Choosing a Method for Connecting Java to the SAS® System Across the Internet - CGI, JDBC or Socket?

Larry Hoyle
Institute for Public Policy and Business Research
University of Kansas

The SAS® system can communicate with Java™ applets across the internet (or within an intranet) in a number of ways. An applet can emulate an HTML form and connect to a SAS program via a CGI script. An applet can use the Java Database Connectivity (JDBC™) API to connect to a SAS/SHARE*NET® server. Applets can also open sockets to SAS programs using the SAS System's socket file access method. This paper explores the strengths and limitations of these methods and shows their use in IPPBR's "Statistics Kansas" WWW pages.

Java & the SAS System

Java is a programming language, based on C++, with features which make it well suited for use with the Internet. Java programs designed to run under a World Wide Web browser are called applets. Java applet clients can communicate with a SAS System server in at least three ways.

Common Gateway Interface (CGI)

First, an HTML form can invoke a remote SAS program through a Common Gateway Interface (CGI) script. The SAS program can set up for, and return a pointer to a Java Applet. The browser will then start the applet which may, in turn, read additional data from the server. The CGI component of the SAS IntrNet product[1] will also allow this approach to starting an applet.

An example, with source code, of launching a Java applet from SAS CGI script program can be found at:

http://www.ukans.edu/cwis/units/IPPBR/ksdata/ksah/javamap.htm

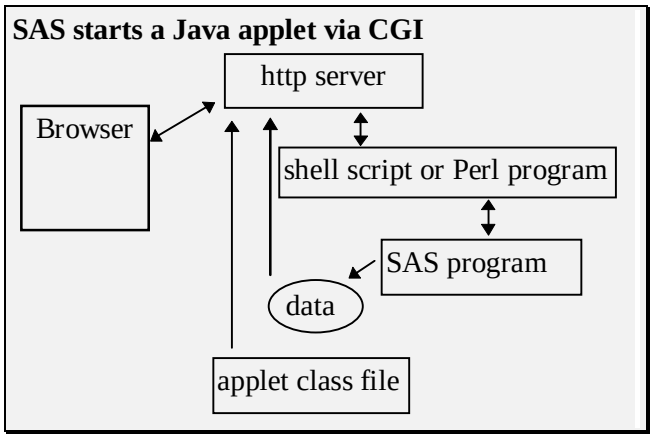
[1] This product did not have its final name when this article went to press.

The CGI method has some disadvantages.

- Once the applet starts it cannot communicate with the SAS program which launched it.
- Unless you are using the CGI component of the SAS IntrNet product, each time a connection is opened the SAS system must be loaded on the server, causing some delay.
- Unless you are using the CGI component of the SAS IntrNet product, the server system may be vulnerable to overloading if a large number of simultaneous hits are received - each one starting a separate SAS session.
- The SAS server program must write temporary files as part of the communication process. These may require some maintenance if automatic cleanup fails.

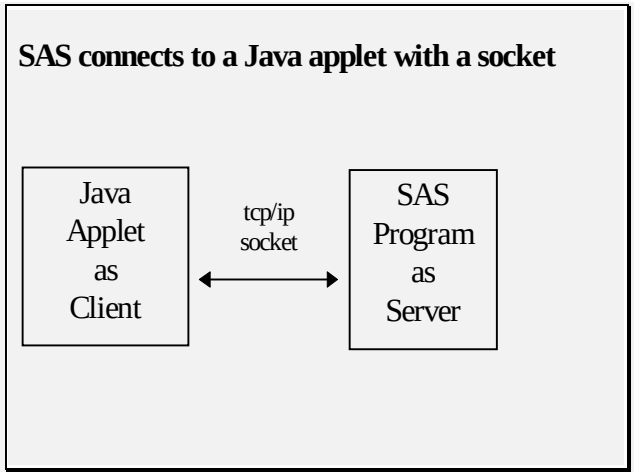
The method's advantages are:

- It relies on a standard protocol between the HTML forms and the CGI script which is managed by the server's http server.
- The SAS program can start the applet with pointers to text, graphics, or other format files.
- The client applet doesn't need a lot of downloaded classes to deal with communication.



Tcp/ip Sockets

A second technique uses tcp/ip sockets between a Java applet and a SAS program. This could also involve an intermediary Perl or C program on the server although an intermediary is not necessary.



