A SAS/IntrNet Java Program for Delivering Graphical Information of Remotely Monitored Processes.

Wing K Chan, The Pennsylvania State University, University Park, Pennsylvania
David C. Steven, The Pennsylvania State University, University Park, Pennsylvania

ABSTRACT
We demonstrate a SAS/IntrNet Java program that generates real-time, continuously updated graphs via SAS/Connect and the JTunnel feature. This tool can deliver graphical information via the web for a variety of remote monitoring or process control applications. For example, a manager may display a real-time control chart of any selected manufacturing process on his networked office (or home!) computer. We use an example originally conceived as a proposed application at Penn State University for delivering continuously updated analytic chemistry graphs from remote monitoring equipment located inside freshwater and marine fish environments at the University's Hetzel Union Building.

INTRODUCTION
The Java programming language has enjoyed growing popularity during the last several years. It is now widely used to build web-based applets, as well as stand-alone applications. Keeping up with the emerging technology, SAS Institute has included Java Tools as part of the SAS/Intrnet software. Users can now use two drivers to write Java programs: the SAS/SHARE driver for JDBC, and the SAS/CONNECT driver for Java.

The SAS/SHARE driver for JDBC allows users to access and update SAS data through Java programs. With the addition of a SAS/SHARE server and a SAS/ACCESS interface, the user may extend the reach of their Java applications to database management systems like Oracle and DB2. The SAS/SHARE driver for JDBC allows the users to access data interactively without having SAS installed on the client machines.

The SAS/CONNECT driver for Java is capable of sending SQL statements, to access and update SAS data.

DEMO APPLICATIONS AND CODE
Live working demonstrations and complete SAS code for this and similar SAS/IntrNet™ applications are available from:

http://cac.psu.edu/beatnic/papers/sugi/

Tunnel Feature
The performance of the two drivers mentioned above can be restricted by issues pertaining to security standards defined for Java applets. Because they are actual programs embedded on a web page, without security restrictions, malicious applets could potentially destroy data on a client computer. For this reason, all Java applets are allowed absolutely no access to the computer's hard drive. Furthermore, in a network environment, applets are allowed only to request information from the same computer from which they came.

These rules are enforced for the protection of client machines, but often they make communications difficult. For example, in many cases the web server and the SAS server are on different machines. Yet, an applet running on the web server must utilize resources on the SAS server to analyze the data. This poses a serious problem because applet security codes prohibit communication across two machines. Fortunately, the SAS/Intrnet software provides us with the Tunnel Feature to bypass this situation.

Figure 1.

This paper focuses on the SAS/Intrnet SAS/CONNECT driver for Java. The SAS/CONNECT driver for Java allows the user to write a Java program that starts a SAS session remotely, connects to that session, creates data sets, accesses existing SAS data, runs procedures to analyze SAS data, and retrieves the results. In addition to working with SAS statements, the SAS/CONNECT driver for Java is capable of sending SQL statements, to access and update SAS data.

The Tunnel Feature is installed on the web server, and it consists of a host of CGI programs. First, the web server collects information from the applet using HTTP (Hypertext Transfer Protocol). Then, it passes the information to the Message Router (shrccgi). If the requests are coming from an approved user and going to an

Source: http://www.sas.com/rnd/web/java/tunnel/howworks.html
approved port on an approved SAS server, the Message Router creates a detached process called the Session Agent (shrproc), which communicates with the SAS server machine. When the processing is finished, the results are first sent to the Session Agent. Then the Session Agent relays them to the Message Router, which then returns them back to the Java applet. By going through this process, the user can successfully bypass the security restrictions of Java applets and access the necessary resources on a remote machine. The entire process is depicted graphically in Figure 1 (above).

HOW THE APPLET WORKS

The fully loaded web page with the initialized applet is depicted in Figure 2 (below). When the user loads the web page initially, s/he is required to supply a valid username and password to gain access to the system. Once the identity of the user has been verified, s/he may begin submitting SAS statements for processing. For simplicity, the SAS commands in this applet have been hard-coded within the Java program. Therefore, after the server checks the username and password, a prompt appears in the Status bar requesting the user to select the SUBMIT button to start the SAS processing. When the user selects this button, s/he activates the rest of the Java program, which submits the SAS statements, retrieves the results, displays the results, and continues the process until the user leaves the web page. The Appendix depicts how different methods in the Java program fit together and how they respond to the user inputs.

DISSECTING THE PROGRAM

How to connect to the SAS server

The first step to build an SAS-Java applet that graphs real-time data is to connect to the SAS server. The tunnel feature is used to achieve this step. In this application, several methods in the Java program, as well as the HTML coding for the web page, play a role in building the tunnel feature. We will look at each item separately.

Figure 2.

