Greetings from the Edge: Using javaobj in DATA Step

Richard A. DeVenezia, Remsen, NY

ABSTRACT

Both SAS and Java are systems with wide and deep foundations that can be used to perform sophisticated analysis and implement complex designs. Version 9 of SAS brings to us the DATA Step Component Interface and Object Dot Syntax[1], "providing a mechanism for instantiating and accessing predefined component objects from within a DATA Step program." One of the predefined components is the **Java Object**. Javaobj is a gateway to hybrid solutions where Java does the things that SAS can't. In this paper you will be guided through several Java Object examples. The final example is "Accessing a Postgresql database from DATA Step."

INTRODUCTION

The reader should be have some knowledge of these topics:

- Configuring a SAS session
- DATA Step
- Writing Java source
- Java RMI
- Compiling Java source into class files
- Database management systems
- JDBC drivers

JAVA DEVELOPMENT

A free software development kit for Java is available at `java.sun.com/j2se`. The kit includes a compiler, javac , a file collector, jar and a program loader, java . The Sun site has excellent tutorials and forums. The are also web sites beyond counting related to Java.

JAVA SOURCE CODE

Use your favorite text editor or development tool to write your Java sources.

COMPILING JAVA

```
$ javac class.java
```

Depending on the source, one or more class files will be created.

JAVA JARS

```
$ jar cf jar-file class-file-1 … class-file-N
```

A jar file is a single compressed file containing one or more other files. The concept is the same as a zip file. If you have developed several classes for use, you should collect them into a jar file.

CONFIGURING SAS SESSION

The javaobj component interface requires that your Java classes reside in one of the paths listed in environment variable CLASSPATH. This variable can be set on the SAS session command line:

$ SAS … -set CLASSPATH %CLASSPATH%;*myPath*

myPath should be either the folder or jar-file where your classes reside.

GOTCHAS

The SAS implementation of the *Component Interface* caches Java classes. If you recompile a Java class after it has been used in a SAS session, you must restart SAS to use the new version of the class.

EXAMPLE1: HELLO SAS

Example1.java

```
public class Example1 {
  public Example1 () {}
  public String getMessage () {return "Hello SAS";}
}
```

The SAS programs in this paper will use a convention of prefixing all javaobj variables with j.

Example1.sas

```
data _null_;
  length message $200;
  declare javaobj j ('Example1');
  j.callStringMethod ('getMessage', message);
  put message=;
run;
--- log --
message=Hello SAS
```

JAVA CODING PATTERNS

Programming in a consistent manner or pattern makes your code and thinking accessible to other programmers (and yourself at a later time). Here is the pattern that will be used in this paper, *italic* parts will be replaced with appropriate identifiers.

***Class*.java**

```
public class Class
{
  private double variable;

  public double getVariable()
  {
    return this.variable;
  }

  public void setVariable(double variable)
  {
    this.variable = variable;
  }
}
```

Class variables and methods, have descriptive names such as word-1Word-2…Word-N. The initial word is lowercase, and subsequent words are uppercase.

Class variables are never directly retrieved or updated. Their values are instead accessed through two methods known as getters and setters. Business or niche logic (context specific rules and value transformations) is done in the getters and setters.

METHOD SIGNATURES

The functionality of a Java class is realized in the design and programming of its methods. A method is defined to take zero or more arguments and to return either nothing (void), a Java primitive or a Java class. A method specifies its required argum-ents as a comma-separated list of in the form *arg-type arg-name.* The **pattern** of the argument types is known as the method's **signature**. An 'overloaded' method is a method that is imple-mented more than once. Each implementation must have a unique signature.

A constructor is a special method of a class. A class's constructor has the same name as the class and is only invoked when a class is instantiated by the Java new operator. A constructor having no arguments is called a null constructor. A class can have any number of constructors, provided their signatures are unique.

DATA Step has two base value types, *NUMERIC* and *CHARACTER,* which correspond to Java base types *Double* and *String.* DATA Step also has the javaobj type. Every SAS javaobj knows what Java class it is, but does not know what it is a subclass of. A javaobj can not be assigned as the result of a Java method; it must be created with the DATA Step _NEW_ operator.

The type restrictions of DATA Step mean that javaobj can only invoke Java methods that 1) have no arguments; or 2) have a signature containing only Strings, Doubles and Java classes (where the class of the SAS javaobj must match **exactly**); and 3) do not return a Java class.

SAS JAVAOBJ STRUCTURAL FORM

A javaobj is declared and instantiated as follows:

```
declare javaobj var ('Class' [,arg-1[,…[,arg-N]]);
```

The declaration and instantiation can be separated as follows:

```
declare javaobj var;
var = _NEW_ javaobj('Class' [,arg-1[,…[,arg-N]]);
```

Class must exist in the locations listed in the environment variable CLASSPATH. All the classes of the Java SDK are implicitly available. Class X in a package named A.B.C is referenced as 'A/B/C/X'.

The public fields of the java object are accessed as follow:

```
var.[get|set][Static][type]Field('field', value);

type is any of String, Double, Int, Short, Byte,
Long, Float
```

Any ***value*** conversions necessary are performed by the javaobj interface. This implicit conversion might be used as a justification for public class fields. However, it contradicts the coding pattern of using only getter and setter methods to access the fields of a class.

The public methods of the java object are accessed as follow:

```
var.call[Static][type]Method
('method' [,arg-1[,…[,arg-N]][,return-arg]);

type is any of Void, Double, String, Boolean, Short,
Byte, Long, Float, Int
```

No type conversions are performed for arguments, and their types must match the Java method signature exactly.