Disadvantages of this method include:

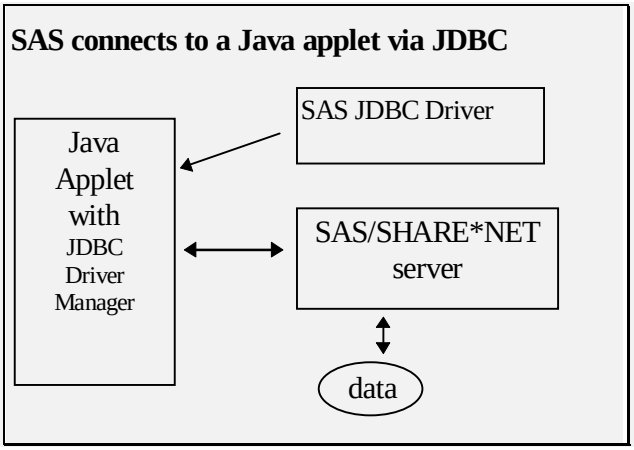
- Much code may have to be written on the server end to support any complicated interaction.
- The developer must design a communication protocol.
- SAS Data steps are not designed to be event driven procedures.
- The SAS socket access method does not have an "accept" function or statement. Data steps will not wait for a connection for output - only for input. The work around is to use a socket first for input to SAS and then for output from SAS.
- A high hit rate may keep users from being connected - the server is not multi-threaded.

Advantages include:

- The SAS server program can respond with text, graphics, or other format files.
- The connection stays open and can be two way. The applet may receive or post data as a result of user actions.
- Response from the SAS system can be quick - as no startup is involved.
- The server program is just one process. Huge numbers of hits may have a more manageable impact on the system if each one does not start a process.

Java Database Connectivity (JDBC)

A third technique uses the JDBC interface to communicate with a SAS/SHARE*NET server. The SHARE server is a special SAS procedure which allows multiple programs to have simultaneous access to data sets and views. It typically runs all the time. SHARE*NET is a special license for that procedure which allows non SAS software access to the server across the network. The JDBC interface communicates via SQL statements and tables.



As with the preceding techniques there are disadvantages:

- Communication is limited to that which can be represented in SQL statements and tables.
- The client must download JDBC driver code which can cause some delay.
- A high hit rate may keep users from being connected - the server is not multi-threaded.

and advantages:

- The connection stays open and is two way. The applet may send requests or data as a result of user actions.
- The SAS/SHARE*NET server runs all the time. There is no wait for the SAS system to load and response can be brisk.
- The use of SQL queries greatly simplifies the effort to develop an applet.
- The server program is just one process. Huge numbers of hits may have a more manageable impact on the system system if each one does not start a process.

