033-29

Greetings from the Edge: Using javaobj in DATA Step

Richard A. DeVenezia, Remsen, NY

ABSTRACT

SAS and Java are platforms for performing sophisticated analysis and implementing complex designs. Version 9 of SAS adds a new DECLARE statement to the DATA Step for creating objects. In order to use objects you must master 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** [2]. This new interface is a gateway to hybrid solutions that combine the best of Java and SAS. In this paper you will be guided through several Java Object examples. The final example is “Accessing a persistent Java hash table across multiple DATA Steps.”

INTRODUCTION

The reader should be have some knowledge of these topics:

- Configuring a SAS session
- DATA Step
- Writing and Compiling Java
- Java RMI

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 a large number of web sites related to Java.

JAVA SOURCE CODE

Use your favorite text editor or integrated development tool[3] 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 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:

```
$ SASROOT/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

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

Example1.java

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

Example1.sas

```
data _null_;
  length message $200;
  declare javaobj j ('Example1');
  j.callStringMethod ('getMessage', message);
  j.delete();
  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;
  }
}
```

FIELD NAMING

Class variable names should be nouns or noun phrases[4]. Method names should be verbs. Use descriptive names in the InfixCaps style; word{1}Word{2}…Word{N}. The first word is lowercase, and subsequent words start in uppercase.

METHOD NAMING

In this paper, class variables are never directly retrieved or updated. Their values are instead accessed through two methods; get*Property* and set*Property*, where *Property* is replaced with the actual property. These methods are 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 arguments 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 implemented 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]]);
```

Or the declaration and instantiation can be separated:

```
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 'getter and setter' coding pattern.

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.

Most DATA Steps instantiate a javaobj once. Guard against extra java objects by using "if _n_ = 1 then …". Java objects should be explicitly deleted, using `.delete()`, before a DATA Step ends; this will prevent memory leaks.

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

CLASS NOT FOUND

Example3.sas

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

UNINITIALIZED OBJECT

`java.net.URI` 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 attempting to invoke a method of an uninstantiated javaobj.

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.
```

NO SUCH 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".
```

WRONG SIGNATURE

Invoking a method whose name is correct, but the arguments form an incorrect signature. `URI` 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

A wrapper class must be created when a class constructor or method has a signature that is incompatible with SAS. Wrappers are most often a subclass or an adapter.

Consider the class `javax.swing.JoptionPane`. Amongst its methods are two for displaying dialogs, `showOptionDialog()` and `showMessageDialog()`. These methods have *integers* in their signature, thus they are not directly accessible to a SAS javaobj. Example 7 is coded in an adapter pattern that lets SAS utilize these methods. Displaying a Java dialog is useful in situations where a specific look and feel is desired.

Example7.java

```
import javax.swing.JOptionPane;

public class Example7 {
  private String optionSelected;

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

  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;
  }
}
```

Display the dialog and ask the object which option text was selected.

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 its functionality outside of SAS. Here is a modified version of Example7.java

Example 7.java with main()

```
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 command prompt. The following command assumes the file Example7.class is located in the current directory.

```
$ java –cp . Example7
```

JAVA ENVIRONMENT

This next example demonstrates 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(). 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.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;
    }
  }
}
```

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

A SAS error is generated when a Java method throws an exception. Catching exceptions and making information about the exception available should be part of the design of a wrapper class and a DATA Step. Version 9.1 will have Javaobj methods for dealing with exceptions; exceptionCheck (), exceptionClear () and exceptionDescribe ().

In this example the constructor throws an exception, causing 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');
  j.delete();
run;
--- log ---
ERROR: An error has occurred during class method
                       OM_NEW(3) of "DATASTEP.JAVAOBJ".
```

A slight modification catches the Exception and stores the message so it can be retrieved.