The last argument of a Javaobj method invocation is a SAS variable that will receive the value returned by the Java method. Javaobj can only interface with methods that return numbers and strings. A Java boolean is returned as 0=false, 1=true.

A javaobj is typically instantiated only once in a DATA Step. If the DATA Step loops, be sure to instantiate inside an `if _n_ = 1 then do` block.

EXAMPLE 2

A Java class with method signatures accessible to SAS javaobj.

Example2.java

```
public class Example2 {
  private String message;
  public Example2(String newMessage) {
    this.setMessage(newMessage);
  }
  public String getMessage() {
    return this.message;
  }
  public void setMessage(String newMessage) {
    this.message = newMessage;
  }
}
```

Example2.sas

```
data _null_;
  length m $20;
  declare javaobj j ('Example2', 'Hello Java');
  j.callStringMethod ('getMessage', m);
  put m=;
run;
--- log ---
m=Hello Java
```

SAS ERRORS

There may be the unusual situation of seeing an error message in the SAS log.

```
dcl javaobj var('Class' [,arg-1[,…[,arg-N]]);
```

```
ERROR: An error has occurred during class method
OM_NEW(3) of "DATASTEP.JAVAOBJ".
```

Any of these may have happened

- *Class* was not found in the
- *Class* was found, but had no constructor with signature matching *arg-1*[,…[,*arg-N*]
  - The class may need to be wrapped to be useable from SAS
  - You accidentally specified too few or too many arguments
  - Reread the class documentation
- *Class* was found, but constructor threw an Exception
  - The class needs to be wrapped and the exception caught

EXAMPLE 3

A class not found in CLASSPATH.

Example3.sas

```
data _null_;
  declare javaobj j ('ClassFooBarDoesNotExist');
run;
--- log ---
ERROR: An error has occurred during class method
OM_NEW(3) of "DATASTEP.JAVAOBJ".
```

EXAMPLE 4

The java.net.URI object does not have a constructor with signature String-String. An error occurs because javaobj is passed two strings, and the javaobj interface can't find a constructor with that signature. The Uninitialized object error is displayed when a method of an uninstantiated javaobj variable is invoked.

Example4.sas

```
data _null_;
  length s $200;
  declare javaobj juri ('java/net/URI', 'http',
                                  'www.sas.com');
  juri.callStringMethod ('toString', s);
  put s=;
run;
--- log ---
ERROR: An error has occurred during class method
OM_NEW(3) of "DATASTEP.JAVAOBJ".
ERROR: Uninitialized object at line 5 column 3.
ERROR: XOB failure detected.  Aborted during the
EXECUTION phase.
```

EXAMPLE 5

Invoking a nonexistent method.

Example5.sas

```
data _null_;
  length s $200;
  dcl javaobj juri ('java/net/URI', 'www.sas.com');
  juri.callStringMethod ('getLastName', s);
  put s=;
run;
--- log ---
ERROR: Could not find method getLastName at line 4
column 3.
ERROR: An error has occurred during instance method
OM_CALLSTRINGMETHOD(3278) of "DATASTEP.JAVAOBJ".
```

EXAMPLE 6

Invoking an existent method, with wrong signature. The URI object does not have a toString() method that requires a double argument.

Example6.sas

```
data _null_;
  length s $200;
  dcl javaobj juri ('java/net/URI', 'www.sas.com');
  juri.callStringMethod ('toString', s, 100);
  put s=;
run;
--- log ---
ERROR: Could not find method toString at line 40
column 3.
ERROR: An error has occurred during instance method
OM_CALLSTRINGMETHOD(3278) of "DATASTEP.JAVAOBJ".
```

WRAPPERS

If a class constructor or method has a signature that is incompatible with SAS, then a wrapper class must be created. A wrapper is most often a subclass or an adapter.

This example shows how a message dialog can be displayed from Java. This is a useful concept for situations where the course of action in a Java class is dependent on user input. Obtaining and handling the user input directly in Java is more convenient than dropping back to SAS DATA Step to obtain and respond to the user input.

Example 7 is coded in an adapter pattern that lets SAS utilize the showOptionDialog() and showMessageDialog() methods of the javax.swing.JoptionPane class. The methods have *int* arguments in their signature and would normally be inaccessible to javaobj.

Example7.java

```
import javax.swing.JOptionPane;

public class Example7
{
  private String optionSelected;

  public Example7() {}

  public void showMessageDialog
                   (String message, String title)
  {
   JOptionPane.showMessageDialog(
     null, message, title,
     JOptionPane.INFORMATION_MESSAGE);
  }

  public int showOptionDialog
                   (String message, String title)
  {
    return showOptionDialog(message,title,null);
  }

  public int showOptionDialog
    (String message, String title, String _options)
  {
    String[] options;

    if (_options == null)
      options = "Yes,No".split(",");
    else
      options = _options.split(",");

    int choice = -1;
    optionSelected = null;
    try {
      choice = JOptionPane.showOptionDialog(
        null, message, title,
        JOptionPane.DEFAULT_OPTION,
        JOptionPane.QUESTION_MESSAGE, null,
        options, options[0]);
      if (choice >= 0)
        this.optionSelected = options[choice];
    }
    catch (Exception e) {
      JOptionPane.showMessageDialog(null,e,
        "Exception in showOptionDialog",
        JOptionPane.WARNING_MESSAGE);
    }
    return choice;
  }

  public String getOptionSelected () {
    return this.optionSelected;
  }
}
```

Example7.sas

```
data _null_;
  dcl javaobj jd ('Example7');
  jd.callIntMethod ('showOptionDialog',
    'Two plus Two equals','Math question',
    'Zero,One,Two,Three,Four,Five', x);
  put x=;
  length s $100;
  jd.callStringMethod ('getOptionSelected', s);
  put s=;
  if x = 4
    then answer = trim(s)||' is correct'||'0a'x;
    else answer = trim(s)||' is incorrect'||'0a'x;

  jd.callVoidMethod ('showMessageDialog',
   trim(answer)||'Thank you for playing', 'Aloha');
run;
--- log ---
x=4
s=Four
```

JAVA MAIN

The development of a Java class should be complete as possible before attempting to use it as a SAS javaobj. Part of the reason for this is the *gotcha* that a SAS session caches javaobjs. If a Java class is recompiled after being used in a SAS session, then SAS needs to be restarted to use the recompiled class.