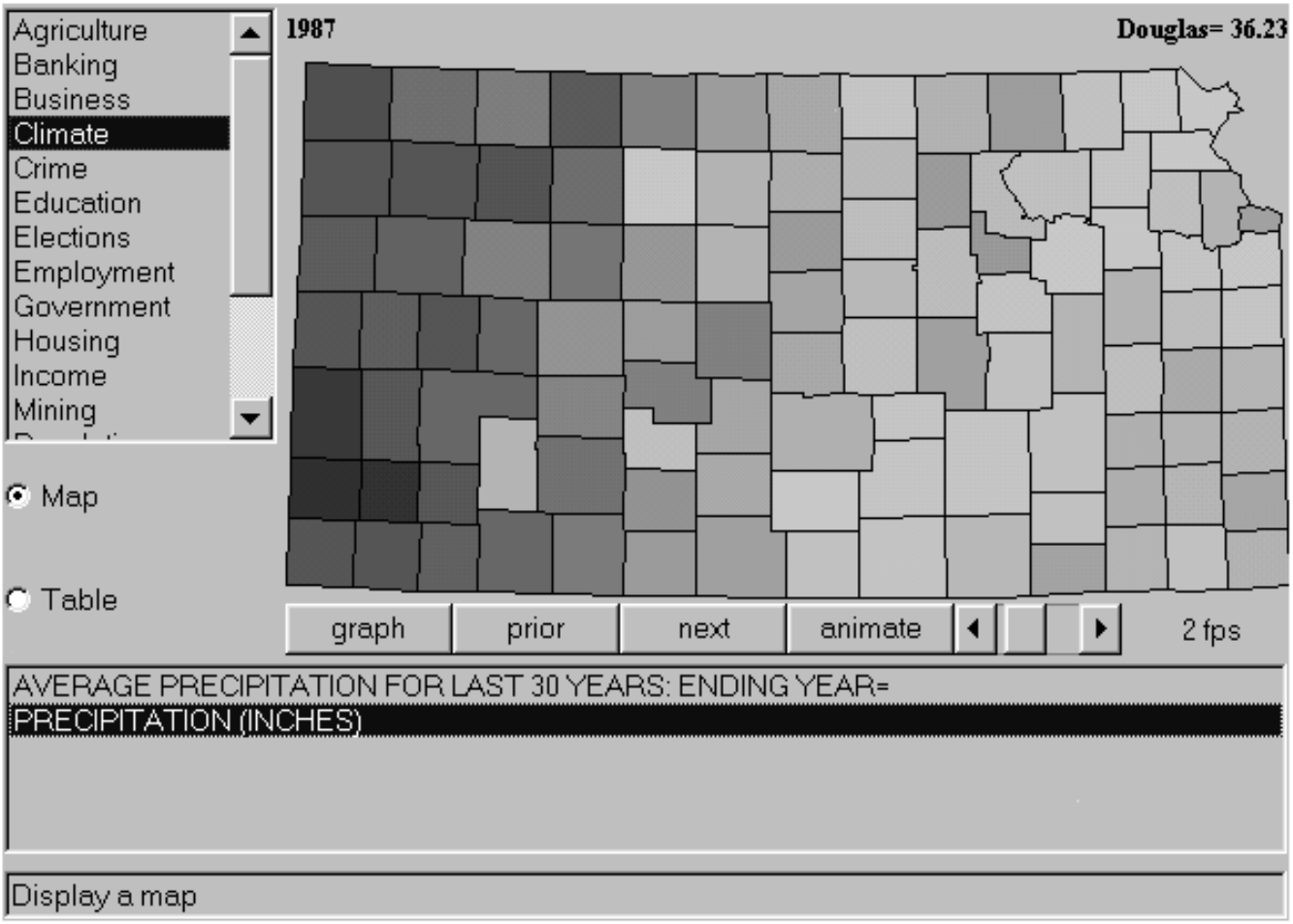


figure 1

JdbcMap - an example

The applet screen shown in figure 1 illustrates the advantage of open communication between an applet and the SHARE*NET server. The selection box at the upper left of the screen contains a list generated by an SQL query which allows a user to select a topic area - e.g. climate. Each time the user selects a topic, another SQL query to the remote server brings back a list of variables to be shown in the bottom box. Selecting a variable sends yet another SQL query against a different data set. The resultant SQL table can be displayed in tabular form or as a map.

Once the applet is started, many tables and maps can be displayed. While the CGI script method could be used to implement this interactivity, it would be awkward and slower - especially for the small queries like getting the list of variables. Here, the list of variables is also always current since it's not in an HTML form.

JDBC - getting started

Sun has developed a package of classes and interfaces named "java.sql" to implement JDBC. SAS Institute has developed drivers which connect the java.sql objects to SAS/SHARE*NET.

To use JDBC you must have the *java.sql* classes on your client machine and the SAS JDBC driver on the host, available to be downloaded by the client. The SAS JDBC driver comes with instructions for setting these up. Java.sql will ship with future browsers.

The most important interface elements are:

- Connection - which allows opening a connection to a database.
- Statement - which is used for executing a SQL statement.
- ResultSet - each Statement is associated with a ResultSet object which gives access to the result of the SQL query.

Using JDBC, some details

The applet shown in figure 1 uses a SQL query to build the scrollable box at the top left (a List Box object). This box contains a list of all the unique subject areas for which there are data available in our Kansas county database.The applet is invoked as follows:

```
<applet code="jdbcMap.class" width=600 height=425>
<param name=title value="KSAH">
<param name=url
value="jdbc:sharenet://lark.cc.ukans.edu:10403">
<param name=columns value="3">
<param name=saslibrary value="ksah">
<param name=oneval value="ksah.oneval">
<param name=varlist value="ksah.varlist">
<param name=rows value="10">
<param name="select" value="select * from
ksah.varlist">
<param name=bounds value=dimejav.prn>
</applet>
```

The applet must first establish a connection. It does so with code that looks like:

```
url = getParameter("url");

connection = driver.connect(url, properties);
```

The url string is retrieved from the parameter specified in the applet tag. It begins with "jdbc:sharenet" and then contains the address and port on which the SHARE*NET server is listening.