HTML codes

The HTML coding can be crucial in constructing the tunnel feature. In particular, the contents in the <param> tag deserves some attention. First of all, the tunnel feature requires a router. This router is a CGI program that will eventually direct the traffic between the web server and the SAS server. The programmer can specify the router with a <param name=routerUrl value="yourCGI.cgi"> tag. In addition to the router requirement, the developer of the applet should also include three other <param> tags in the HTML codes. These tags represent the property values used by a SAS/CONNECT driver for Java program in connecting to a telnet daemon (or the spawner). They map directly to the telnet connection information that is passed when manually establishing a remote SAS session via telnet. For example, assuming the followings are the prompts that appear in a manual SAS connection via telnet (with the prompts in boldface),

Trying...
Connected to armstrong.cac.psu.edu.
Escape character is '^[].'
SunOS 5.6

login: wkcl10
Password:

Last login: Wed Sep 15 09:50:11 from vailima.cac.psu.
Sun Microsystems Inc. SunOS 5.6  Generic August 1997
You have new mail.

armstrong[1]:

then the complete Applet tag for the HTML is show could be:

<applet codebase=.
code=com.sas.net.connect.samples.threadGraph.class
height=450 width=700>
<param name=routerUrl value="http://cac.psu.edu/cgi-bin/shrcgi.cgi">
<param name=prompt1 value="in:">
<param name=prompt2 value="word:">
<param name=prompt3 value="[1]:">
</applet>

I would like to give special thanks to Jim Vuicic, who gave me numerous advice and hints on how to figure out Java.
The HTML <param> tags values are read into the Java program in the readParameter() method. In additional to the values from the web page (which are prompts), the readParameter() method also reads in the response to those prompts. Finally, it also puts a value called SASPortTag into the Java program's "memory". The SASPortTag uniquely identifies that the SAS session has completed initialization and allows the connection client to parse the port value from the message. In summary, the Java program now has eight different values stored in its property object: three prompts from the HTML page, three responses to those prompts, the router URL, and the SASPortTag. The code for the readParameters() method is:

```java
public void readParameters()
{
    String temp = getParameter("routerUrl");
    info.put("routerUrl", temp);
    temp = getParameter("prompt1");
    info.put("prompt1", temp);
    temp = getParameter("prompt2");
    info.put("prompt2", temp);
    temp = getParameter("prompt3");
    info.put("prompt3", temp);
    String temp = username.getText();
    info.put("prompt1", temp);
    info.put("prompt2", temp);
    info.put("prompt3", $mySasCommand");
    info.put("sasPortTag", "PORT=");
}
```

The connectThread and Connect()

When all of the parameter values are entered into the Java program, it is ready to establish the connection to the SAS server. First, a thread is started. Threads are necessary in this situation because it allows us to do more than one thing at a time. The connect thread is started in the ActionPerformed() method. When the user puts the username and password in the designated fields, s/he selects the submit button. When the Java program receives the command, it validates that all the necessary information is indeed supplied, and then starts the connect thread. The code that accomplish this task is shown below:

```java
if ((temp.length() !=0) && (temp2.length() !=0) &&
(connected==false))
{
    usernameIn = true;
    if ((connectThread == null) && (usernameIn ==
true))
    {
        connectThread = new Thread(this);
        connectCommand=1;
        connectThread.start();
    }
}
```

After the connect thread is started, the connect() method is called to formally establish connection to the SAS server. The code for accomplishing this is:

```java
synchronized void connect()
{
    try
    {
        status.setText("Logging onto:" + host);
        connection = (ConnectClient) new
TunneledConnectClient(info);
        connection.connect(host,port);
        jdbcConnection = connection.getSharenet();
        status.setText("push SUBMIT to run sample
production");
        connected=true;
    }
```
synchronized void submitLines() {
    try {
        status.setText("submitting SAS statements ...");
        connection.rsubmit(lines);
        gettingImage = true;
        status.setText("Execution complete");

        im= getImage(getCodeBase(), "gplot.gif");
        MediaTracker tracker;
        tracker = new MediaTracker(this);
        tracker.addImage(im, 1);
        try {
            tracker.waitForAll();
        } catch (InterruptedException e) {
        }
        repaint();
        count++;
    } catch (ConnectException se) {
        status.setText("Error: "+ se.getMessage());
    }
}

The update() method

After the submitLines() method successfully retrieved the results, it calls the update() method to have the graph painted. Usually, Java programmers use the paint() method to paint a picture on the applet. However, because we want a visual impression of continuous, smooth transition of real-time graph, we will be using the update() method instead. If we were to use the paint() method, there will be “flashes” between two graphs as the old graph is deleted from the applet and the new one painted on. With the update method, the new graph is superimposed onto the old ones, thus eliminating the “flashes,” or gaps during the transition. The codes for the update method are shown below.

public void update(Graphics g) {
    if (gettingImage == true) {
        g.drawImage(im, 10, 10, 680, 350, this);
        im= null;
        gettingImage = false;
    }
}

CONCLUSION

We have seen how a Java applet can utilize SAS to interactive draw graphs with real-time data. In particular, we have explored the power of and some requirements for the SAS/CONNECT driver for Java, and the Tunnel Feature. Also, we have looked at each method individually, and saw how they worked. The complete codes for the entire application is included in the appendix. Figure 3 (below) shows what the completed applet in production.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

David C. Steven
Graduate Education and Research Services
The Center for Academic Computing
The Pennsylvania State University
University Park, PA 16802
+1 814 865 0980
+1 814 863 7049
dcsteven@psu.edu
http://cac.psu.edu/gears/
APPENDIX

Init() method
[Builds the interface]

ReadParameter()

Applet Initialized & running

Connect Thread started
Run() Called

Connect() called and makes connection to SAS server

Check if offline

User enters name and password

Check if online

Submit Thread started, run() called

Called submitlines(), and sends SAS statements. Also receives the image after the processing

Calls update() and paints the graph on the screen

User pushes submit and sends SAS statements