A Java class that has a method `public static void main (String arg[])` can test method functionality outside of SAS.

Example 7.java with main()

Suppose this method were added to Example7.java

```
public static void main (String arg[]) {
    Example7 me = new Example7 ();
    int answer = me.showOptionDialog (
      "Two plus Two equals", "Math question",
      "Zero,One,Two,Three,Four,Five");
    System.out.println ("You selected option " +
      answer + " " + me.getOptionSelected());
    me.showMessageDialog ("Thanks for playing","");
    System.exit(0);
  }
}
```

The main() can be run from a terminal or command prompt. The following command assumes the java class is located in the current directory.

```
$ java -cp . Example7
```

JAVA ENVIRONMENT

This next example demonstrates how non-trivial tasks require more Java programming than SAS programming. It also shows how looping can be controlled by an external agent (SAS in this case).

In programming Java applications, it is often important to know what version of the VM is running. The javaobj interface is still experimental and there is no documentation on how it selects the VM it will use. This example will reveal the properties of the VM that is hosting the SAS javaobj.

Example8 is a wrapper class that surfaces the Enumeration returned by the static method System.getProperties().

Example8.java

```
import java.util.Enumeration;

public class Example8 {
  private Enumeration e;
  public Example8 () {
    e = System.getProperties().propertyNames();
  }
  public String getNextProperty () {
    if (e.hasMoreElements()) {
      String p = (String) e.nextElement();
      return p + "=" + System.getProperty(p);
    }
    else {
      return null;
    }
  }
}
```

By assuming all properties are not blank, the class can return a null to indicate the last property has been delivered. SAS uses the blank string (null) as the condition for not requesting more properties.

Example8.sas

```
data _null_;
  dcl javaobj j ('Example8');
  length s $200;
  j.callStringMethod ('getNextProperty', s);
  do while (s ne '');
    put s;
    j.callStringMethod ('getNextProperty', s);
  end;
run;
--- log ---
…
java.vm.version=1.4.1_01-b01
java.vm.vendor=Sun Microsystems Inc.
…
```

At least 28 system properties will be listed. The standard set of property keys of a JVM is documented at java.lang.System.getProperties()

JAVA EXCEPTIONS

Often a SAS program will need to respond to an exception thrown by a Java method. The default behavior is to generate a SAS error. Catching exceptions and making information about the exception available should be part of the design of a wrapper class. According to a little birdie:

SAS version 9.1 will have new Javaobj methods for dealing with exceptions:

- exceptionCheck ()
- exceptionClear ()
- exceptionDescribe ()

This example is a wrapper for FileInputStream, when the constructor throws an exception it generates a SAS error.

Example9.java

```
import java.io.FileInputStream;

public class Example9 {
  private FileInputStream fis;
  public Example9 (String path) throws Exception
  {
    this.fis = new FileInputStream (path);
  }
}
```

Example9.sas

```
data _null_;
  dcl javaobj j ('Example9', 'foobar.file');
run;
--- log ---
ERROR: An error has occurred during class method
OM_NEW(3) of "DATASTEP.JAVAOBJ".
```

A slight modification catches the Exception and records its message for review.

Example10.java

```
import java.io.FileInputStream;

public class Example10 {
  private FileInputStream fis;
  private String exceptionMessage;
  public Example10(String path) {
    try {
      this.fis = new FileInputStream (path);
    }
    catch (Exception e) {
      this.exceptionMessage = e.toString();
      for (int i=0;i<e.getStackTrace().length; i++)
      {
        this.exceptionMessage
        += "\n" + e.getStackTrace()[i];
      }
    }
  }
  public String getExceptionMessage () {
    return this.exceptionMessage;
  }
}
```

Example10.sas

```
data _null_;
  dcl javaobj j ('Example10', 'foobar.file');
  length s $1000;
  j.callStringMethod ('getExceptionMessage', s);
  if s ne '' then do;
    i = 1;
    do while (scan (s,i,'0a'x) ne '');
      l = scan (s,i,'0a'x);
      i+1;
      put l;
    end;
  end;
run;
--- log ---
java.io.FileNotFoundException: foobar.file (The
system cannot find the file specified)
java.io.FileInputStream.open(Native Method)
java.io.FileInputStream.<init>(FileInputStream.java:
103)
java.io.FileInputStream.<init>(FileInputStream.java:
66)
Example10.<init>(Example10.java:8)
```

REMOTE DATABASES

There are both commercial and open source database providers. A commercial provider is a company such as SAS or Oracle, and most other systems supported by SAS/Access. In the open source arena, two favorites are MySQL and Postgresql.

JDBC DRIVER

Accessing the remote system from Java requires a JDBC driver. The concept is the same as ODBC drivers found in Windows.