Once a connection has been established, submitting a query is quite simple. The Java statements below show how the subject list is retrieved in the jdbcMap application.

```
String Stmt =
  "select distinct subject from ksah.varlist";
statement = connection.createStatement();
resultset = statement.executeQuery(Stmt);
```

An applet builds an SQL query statement as a string. In this case the statement selects the distinct values of the variable "subject" from the data set "ksah.varlist".
The applet then uses the method *connection.CreateStatement()* to create a statement object. That object's *statement.executeQuery()* method sends the SQL statement to the remote SHARE*NET server. This method returns the result of the query in a special object called a ResultSet. The ResultSet can be read sequentially by the applet a row at a time. Information about the ResultSet's columns is available in an associated ResultSetMetaData object.

The Java method in the following table shows how an applet can go through a ResultSet. In this case it returns a List object from the items in one column. The *ResultSet.next()* method moves the cursor to the next row of the ResultSet. The *ResultSet.getString(int I)* method returns the item in column I as a String object.

```
// getList gets a list of the items in
//column cN from the ResultSet rS

  public List getList(ResultSet rS, int cN){
  List lst = new List(10,false);

    try{
       while(rS.next()){
         lst.addItem(rS.getString(cN));
       }
    } catch (SQLException e) {
      System.out.println("couldn't get column");
    }

  return lst;
}
```

For More Details

The GenericJDBC applet from SAS Institute shows using the JDBC interface in a complete application. It includes the use of threads to allow the communication to procede independently from the user interface. It is included with the driver package and can be found at:
http://www.sas.com/rnd/web/jdbc.html

The JdbcMap example is located at:
http://lark.cc.ukans.edu/~lhoyle/sasjdbc/varlist.html
your browser must have the java.sql classes available to view it.

Sockets

In many cases the limitation of JDBC to SQL queries will not be a problem. There are situations, though, where a more flexible communication method may be desirable - perhaps in combination with JDBC.

Imagine extending the sample application in figure 1 to allow the selection of the state or a combination of states. While the boundaries could just be extracted from a national county level file with an SQL query, the cartographic projection for some counties might be undesirable. Doing the projection on the client in interpreted Java would also not be ideal. Instead, the applet could open a socket to a SAS program running on the server and pass it the set of states desired. The SAS program would run PROC GPROJECT on the extract and then pass the boundaries back to the applet.

clicks.java

The "clicks" applet is a simple example of a Java applet which communicates with a SAS program through the socket access method. This applet, *clicks.java*, sends the *x,y* coordinates of each Mousedown event to a SAS server program and then reads the sum of x and y back from the SAS program.

The clicks.java applet consists of some definitions and initializations and two methods. - *mouseDown* and *paint*.

The mouseDown method, which is called by the browser when the mouse button is down, opens a socket to a SAS server program, writes the x & y coordinates of the mouse click, reads their sum, causes the screen to be repainted, and then closes the socket. All of the real work of the applet is done in the mouseDown method.

The paint method just echoes x, y and their sum along with the address of the remote server to which the applet connects.

Clicks would be improved by multiple threads separating socket I/O from user interface handling - but then it wouldn't all fit on a page.

