmlStream =
    new ByteArrayInputStream(downloadBytes);
com.sas.vrml.browser.VRMLBrowserbrowser =
    new VRMLBrowser();
browser.setSourceStream(vrmlStream);
```

Figure 7: Create a VRML file with PROC GGRAF

Figure 8 depicts components involved in sending a request from a Java client and obtaining the response from the SAS server.

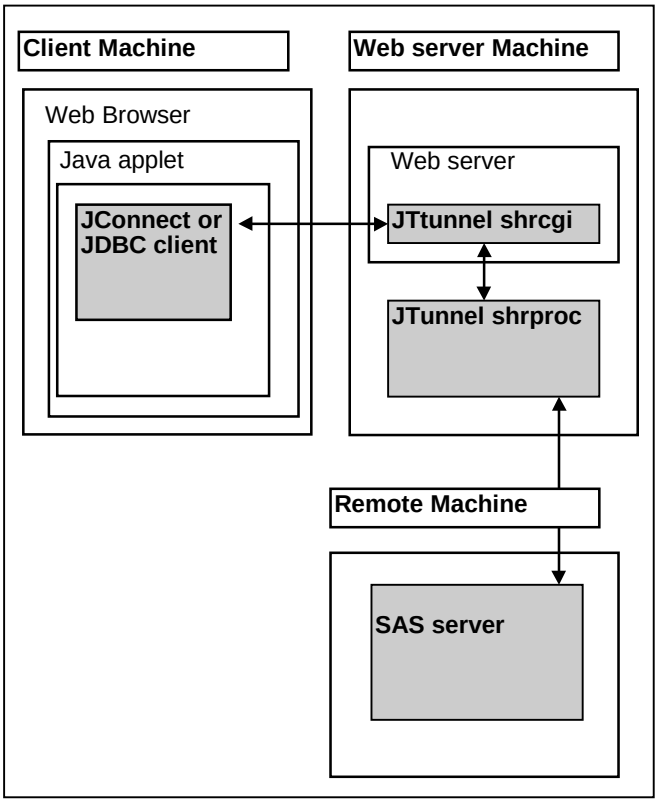

Figure 8: JTunnel Data Flow

JTunnel

The JTunnel program was created to help solve two problems. The first problem is that many users on the Internet access the Internet from behind a firewall. Many commercial firewalls allow HTTP traffic through the firewall, but not other types of traffic. Because access to SAS is through a socket connection, users behind firewalls are precluded from accessing the SAS/SHARE or SAS/CONNECT servers.

The second problem is the Java security manager restriction that unsigned applets can only establish socket connections to the machine from which they were downloaded. This restriction requires the SAS servers to run on the same machine as the Web server.

The JTunnel program is a CGI program that is installed on the Web server. It accepts HTTP requests, converts the request to something a SAS server understands then forwards the converted request to the appropriate SAS server.

Rather than establish a socket connection directly to the SAS server, the Java client sends an HTTP request to the Web server. The Web server invokes the JTunnel CGI program, which forwards the request to the SAS server. The CGI program captures the server's response and sends the response back to the client. The communication is performed using only HTTP protocol so the communication between the Java client and the SAS server is allowed through the firewall. Because the communication is between the Java client and the Web server machine, the Java security manager allows the communication.

Two programs run on the Web server machine. The first is the program that runs each time a CGI request is sent; that program is shrcgi. The first time a Java client sends a request to shrcgi, it starts another program that continues to run as long as the Java client requires its services. This second program is shrproc, and it establishes a socket connection with the SAS server.

Each time the Java client sends a request, a new copy of shrcgi is started. shrcgi forwards the request to shrproc. shrproc sends the request to the SAS server and collects the server's response. shrcgi sends the response back to the Java client and shrcgi stops running.

The JTunnel programs require a configuration file. This file contains a list of machines that the Java client is allowed to communicate with. This configuration file can be used to restrict access to the SAS servers. Not only does the configuration file limit the server machines that can be accessed, it also limits which ports are available for communication.

The configuration file also provides aliases for the command used to start the SAS/CONNECT server. It is quite possible that the administrator of the server does not want the command used to start SAS to be public knowledge. For this reason, an alias is used by the Java client. When the JTunnel program encounters the alias, it provides the proper substitution using the value of the alias found in the configuration file.

It should be noted that a request from the client sent through JTunnel to the server takes longer to complete than a request sent directly from the client to the server via a socket. For each JTunnel requests, a copy of the CGI program is started; starting this separate process on the Web server is not without a cost in performance. However, without JTunnel, many users behind firewalls are not be able to access the SAS server at all.

Issues Concerning Java

Some performance concerns with Java clients that should be mentioned. The first is download time. An applet may have JAR files that contain some or all of the classes used by the applet. The entire JAR file is downloaded before the applet starts to execute. If the JAR file is very large the download time is noticeable. JAR files provide a significant performance improvement compared to downloading classes one at a time, but it still takes time to download the JAR file. If a corporate intranet uses the SAS Java components in multiple applets, users could choose to install the Java classes on their local machine. In this case, the JAR file could be much smaller and only include the applet specific classes.

A high amount of variability exists between Java implementations. We have encountered many differences in behavior of the Java classes between the different JVM's distributed by JavaSoft, Microsoft and Netscape. Hopefully, JavaSoft's Activator and the release of a standard test suite for the JVM will help stabilize the Java environment.

Despite these issues, Java is important technology for the Web. We expect Java to mature significantly over the next two years and key to deploying Internet and intranet applications.

Summary

This paper provided an overview of Java, especially the enhancements included in Version 1.1 This paper described the Java classes provided by SAS Institute. We provide a package for data access, the SAS/SHARE*NET driver for JDBC, JConnect, which provides access to remote SAS computing services, and the VRML browser which can render 3D graphics. We provide Java classes and programs to support HTTP tunneling to alleviate some configuration restrictions and provide access to users behind a firewall. In addition, the paper contains code examples demonstrating how to create different types of output within SAS and retrieve that output for the Java client.

References

http://java.sun.com/products/jdbc/ - Use this URL to review JDBC information.

http://www.sas.com/rnd/web – This is the home page for SAS/IntrNet Product.

Data Visualization Using Java and VRML (Lingxiao Li, Art Barnes)

SAS, SAS/SHARE*NET, SAS/CONNECT, and SAS/IntrNet are registered trademarks of SAS Institute Inc. in the USA and other countries. Java and JDBC are registered trademarks or trademarks of Sun Microsystems, Inc. ® indicates USA registration.

Authors

Barbara Walters
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
(919) 677-8000 x6668
sasbbw@sas.com

Don Chapman
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
(919) 677-8000 x6707
sasdnc@sas.com

Acknowledgements

The authors would like to thank the following reviewers:

- Biff Beers
- Phil Herold
- Murali Srinivasan
- Renee Harper
- Tim Mattson