	JDBC Driver	
Database	Commercial	Open Source
Commercial	Typical	Rarely ?
Open Source	Sometimes	Typical

Sun lists 177 DBMS vendors with JDBC drivers[2]. Depending on the database system, you may need to purchase additional system components and drivers.

CONNECTING FROM JAVA

Once you obtain the required JDBC driver (typically a jar file), you will need database connecting parameters:

- URL
- Password
- Username

The URL is a specially constructed value particular to each individual JDBC driver. The construct for Postgresql is

```
jdbc:postgresql://{host}[:{port}]/{database}
```

where {host} is either an IP-address or domain name.

USING A JDBC DRIVER

The typical pattern is:

```
Class.forName(jdbcDriver);
Connection con = DriverManager.getConnection
                   ( URL, username, password );
```

Class.forName explicitly loads the JDBC driver class. The class named in *jdbcDriver* must exist somewhere on your CLASSPATH. This typically means adding a jar to the CLASSPATH of the SAS session.

> "When the method getConnection is called, the Driver-Manager will attempt to locate a suitable driver from amongst those loaded at initialization and those loaded explicitly using the same classloader as the current applet or application."[3]

Sun also has excellent tutorials for learning more about Java and JDBC (see Links at end).

JAVA REMOTE METHOD INVOCATION (RMI)

Making connections is an important part of an application using JDBC, and one of the most time-consuming. When a Connection object is deleted() or otherwise nulled, the connection will be lost. Repeatedly making and breaking connections is undesirable.

Javaobj variables are not persistent. When a DATA Step is finished running, the java objects are gone. Some SAS programming tasks require dealing with JDBC in multiple DATA Steps, implying undesired multiple reconnects. However, there is a way to access a java object that persists outside a SAS session.

Java RMI was designed for this kind of scenario. RMI allows objects in one Java Virtual Machine (JVM) to run methods of an object in another JVM. In the final example, one VM is an independent Java process that acts as a server and the DATA Step that acts as a 'client VM'.

Typical RMI schemes have these components:

- an interface (a set of methods that can be remotely invoked)
- an interface implementation
- implementation _stub and _skel classes created by the *rmic* tool
- a server that hosts a registered instance of the implementation
- a client that retrieves a reference to the registered instance so that it may operate upon it.

Detailed discussion of RMI systems is beyond the scope of this paper (see Links at end).

SAMPLE SCENARIO

A SAS analyst needs to retrieve data from and store results in a client's DBMS. The client provides this information.

Database version	Postgresql 7.3
Hostname	www.devenezia.com
Port	5434
Database	Sesug03demo
User	Sesug03demo
Pass	D3m0oeoe

The analyst does not have SAS/Access licensed, and even if she did, there is no SAS/Access component to connect to Postgresql or JDBC drivers.

Note: The complete java and sas source of this sample is available at the author's website.

GETTING A DRIVER

Postgresql JDBC drivers can be downloaded from
http://jdbc.postgresql.org/download.html

For readers connecting to other database systems; this example requires a JDBC 2.1 API compliant driver which implements the ResultSet moveToInsertRow() method.

JAVA CLASSES

The interface between DATA Step and JDBC is an implemention of an RMI scheme and is aptly named DataStepJdbcInterface. What are the classes and some of their important features?

- DataStepJdbcInterface
  - Declares the methods that may be remotely invoked
  - Fine level access to Connections, Statements and ResultSets
- DataStepJdbcManager (implements the interface)
  - Manages multiple Connections
  - Allows multiple Statements per Connection
  - Allows one ResultSet per Statement
  - Connections, Statements and ResultSets maintained in Hashtables keyed by handle
  - Invalid handles incur ½ second delay as a security feature to slow down 'scanners'
- DataStepJdbcServer (hosts an instance of the manager)
  - Controls registry port, lookup name and how many connections manager will allow
  - getReferenceToPersistentManager() returns hosted manager, allowing for server process startup delay
  - Manager messages are written to the server stdout. Typically the terminal window that served was started from.
- DataStepJdbcWrapper
  - Reimplements the interface for SAS (remember javaobj can never be returned to SAS)
  - Each interface method is delegated to the remote manager.