clicks.java

```
/* clicks - L.Hoyle December 1996 - writes to a socket */
/* then reads from it. */

import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.awt.Graphics;
import java.awt.Event;
import java.io.DataInputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.io.IOException;

public class clicks extends java.applet.Applet {
String locHost;
String remHost = "lark.cc.ukans.edu";
String sasSays;
int port = 5050;
int lastX=1;
int lastY=1;
                /* ********************************          */
                /* A mousedown causes the socket to   */
                /* open, writes an x,y pair and reads     */
                /* their sum.                                 */
                /* ********************************          */

public boolean mouseDown(Event evt, int x, int y){
  try{

   Socket mySock = new Socket(remHost,port);
    PrintStream  psSock =
      new PrintStream(mySock.getOutputStream(),true);
    lastX=x;
    lastY=y;
    psSock.println(""+x+"  "+y);
    DataInputStream DISServer =
      new DataInputStream(mySock.getInputStream());
    sasSays = DISServer.readLine();
    repaint();
   mySock.close();

   }catch (Exception e){
         System.out.println(e.toString());
         showStatus(e.toString());
   }
   return true;
} /* mouseDown */

                /* ********************************* */
                /* Paint echoes the x,y value and  */
                /* their sum.                                */
                /* ********************************* */

public void paint(Graphics g) {
   g.drawString("Remote host is: "+remHost,10,10);
   g.drawString("Click in the box to send a value.",10,30);
   g.drawString("Last: x="+lastX+" y="+lastY,10,50);
   g.drawString("SAS responds: "+sasSays,10,70);
}
} /* class clicks */
```

Clicks.sas

The SAS server program, *clicks.sas* archives the x,y data and refreshes the *clicks.gif* file which contains a histogram of all of the x coordinates ever selected.

It begins with a section which defines a libname and a filename and sets graphics options.

The ongoing work of the server is done in a looping macro named *serve*. This section first starts a socket server listening on port 5050. Because the *filename fromjava* statement contains the keyword *server*, the program waits at the *input x y* statement until a client program connects to port 5050 of the server machine.

The server then reads x and y. The client program (clicks.java) must begin its interaction with the server by writing a record with two numeric values separated by a space.

The clicks.sas program saves these values to the data set named tcpip and then writes their sum back to the same socket connection from which it read x and y.

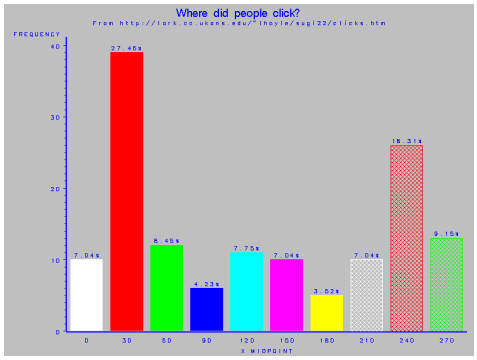
The server program quits if the sum of x and y is greater than 500. In this example, the applet tag in the HTML file presented to the public would have the applet window sized so that a user would not be able to send a sum larger than 500.

An administrator would have an HTML file with an applet window sized to allow termination of the server. A curious or mischievous person, of course could modify a copy of the HTML file and shut down the server. This highlights the limitation of having to implement your own protocol for a socket service.

```
/* clicks.sas - Larry Hoyle, IPPBR, Univ. of Kansas, Dec. 1996  */
/* a socket server which reads x and y                          */
/* from 1 record on port 5050                                   */
/* then writes the sum back to the same socket                  */
/* then generates a histogram of the x values                   */
/* ************************************************************ */

libname sugi22 '/homea/lhoyle/sugi22';
filename xhist '/homea/lhoyle/public_html/sugi22/clicks.gif';

                /* ********************************      */
                /* the graphics options don't change  */
                /* ********************************      */

goptions reset=(axis, legend, pattern, symbol, title, footnote)
        norotate hpos=0 vpos=0 htext= ftext= ctext=
        target= gaccess= gsfmode= ;
goptions device=imggif gsfmode=replace gsfname=xhist
        ctext=blue cback=ligr
        graphrc interpol=join;

title1  "Where did people click?";
title2  "From http://lark.cc.ukans.edu/~lhoyle/sugi22/clicks.htm";

pattern1 value=SOLID;
axis1  color=blue  width=2.0;
axis2  color=blue  width=2.0;
axis3  color=blue  width=2.0;

                /* ********************************   */
                /* the macro determines how long */
                /* the server runs.                  */
                /* ********************************  */

%macro serve;
%DO I = 1 %TO 10;

                /* ********************************* */
                /* read and write from the socket */
                /* ********************************* */

filename fromjava socket ':5050' server reconn=0;

data tcpip;
   infile fromjava eov=v;
   input x  y;
   put x= y=;
   output;
   file fromjava;
   xsum=x+y;
   put xsum;
   if xsum < 500 then stop;    /* stop ends this transaction */
   else abort return;          /* abort stops the server */
run;
```

After collecting and echoing the x,y transaction, the clicks.sas program archives the x,y pair into a data set named sugi22.xy.