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

```
%macro putln(char_expr,newline='0a'x);
if ''=resolve(
'%put ' || tranwrd(&char_expr,&newline,'; %put ')
) then ;
%mend;

data _null_;
  dcl javaobj j ('Example10', 'foobar.file');
  length s $1000;
  j.callStringMethod ('getExceptionMessage', s);
  if s ne '' then %putln(s);
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)
```

GETTING A LIST OF FILES

ListFiles.java

```
import java.io.*;
import java.util.*;

public class ListFiles  {

  File[] files;
  int i = -1;

  public ListFiles () {
      ListFiles(".");
  }

  public ListFiles ( String path ) {
      File dir = new File(path);
      files = dir.listFiles();
  }

  public boolean getNext () {
    i++;
    return i<files.length;
  }

  public String getName () {
    if (i<files.length)
      return files[i].getName();
    else
      return "";
  }

  public String getType () {
    if (i<files.length)
      if (files[i].isDirectory())
        return "Dir";
      else
      if (files[i].isFile())
        return "File";
      else
        return "?";
    else
      return "";
  }

  public long getLength () {
    if (i<files.length)
      return files[i].length();
    else
      return Double.NaN;
  }

  public double getModifiedAsSasDateTime () {
    if (i<files.length) {
      Calendar saszero = Calendar.getInstance();
      saszero.set(1960,Calendar.JANUARY,1,0,0,0);
      long sasmillis = files[i].lastModified()
                     - saszero.getTimeInMillis();
      return sasmillis / 1000;
    }
    else
      return Double.NaN;
  }
}
```

ListFiles.sas

```
data files;
  length name $200 type $4 len mod b 8; drop b;
  format mod datetime19. len comma9.;
  dcl javaobj jlf ("ListFiles", "C:\WINNT");
  jlf.callBooleanMethod ("getNext", b);
  do while (b);
   jlf.callStringMethod("getName", name);
   jlf.callStringMethod("getType", type);
   jlf.callLongMethod  ("getLength", len);
   jlf.callDoubleMethod("getModifiedAsSasDateTime",mod);
   output;
   jlf.callBooleanMethod ("getNext", b);
  end;
  jlf.delete();
run;
```

READING DATA ARCHIVED IN A REMOTE ZIP FILE

Consider the situation of needing to read text data from files stored in a zip archived reposited on a web server. The archive is named "alarms-2000.zip" and contains text files, one file per month, named "*yyyy-mm*-alarms.log". Each line of data in these files is formatted as *day-of-month time-alarm-started duration-of-alarm alarm-type*. A DATA Step uses "ReadZippedFiles" to read the text lines from each file, and parses the lines using an input statement. Replacing the _infile_ buffer prior to the input statement is a 'trick' technique.

ReadZippedFiles

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.zip.*;

public class ReadZippedFiles {

  ZipInputStream zis = null;
  Enumeration e = null;
  ZipEntry ze = null;

  boolean zeLoopStarted = false;
  boolean zerLoopStarted = false;
  boolean zerEndOfFile = false;

  BufferedReader zer;

  public ReadZippedFiles ( String aZipFile ) {
    URL url = null;
    InputStream is = null;

    try
    { // is the string a URL ?
      url = new URL ( aZipFile );
      is = url.openStream();
    }
    catch (Exception e1)  {}

    if (url == null) {
      try { // is the string a filename ?
        is = new FileInputStream ( aZipFile );
      }
      catch (IOException e2)
      { }
    }

    if (is == null) return;

    zis = new ZipInputStream ( is  );
  }

  public boolean getNextEntry () {
    if (zis != null)
    try
    {
      if (! zeLoopStarted) {
        ze = zis.getNextEntry();
        zeLoopStarted = true;
      }
      else
      if (ze != null) {
        ze = zis.getNextEntry();
      }

      if (ze == null) {
        zis.close();
      }
    }
    catch (java.io.IOException ioe)
    { ze = null; }

    zerLoopStarted = false;
    zerEndOfFile = false;

    return (ze != null);
  }

  public String getEntryName () {
    if (ze != null)
      return ze.getName() ;
    else
      return "";
  }

  public String readLine () {
    try
    {
      if (! zerLoopStarted) {
        zer = new BufferedReader ( new
InputStreamReader ( zis ) ) ;
        zerLoopStarted = true;
      }
      String s = zer.readLine ();
      if (s == null) {
        zerEndOfFile = true;
        zis.closeEntry();
      }

      return s;
    }
    catch (java.io.IOException ioe)
    { return ""; }
  }

  public boolean endOfEntry () {
    return zerEndOfFile;
  }
}
```

ReadZippedFiles.sas