DataStepJdbcInterface.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface DataStepJdbcInterface extends Remote
{
  public double getConnectionHandle (String driverClass,
       String databaseURL, String username, String password)
                                    throws RemoteException;
  public double getStatementHandle (double cHandle)
                                    throws RemoteException;
  public void closeConnectionHandle (double handle)
                                    throws RemoteException;
  public void closeStatementHandle (double handle)
                                    throws RemoteException;
  public void closeResultSetHandle (double handle)
                                    throws RemoteException;
  public String getExceptionMessage (double handle)
                                    throws RemoteException;
  public void executeUpdate (double handle, String sql)
                                    throws RemoteException;
  public double executeQuery (double handle, String sql)
                                    throws RemoteException;
  public boolean nextRow (double handle)
                                    throws RemoteException;
  public double getValue (double handle, double colNum)
                                    throws RemoteException;
  public String getText (double handle, double colNum)
                                    throws RemoteException;
  public void setValue (double handle,
    double colNum, double value)      throws RemoteException;
  public void setText (double handle,
    double colNum, String text)       throws RemoteException;
  public void insertRow (double handle)
                                    throws RemoteException;
  public void moveToInsertRow (double handle)
                                    throws RemoteException;
  public String getColumnName (double handle, double colNum)
                                    throws RemoteException;
  public String getSasColumnType (double handle,
    double colNum)                  throws RemoteException;
  public int getColumnCount(double handle)
                                    throws RemoteException;
  public int getColumnDisplaySize(double handle,
    double colNum)                  throws RemoteException;
}
```

DataStepJdbcManager.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.sql.*;
import javax.swing.JOptionPane;

public class DataStepJdbcManager
  extends UnicastRemoteObject
  implements DataStepJdbcInterface
{
  private Hashtable connections;
  private Hashtable statements;
  private Hashtable resultSets;
  private Hashtable resultSetMetaDatas;
  private Hashtable exceptionMessages ;
  private int numberOfHandles;

  java.text.DecimalFormat nf = new
  java.text.DecimalFormat ("########");

  //-------------------------------------------------------
  public DataStepJdbcManager (int numHandles)
                                     throws RemoteException
  {
    this.numberOfHandles    = numHandles;
    this.connections        = new Hashtable (numHandles);
    this.statements         = new Hashtable ();
    this.resultSets         = new Hashtable ();
    this.resultSetMetaDatas = new Hashtable ();
    this.exceptionMessages  = new Hashtable ();
  }
  //-------------------------------------------------------
  private Double getHandle () {
    Double handle;
    do {
      handle = new Double (Math.floor (Math.random()*1e8));
    }
    while (
        connections.containsKey(handle)
     || statements.containsKey(handle)
     || resultSets.containsKey(handle)
     || exceptionMessages.containsKey(handle)
    );
    return handle;
  }
    //-----------------------------------------------------
  private Object getObjectOfKey(Hashtable H, Object key,
                     boolean clearMessage) throws Exception
  {
    Object o = H.get (key);

    if (o == null) {
      Thread.currentThread().sleep(500);
      throw new Exception ("Invalid handle.");
    }

    if (clearMessage)
      exceptionMessages.put (key, new String());

    return o;
  }

  //-------------------------------------------------------
  private Connection getConnectionOfHandle(double handle,
                     boolean clearMessage) throws Exception
  {
    Double key = new Double(handle);
    return (Connection)
             getObjectOfKey (connections, key, clearMessage);
  }

  //-------------------------------------------------------
  private Statement getStatementOfHandle(double handle,
                     boolean clearMessage) throws Exception
  {
    Double key = new Double(handle);
    return (Statement)
              getObjectOfKey (statements, key, clearMessage);
  }

  //-------------------------------------------------------
  private ResultSet getResultSetOfHandle(double handle,
                     boolean clearMessage) throws Exception
  {
    Double key = new Double(handle);
    return (ResultSet)
              getObjectOfKey (resultSets, key, clearMessage);
  }

  //-------------------------------------------------------
  private ResultSetMetaData getResultSetMetaDataOfHandle
       (double handle, boolean clearMessage) throws Exception
  {
    Double key = new Double(handle);
    return (ResultSetMetaData)
      getObjectOfKey (resultSetMetaDatas, key, clearMessage);
  }

  //-------------------------------------------------------
  public double getConnectionHandle (String driverClass,
        String databaseURL, String username, String password)
                                      throws RemoteException
  {
    Double cHandle = getHandle();
    Connection con ;

    if (connections.size() == numberOfHandles) {
      System.out.println ("No more handles.");
      throw new RemoteException ("No more handles.");
    }

    try {
      System.out.println ("loading "+driverClass);
      Class.forName(driverClass);
      System.out.println ("connecting to "+databaseURL);
      con = DriverManager.getConnection
                       (databaseURL, username, password);
      System.out.println ("connected");
    }
    catch (Exception e) {
      System.out.println (e);
      throw new RemoteException
                   ("Problem connecting to database", e);
    }

    connections.put (cHandle, con);

    System.out.println ("Connection handle " +
                        nf.format(cHandle) + " is for " +
                   username + " connected to " + databaseURL);

    return cHandle.doubleValue();
  }
  //-------------------------------------------------------
  public double getStatementHandle (double cHandle)
                                      throws RemoteException
  {
    Double sHandle = getHandle ();
    Connection con ;
    Statement st;

    try {
      con = getConnectionOfHandle (cHandle, true);
      st = con.createStatement ();
    }
    catch (Exception e) {
      throw new RemoteException
                      ("Problem getting a statement", e);
    }

    statements.put (sHandle, st);
    System.out.println ("Statement handle " +
                              nf.format(sHandle) + " ready");

    return sHandle.doubleValue();
  }
  //--------------------------------------------------------
  public void closeConnectionHandle (double handle)
                                       throws RemoteException
  {
    Double key = new Double(handle);
    try {
      Connection con = getConnectionOfHandle (handle,false);
      for (Enumeration e = statements.keys() ;
           e.hasMoreElements() ;) {
        Double skey = (Double) e.nextElement();
        Statement st = (Statement) statements.get(skey);
        if (st.getConnection() == con) {
          closeStatementHandle (skey.doubleValue());
        }
      }
      con.close();
      connections.remove (key);
      exceptionMessages.remove (key);
    }
    catch (Exception e){throw new RemoteException("", e);}

    System.out.println ("Connection handle " +
                          nf.format(handle) + " was closed");
  }
  //--------------------------------------------------------
  public void closeStatementHandle (double handle)
                                       throws RemoteException
  {
    Double key = new Double (handle);
    try {
      Statement st = getStatementOfHandle (handle, false);

      for (Enumeration e = resultSets.keys() ;
           e.hasMoreElements() ;) {
        Double rkey = (Double) e.nextElement();
        ResultSet rs = (ResultSet) resultSets.get(rkey);
        if (rs.getStatement() == st) {
          closeResultSetHandle (rkey.doubleValue());
        }