It then uses PROC GCHART to create a histogram of all the x values ever entered into the archive file. This graph is written to a GIF file which is referenced in the HTML file which launched the applet.



clicks.sas (continued)

```
                /* ********************************** */
                /* archive data from the socket    */
                /* ********************************** */

proc sql;
 insert into sugi22.xy
  select x,y from tcpip;
                /* ********************************** */
                /* recreate the histogram of x clicks */
                /* ********************************** */

proc gchart data=SUGI22.XY;
   vbar X /
      percent    maxis=axis1    raxis=axis2
      width=6    patternid=midpoint
      type=FREQ
      ;
run;
quit;

%END;

%mend serve;

%serve
```

Considerations

The clicks.sas program does a pretty good job of collecting x,y clicks, but it's not perfect. First, if the applet sends clicks too fast, x,y values will be ignored.

More serious problems exist. If the program accessing clicks.sas sends unexpected data it may generate an error which stops the SAS server program. If the applet disconnects before the server sends the sum, an error may also terminate the server.

What is a tcp/ip socket?

A tcp/ip socket is a connection between two programs across a network using the tcp/ip protocol, which allows each program to treat the other as an input/output device.

One program operates as a *server* in that it waits for the other to initiate a connection. The server listens for a connection on a particular *port*. The machine on which the tcp/ip server program is running may keep track of certain port numbers so that other server programs may avoid them. These pre-assigned ports are known as "well known ports". On a UNIX system these ports are listed in the file /etc/services. Other port numbers are up for grabs.

Once the server accepts the connection from the client it may either read or write to the socket. As currently implemented, however, a SAS DATA step in server mode must read from the socket before writing to it. If the DATA step contains a "put" statement immediately following a "filename... server", it will generate an error stating that the connection was not present.

Conclusions

Each of the three methods discussed here has its strengths and limitations. Fortunately, if no one method suffices, it is possible to use more than one of them in combination.

A CGI script in Perl, for example, could generate and check a port number and pass it and the form data to a SAS program. It could then pass a reference to an applet back to the browser with the port number included as a parameter. This would allow several copies of a SAS socket server to run simultaneously, each using a different port.

A Java applet could pass SQL requests to a SAS/SHARE*NET server and also connect to a SAS program serving a socket.

SAS Institute has provided a useful compliment of interface methods.

Trademarks

Resources and References

The best source for information on using SAS with Java applets and with the Internet in general are the SAS Institute World Wide Web pages. These include:

SAS Institute Inc., *SAS Institute Web Tools*. http://www.sas.com/rnd/web/intro.html.

SAS Institute Inc., *Accessing SAS Data with Java(tm) and JDBC Technologies* http://www.sas.com/rnd/web/jdbc.html

SAS Institute Inc., *SAS Institute Web Tools -- SAS CGI* http://www.sas.com/rnd/web/sascgi.html

SAS Institute Inc., *SAS Institute Web Tools -- FTP and SOCKET access Methods* http://www.sas.com/rnd/web/ftpaccess.html

Other resources include:

Friendly, Michael, *Online Statistics* http://www.math.yorku.ca/SCS/Online/

Hoyle, Larry, *Examples of Connecting SAS to WWW* http://www.ukans.edu/cwis/units/IPPBR/ksdata/ksdata.htm#ecsw

Hoyle, Larry, *SAS Software and the WWW - What Next?, SAS User Group International Conference (SUGI 21),* Chicago, March 1996.

Hoyle, Larry, *More on using SAS with WWW* MidWest SAS Users Group Conference, Cleveland, October 1995.

Hoyle Larry, *Connecting SAS to the World Wide Web - Forms Across the Internet* MidWest SAS Users Group Conference (MWSUG94) Omaha, September 1994

Sun Microsystems Inc*.,* JavaSoft Home Page http://java.sun.com/

Sun Microsystems Inc*., The JDBC(tm) Database Access API* http://splash.javasoft.com/jdbc

Larry Hoyle
IPPBR, University of Kansas
607 Blake Hall
Lawrence, KS, 66045-2960
l-hoyle@ukans.edu
http://www.ukans.edu/cwis/units/IPPBR