```
filename buffer catalog 'work.foo.bar.source';
* create dummy entry that infile will use;
data _null_;
  file buffer;
  put ' ';
run;

data alarms (keep=when duration type);
  format when datetime16.;
  format time duration time8.;
  length type $10;

  length archive entryname $200;
  length line $256;

  * would also work if archive is a local file;
  archive = 'http://www.devenezia.com/papers/sugi-
29/examples/alarms-2000.zip';

  * prep _infile_ buffer;
  infile buffer ;
  input @;

  declare javaobj jz ('ReadZippedFiles', archive);

  jz.callBooleanMethod ("getNextEntry", entryFound);

  do while (entryFound);
    jz.callStringMethod ("getEntryName", entryname);
    yearmo = substr(entryname,1,8);

    jz.callStringMethod ("readLine", line);
    jz.callBooleanMethod ("endOfEntry", eoe);

    do while (not eoe);
      * parse line of text with input statement;
      _infile_ = trim(yearmo) || line;
      input @1 date yymmdd10. time: time8. duration:
time8. type: @ ;
      put _infile_;
      when = dhms(date,0,0,0)+time;
      OUTPUT;
      jz.callStringMethod ("readLine", line);
      jz.callBooleanMethod ("endOfEntry", eoe);
    end;

    jz.callBooleanMethod ("getNextEntry", entryFound);
  end;

  jz.delete();
  stop;
run ;

filename buffer;
```

PERSISTENCE USING RMI AND A WRAPPER

Java objects are instantiated and referenced by DATA Step variables. This means a java object is only available while a DATA Step is running. There is no built-in mechanism for having a java object persist for use in a second DATA Step.

The final example demonstrates how to persist Java objects in the memory space of a server process. Such objects are available to different DATA Steps, or even different SAS sessions. The technique is by implementing a client / server system that utilizes the well established Java concept of remote method invocation (RMI). An RMI system is comprised of three components; an interface, a manager and a server. The interface defines what methods are available, the manager implements the methods and the server hosts an instance of the manager, achieving persistence. SAS javaobj variables can only be created through the _NEW_ operator, they can not be assigned as the result of a java objects method. This means that an additional adapter class is needed to allow SAS to hook into the RMI scheme. This adapter has to reimplement the methods of the interface using signatures that SAS can use. The adapter methods perform typecasting required to move SAS values into the RMI system.

The RMI scheme of the example implements a system that uses the concept of integer handles. A request is made to the server process to create a managed object and a handle is returned. Any future action is mediated through the handle (which acts as a surrogate object reference).

HashInterface.java

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

public interface HashInterface extends Remote
{
  // returned handle must be used to access all functions

  public int getHashHandle (int size, float load) throws RemoteException;

  public void freeHashHandle (int handle) throws RemoteException;

  public void put (int handle, String key, String value) throws RemoteException;
  public void put (int handle, String key, double value) throws RemoteException;

  public void put (int handle, double key, String value) throws RemoteException;
  public void put (int handle, double key, double value) throws RemoteException;

  public String getString (int handle, String key) throws RemoteException;
  public String getString (int handle, double key) throws RemoteException;

  public double getDouble (int handle, String key) throws RemoteException;
  public double getDouble (int handle, double key) throws RemoteException;
}
```

HashManager.java

To save space some methods are not shown

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.io.*;

public class HashManager
  extends UnicastRemoteObject
  implements HashInterface
{
  private Hashtable hashes;                // handle --> Hash
  private int numberOfHandles;