      }
      st.close();
      statements.remove (key);
      exceptionMessages.remove (key);
    }
    catch (Exception e) {throw new RemoteException("", e);}
    System.out.println ("Statement handle " +
                          nf.format(handle) + " was closed");
  }
  //--------------------------------------------------------
  public void closeResultSetHandle (double handle)
                                       throws RemoteException
  {
    Double key = new Double (handle);
    try {
      ResultSet rs = getResultSetOfHandle (handle, false);
      rs.close();
      resultSets.remove (key);
      resultSetMetaDatas.remove (key);
      exceptionMessages.remove (key);
    }
    catch (Exception e){throw new RemoteException("", e);}

    System.out.println ("ResultSet handle " +
                          nf.format(handle) + " was closed");
  }
  //--------------------------------------------------------
  private void createExceptionMessage
                                 (double handle, Exception e)
  { createExceptionMessage (handle, e, null); }
  //--------------------------------------------------------
  private void createExceptionMessage
                  (double handle, Exception e, String text)
  {
    String msg;
    Double key = new Double (handle);

    if (text != null) msg = text + "\n";
              else msg = "";

    msg += e.toString();
    exceptionMessages.put(key,msg);
  }
  //--------------------------------------------------------
  public String getExceptionMessage (double handle)
                                       throws RemoteException
  {
    Double key = new Double (handle);
    try {
      return (String)
             getObjectOfKey (exceptionMessages, key, false);
    }
    catch (Exception e){ throw new RemoteException("", e); }
  }
  //--------------------------------------------------------
  public void executeUpdate (double sHandle, String sql)
                                       throws RemoteException
  {
    try {
      Statement st = getStatementOfHandle (sHandle, true);
      System.out.println ("handle " + nf.format(sHandle) +
                             " is executing update: " + sql);
      try {
        int rowCount = st.executeUpdate (sql);
        System.out.println ("handle " + nf.format(sHandle) +
                           " updated " + rowCount + " rows");
      }
      catch (Exception e) {
        createExceptionMessage(sHandle, e, sql);
      }
    }
    catch (Exception e){throw new RemoteException("", e);}
  }
  //--------------------------------------------------------
  // returns handle to a ResultSet
  public double executeQuery (double sHandle, String sql)
                                       throws RemoteException
  {
    Double rHandle = getHandle();
    try {
      Statement st = getStatementOfHandle (sHandle, true);
      System.out.println ("handle " + nf.format(sHandle) +
                              " is executing query: " + sql);
      try {
        ResultSet rs = st.executeQuery (sql);
        ResultSetMetaData rsmd = rs.getMetaData();
        resultSets.put (rHandle, rs);
        resultSetMetaDatas.put (rHandle, rsmd);
      }
      catch (Exception e) {
        createExceptionMessage(sHandle, e, sql);
      }
    }
    catch (Exception e) {throw new RemoteException("", e);}

    return rHandle.doubleValue();
  }

  //--------------------------------------------------------
  public boolean nextRow (double rHandle)
                                       throws RemoteException
  {
    try {
      ResultSet rs = getResultSetOfHandle (rHandle, true);
      try
      { return rs.next(); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); return false; }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }

  //--------------------------------------------------------
  public double getValue (double rHandle, double colNum)
                                       throws RemoteException
  {
    try {
      ResultSet rs = getResultSetOfHandle (rHandle, true);
      try
      { return rs.getDouble((int)colNum); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); return
Double.NaN;}
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }
  //-------------------------------------------------------
  public String getText (double rHandle, double colNum)
                                      throws RemoteException
  {
    try {
      ResultSet rs = getResultSetOfHandle (rHandle, true);
      try
      { return rs.getString((int)colNum); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); return ""; }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }

  //-------------------------------------------------------
  public void setValue (double rHandle,
          double colNum, double value) throws RemoteException
  {
    try {
      ResultSet rs = getResultSetOfHandle (rHandle, true);
      try
      { rs.updateDouble((int)colNum, value); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }

  //-------------------------------------------------------
  public void setText (double rHandle,
           double colNum, String text) throws RemoteException
  {
    try {
      ResultSet rs = getResultSetOfHandle (rHandle, true);
      try
      { rs.updateString((int)colNum,text); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }

  //-------------------------------------------------------
  public void insertRow (double rHandle)
                                      throws RemoteException
  {
    try {
      ResultSet rs = getResultSetOfHandle (rHandle, true);
      try
      { rs.insertRow(); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }

  //-------------------------------------------------------
  public void moveToInsertRow (double rHandle)
                                      throws RemoteException
  {
    try {
      ResultSet rs = getResultSetOfHandle (rHandle, true);
      try
      { rs.moveToInsertRow(); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }

  //-------------------------------------------------------
  public String getColumnName (double rHandle,
                        double colNum) throws RemoteException
  {
    try {
      ResultSetMetaData rsmd =
               getResultSetMetaDataOfHandle (rHandle, true);
      try
      { return rsmd.getColumnName((int)colNum); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); return ""; }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }

  //-------------------------------------------------------
```

```
  public String getSasColumnType (double rHandle,
                        double colNum) throws RemoteException
  {
    try {
      ResultSetMetaData rsmd =
               getResultSetMetaDataOfHandle (rHandle, true);
      try {
        int type = rsmd.getColumnType((int)colNum);
        switch (type) {
          case java.sql.Types.CHAR:    return "C";
          case java.sql.Types.VARCHAR: return "C";
          case java.sql.Types.BIGINT:  return "N";
          case java.sql.Types.BINARY:  return "N";
          case java.sql.Types.BOOLEAN: return "N";
          case java.sql.Types.DECIMAL: return "N";
          case java.sql.Types.DOUBLE:  return "N";
          case java.sql.Types.FLOAT:   return "N";
          case java.sql.Types.INTEGER: return "N";
          case java.sql.Types.NUMERIC: return "N";
          case java.sql.Types.REAL:    return "N";
          case java.sql.Types.SMALLINT:return "N";
          default: return "?";
        }
      }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); return "?"; }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }

  //-------------------------------------------------------
  public int getColumnCount(double rHandle)
                                      throws RemoteException
  {
    try {
      ResultSetMetaData rsmd =
               getResultSetMetaDataOfHandle (rHandle, true);
      try
      { return rsmd.getColumnCount(); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); return -1; }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }

  //-------------------------------------------------------
  public int getColumnDisplaySize(double rHandle,
                        double colNum) throws RemoteException
  {
    try {
      ResultSetMetaData rsmd =
               getResultSetMetaDataOfHandle (rHandle, true);
      try
      { return rsmd.getColumnDisplaySize((int)colNum); }
      catch (Exception e)
      { createExceptionMessage(rHandle, e); return -1; }
    }
    catch (Exception e) {throw new RemoteException("", e);}
  }
}
```

DataStepJdbcManager.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.*;

public class DataStepJdbcServer
{
  protected static final String
                        RMI_NAME = "DATASTEP-JDBC-MANAGER";

  private String exceptionMessage;

  // Main is run when class is run as stand-alone java
  // application and provides the run-time persistence

  public static void main(String args[]){

    // set security according to file named in environment
    // variable java.security.policy
    System.setSecurityManager(new RMISecurityManager());

    try{
      // 1099 is the default port for RMI
      // This method removes the need to also run
      // "rmiregistry" as stand-alone process prior
      // to starting this class as a stand-alone process
      LocateRegistry.createRegistry(1099);

      // Instantiate the implementation of the interface
      // that will persist as long as this server runs
      int nCon = 5;
      DataStepJdbcManager manager = new
      DataStepJdbcManager (nCon);

      // Give this process the name that localhost clients
      // will locate with Naming.lookup()
      Naming.rebind (RMI_NAME, manager);

      System.out.println(manager.getClass().getName()
             + " ready to manage " + nCon + " connections.");
    }
    catch(Exception e)
    {
      System.out.println("Error: " + e);
    }
  }

  //----------------------------------------------------
  public static DataStepJdbcInterface
                           getReferenceToPersistentManager ()
  {
    int numberOfTimesToWait = 4;
    long durationToWaitInMillis = 250;

    Remote remote = null;

    try {
      for (int i=0;
           (i<numberOfTimesToWait) && (remote==null); i++)
      {
        try {
          remote = Naming.lookup(RMI_NAME);
        }
        catch (java.rmi.NotBoundException e)
        {
          // if not bound, wait a little, in case server was
          // just started and things haven't settled in yet
          System.out.println ("waiting ");
          Thread.currentThread().sleep
                                     (durationToWaitInMillis);
        }
      }
    }
    catch (Exception e) {System.out.println("Error " + e);}

    return (DataStepJdbcInterface) remote;
  }
}
```

Source of DataStepJdbcWrapper can be found at the author's website.

SAS MACROS

Several SAS macros were written to facilitate use of the interface for typical tasks. What are the macros and some of their important features?

- %StartServer
  - Start the java process that hosts the persistent manager
- %GetConnectionHandle, %GetStatementHandle
  - Get access to database
- %CloseConnection, %CloseStatement, %CloseResultSet
  - Clean up
- %CheckException
  - Show java exception message in SAS log
- %JdbcLoadTable
  - Copies SAS table to database
  - Runs one DATA Step
  - Executes DROP TABLE and CREATE TABLE statements in database
    - Database table column names will match SAS column names
    - Database table columns will be either type VARCHAR or DOUBLE
- %JdbcQuery
  - Database query result copied to a SAS table
  - Access database twice in two separate DATA Steps
    - first – obtains metadata needed to define SAS table
    - second – obtains data needed to populate SAS table

In each DATA Step a javaobj for DataStepJdbcWrapper is created to access the persistent Java SQL classes maintained in the server process. Macro variables are used to store handle values that are used in subsequent DATA Steps.

Note: The jdbcLoadTable macro and some other utility macros are not listed. JdbcQuery is listed to show a SAS programming task that benefits from a persistent connection.

jdbc_macros.sas