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

  //---------------------------------------------------------------------------
  public HashManager (int numberOfHandlesToManage) throws RemoteException
  {
    this.numberOfHandles    = numberOfHandlesToManage;
    this.hashes             = new Hashtable (numberOfHandlesToManage);
  }

  //---------------------------------------------------------------------------
  private Integer getNewHandle () {
    Integer handle;  // has to be an object so that it can be used in a Hashtable
    do {
      handle = new Integer ( (int) Math.floor (Math.random() * 1e8));
    }
    while (hashes.containsKey (handle));

    return handle;
  }

  //---------------------------------------------------------------------------
  private Object getObjectOfKey(Hashtable H, Object key) throws Exception
  {
    Object mv = H.get (key);

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

  //--------------------------------------------------------------------------
  private Hashtable getHashOfHandle(int handle) throws Exception
  {
    Integer key = new Integer(handle);
    return (Hashtable) getObjectOfKey (hashes, key);
  }

  //--------------------------------------------------------------------------
  public int getHashHandle (int size, float load) throws RemoteException
  {
    Integer hhandle = getNewHandle();
    Hashtable hash;

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

    try {
      hash = new Hashtable (size, load);
      System.out.println ("Hashtable allocated, handle="+hhandle);
    }
    catch (Exception e) {
      System.out.println (e);
      throw new RemoteException ("Problem allocating a Hashtable", e);
    }

    hashes.put (hhandle, hash);

    return hhandle.intValue();
  }

  //--------------------------------------------------------------------------
  public void freeHashHandle(int handle) throws RemoteException {
    try {
      getHashOfHandle(handle).clear();
      hashes.remove (new Integer(handle));
    }
    catch (Exception e) { throw new RemoteException("", e); }

    System.out.println ("Hashtable deallocated, handle was " + nf.format(handle));
  }

  //--------------------------------------------------------------------------
  public double getDouble(int handle, double key) throws RemoteException {
    try {
      Object value = getHashOfHandle(handle).get(new Double(key));
      return (value instanceof Double)
             ? ((Double)value).doubleValue()
             : Double.NaN;
    }
    catch (Exception e) { throw new RemoteException("", e); }
  }
...
  //--------------------------------------------------------------------------
  public void put(int handle, double key, double value) throws RemoteException {
    try { getHashOfHandle(handle).put(new Double(key), new Double(value));
    } catch (Exception e) { throw new RemoteException("", e); }
  }
...
}
```

HashServer.java

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

public class HashServer
{
  protected static final String RMI_NAME = "PERSISTENT-HASH-MANAGER";

  private String exceptionMessage;

  // This method is run when class is run as stand-alone java application,
  // from whence comes run-time persistence

  public static void main(String args[]){

    // sets 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
      Integer n = Integer.getInteger("number.of.hashes", 5);
      HashManager manager = new HashManager (n.intValue());

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

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

  //------------------------------------------------------------------------
  public static HashInterface 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 (HashInterface) remote;
  }

}
```

DataStepHashAdapter.java

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

// This wrapper class only exists because DATA Step can only deal with java objects created with new
// and thus can not receive the reference returned by HashServer.getReferenceToPersistentManager
//
// If DATA Step is ever updated to handle javaobj methods that return other java objects, this wrapper
// class will not be needed

public class DataStepHashAdapter
{
  private HashInterface hi;

  public DataStepHashAdapter() throws Exception {
    hi = HashServer.getReferenceToPersistentManager ();
  }

  public int getHashHandle (double size, double load) throws Exception
 { return hi.getHashHandle ( (int) size, (float)load); }

  public void freeHashHandle (double handle) throws Exception
         { hi.freeHashHandle ( (int) handle); }

  public void put (double handle, String key, String value) throws Exception
         { hi.put ( (int) handle,        key,        value); }

  public void put (double handle, String key, double value) throws Exception
         { hi.put ( (int) handle,        key,        value); }