```
%*---------------------------------------------------;
%macro startServer (policy=);
  %local restore;
  %let restore
  = %sysfunc(getoption(xsync))
    %sysfunc(getoption(xwait))
    %sysfunc(getoption(xmin))
  ;
  options noxsync noxwait xmin;
  %sysexec start "Jdbc Server for DATA Step"
     java -Djava.security.policy=&policy DataStepJdbcServer ;
  options &restore;
%mend;
%*---------------------------------------------------;
%macro getConnectionHandle
(driver=,url=,user=,pass=,cHandle_mv=);
  data _null_;
    declare javaobj ji ("DataStepJdbcWrapper");
    ji.callDoubleMethod (
      "getConnectionHandle"
    , "&driver", "&url", "&user", "&pass", handle);
    put "Connection " handle "established.";
    call symput ("&cHandle_mv", put(handle,best12.));
  run;
%mend;
%*---------------------------------------------------;
%macro getStatementHandle (cHandle=, sHandle_mv=);
  data _null_;
    declare javaobj ji ("DataStepJdbcWrapper");
    ji.callDoubleMethod (
      "getStatementHandle", &cHandle, handle);
    put "Statement " handle "obtained.";
    call symput ("&sHandle_mv", put(handle,best12.));
  run;
%mend;
%*---------------------------------------------------;
%macro closeConnection (cHandle=);
  data _null_;
    declare javaobj ji ("DataStepJdbcWrapper");
    ji.callVoidMethod ("closeConnectionHandle", &cHandle);
  run;
%mend;
%*---------------------------------------------------;
%macro closeStatement (sHandle=);
  data _null_;
    declare javaobj ji ("DataStepJdbcWrapper");
    ji.callVoidMethod ("closeStatementHandle", &sHandle);
  run;
%mend;
%*---------------------------------------------------;
%macro tlp(varname,format);
  trim(left(put(&varname,&format)))
%mend;
%*---------------------------------------------------;
%macro checkException(handle, msgVar, action=);
 ji.callStringMethod("getExceptionMessage",&handle,&msgVar);
 if &msgVar ne "" then do;
    lfat = index (&msgVar, '0A'x);
    do while (lfat or &msgVar ne '');
           if lfat = 0 then _line = &msgVar;
      else if lfat = 1 then _line = '';
      else _line = substr (&msgVar,1,lfat-1);

      put _line;

      if lfat = 0 or lfat = length (&msgVar)
        then &msgVar = '';
        else &msgVar = substr (&msgVar, lfat+1);

      lfat = index (&msgVar, '0A'x);
    end;
    %if (%bquote(&action) ne ) %then %str(&action;);
  end;
%mend;
%*-----------------------------------------------;
%macro jdbcQuery (cHandle=NONE, sql=, obs=0, out=);
  %if &cHandle=NONE %then %do;
    %put ERROR: jdbcQuery: connection handle not specified.;
    %goto EndMacro;
  %end;

  %local sHandle rHandle;

  %getStatementHandle(cHandle=&cHandle, sHandle_mv=sHandle);

 * perform query, store ResultSetHandle in macro var;
 data _null_;
  length msg $1000;
  declare javaobj ji;
  ji = _new_ javaobj( "DataStepJdbcWrapper" );

  ji.callDoubleMethod("executeQuery",&sHandle,"&sql"
                                                  ,rHandle);
  %checkException (&sHandle, msg, action=STOP)
  ji.delete();
  call symput ("rHandle", put(rHandle,best12.));
  STOP;
 run;

 * process ResultSet metadata;
 %local nvars length get;

 data jdbc_columns (keep=varnum varname vartype varlen);
  declare javaobj ji;
  ji = _new_ javaobj( "DataStepJdbcWrapper" );

  length varname $64 vartype $1 varlen 4;

  ji.callIntMethod("getColumnCount",&rHandle,nvars);
  do varnum = 1 to nvars;
    ji.callStringMethod
    ("getColumnName",&rHandle,varnum,varname);
    ji.callStringMethod
    ("getSasColumnType",&rHandle,varnum,vartype);
    ji.callIntMethod
    ("getColumnDisplaySize",&rHandle,varnum,varlen);
    output;
  end;

  ji.delete();
  STOP;
 run;

 * generate SAS code to read each ResultSet row into SAS
 * observations, the code template is:
 * ji.callXMethod("getY",HANDLE,COLNUM,SASVAR);
 * X - String or Double
 * Y - Text or Value

 proc sql noprint;
   select
    trim(varname)
    || case vartype
        when 'C' then ' $' || %tlp(varlen,5.)
        else ' 8'
       end
   , case vartype
      when 'C' then
           'ji.callStringMethod("getText",'
           || "&rHandle.,"
           || %tlp(varnum,4.) || ','
           || trim(varname) || ')'
      else 'ji.callDoubleMethod("getValue",'
           || "&rHandle.,"
           || %tlp(varnum,4.) || ','
           || trim(varname) || ')'
     end
   , case varname
      when 'sas_rowid' then '' else varname
     end
   into
    :length separated by ' '
   ,:get separated by ';'
   ,:var separated by ' '
   from jdbc_columns
   ;
 quit;

 %let nvars = &sqlobs;

 data &out(keep=&var) ;
  length &length msg $1000;
  declare javaobj ji;
  ji = _new_ javaobj( "DataStepJdbcWrapper" );
  ji.callBooleanMethod("nextRow", &rHandle, _N_);
  do while (_N_);
   &get;
   OUTPUT;
   ji.callBooleanMethod("nextRow", &rHandle, _N_);
  end;
  ji.delete();
  STOP;
 run;

 %closeStatement (sHandle=&sHandle);
%EndMacro:
%mend jdbcQuery;
```

CONCLUSION

The Data Step Component Interface is a highly desirable addition to the SAS repertoire. A developer fluent in both sides of the interface can develop useful and creative tools for addressing previously unsolvable or expensively solved problems.

ACKNOWLEDGMENTS

Many thanks to David Staelens of Special Metals Corporation, New Hartford NY, for reviewing Java concepts and code.

CONTACT INFORMATION

Richard A. DeVenezia
9949 East Steuben Road
Remsen, NY 13438
richard@devenezia.com, http://www.devenezia.com

[1] The DATA Step Component Interface and Object Dot Syntax
http://support.sas.com/rnd/base/topics/datastep/dot/obj.html

[2] DBMS vendors with JDBC drivers
http://industry.java.sun.com/products/jdbc/drivers

[3] http://java.sun.com/j2se/1.4/docs/api/java/sql/DriverManager.html

Additional links
Sun tutorials - http://java.sun.com/docs/books/tutorial/
RMI - http://www.artima.com/javaseminars/modules/RMI/index.html