  public void put (double handle, double key, String value) throws Exception
         { hi.put ( (int) handle,        key,        value); }

  public void put (double handle, double key, double value) throws Exception
         { hi.put ( (int) handle,        key,        value); }

  public String getString (double handle, String key) throws Exception
    { return hi.getString ( (int) handle,        key); }

  public String getString (double handle, double key) throws Exception
    { return hi.getString ( (int) handle,        key); }

  public double getDouble (double handle, String key) throws Exception
    { return hi.getDouble ( (int) handle,        key); }
  public double getDouble (double handle, double key) throws Exception
    { return hi.getDouble ( (int) handle,        key); }
}
```

HashMacros.sas

```
%*-------------------------------------------------;
%macro startServer
( policy=
, title=Hash Server
, define=
);
  %local restore;
  %let restore
  = %sysfunc(getoption(xsync))
    %sysfunc(getoption(xwait))
    %sysfunc(getoption(xmin))
  ;
  options noxsync noxwait xmin;

  x start "&title" java -Djava.security.policy=&policy &define HashServer ;

  options &restore;
%mend;
%*-------------------------------------------------;
%macro getHashHandle (size=100, load=0.75, handle_mv=);
  data _null_;
    declare javaobj ji ("DataStepHashAdapter");
    ji.callIntMethod ("getHashHandle",&size,&load,handle);
    ji.delete();
    put "Hash allocated, " handle=;
    call symput ("&handle_mv", trim(left(put(handle,best12.))));
  run;
%mend;
%*-------------------------------------------------;
%macro freeHashHandle (handle=);
  data _null_;
    declare javaobj ji ("DataStepHashAdapter");
    ji.callVoidMethod ("freeHashHandle", &handle);
    ji.delete();
  run;
%mend;
```

HashSample.sas

The hash is filled in one data step and read back in a second data step. Proc COMPARE verifies that values read back match those initially placed in the hash.

```
/* NOTE:
 * The jars or directories containing the java classes
 * DataStepHashAdapter
 * _must_ be listed in the environment variable
 * CLASSPATH _prior_ to the SAS session being started.
 */

%include "hash-macros.sas";

%let handle =;

options mprint;

%startServer (
  policy = persist.policy
, define = -Dnumber.of.hashes=10
)

%getHashHandle (handle_mv = handle, size=10000)

data foo;
  do i = 1 to 1e4;
    j = ranuni(0);
    output;
  end;
run;

data _null_;
  declare javaobj ji ("DataStepHashAdapter");
  do until (end);
    set foo end=end;
    ji.callVoidMethod ("put",&handle,i,j);
  end;

  ji.delete();
  stop;
run;

data bar;
  declare javaobj ji ("DataStepHashAdapter");
  do i = 1 to 1e4;
    ji.callDoubleMethod ("getDouble",&handle,i,j);
    output;
  end;
```

```
run;

%freeHashHandle ( handle = &handle )

ods listing;
proc compare base=foo compare=bar;
run;
```

This example was developed based on a question posed at SESUG 2003 regarding persistent hashes. It works as a proof of concept, however it is neither efficient nor sensible. Version 9 DATA Step has a native hash variable type which is extremely faster and more adaptable to the specifics of DATA Step. An example of persisting a JDBC connection can be found on my website[5].

CONCLUSION

The Data Step Component Interface is a useful addition to the SAS repertoire. A developer fluent in both sides of the interface can develop solutions that benefit from the utility of Java .

ACKNOWLEDGMENTS

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

This document was produced using OpenOffice.org Writer.

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] The Java Object and the DATA Step Component Interface
`http://support.sas.com/rnd/base/topics/datastep/dot/javaobj.html`

[3] netBeans IDE
`http://www.netbeans.org`

[4] Java™ Coding Style Guide
`http://wwws.sun.com/software/sundev/whitepapers/java-style.pdf`

[5] Persisting a JDBC connection across DATA Steps
`http://www.devenezia.com/papers/sesug-2003/`