Paper 008-2012

# A Guide for Connecting Java to SAS® Data Sets

Ryan Snyder, Institute for Advanced Analytics, North Carolina State University, Raleigh, N.C.

Patrick Hall, Institute for Advanced Analytics, North Carolina State University, Raleigh, N.C.

## ABSTRACT

This paper is written for Java programmers who want to access SAS data sets. We provide step-by-step examples to connect Java to SAS using three SAS technologies:

- the Base SAS® Java Object interface (Javaobj)
- SAS/SHARE®
- SAS/ACCESS® to a third-party database

The first example sets up a Java class and then drives it from Base SAS. The second and third examples build a simple Java applet and then use Java Database Connectivity (JDBC) code and drivers to read data from a server. Each example is based on the Windows operating system using the *sashelp.class* sample data.

## INTRODUCTION

Each programming language has its unique strengths. SAS reads and processes raw data efficiently and supports powerful statistical and analytic tools, often requiring only a few lines of code. Java, on the other hand, is a versatile Object Oriented language with a long list of third-party libraries developed on the Java Virtual Machine (JVM) and a widely-used Web platform.

SAS provides a number of connection methods to communicate with Java. The challenge of integrating any two (or more) software technologies, however, lies in lining up all of the pieces. Even practiced Java and/or SAS programmers may be left scratching their heads, trying to answer questions such as "Which technology should I use?", "Did I set this classpath correctly?" or "Am I using the correct drivers?".

We attempt to answer these and other basic questions using ground-up examples to access the *sashelp.class* sample data set from Java. *Sashelp.class* contains name, sex, age, height and weight information for 19 individual students.

## BASE SAS (JAVAOBJ)

The easiest method for getting SAS data into Java is to instantiate a Java object from within SAS using the Javaobj construct. This method has the fewest prerequisites; all it takes is SAS version 9.1.3 or newer and a JVM installed and working.

This example is demonstrated using the Eclipse development environment, though a text editor and Java command line tools would work just as well.

## STRUCTURING JAVA CLASSES

We know that each student has five attributes (name, sex, age, height and weight), so we can create a new Java object named "Student" to store the attributes of individual students.

```
package classdemo;

class Student {
    // attributes for each student
    String name;
    String sex;
    double age;
    double height;
    double weight;

    // constructor
    public Student(String nm, String sx, double ag, double ht, double wt) {
        name = nm;
        sex = sx;
        age = ag;
        height = ht;
        weight = wt;
    }
}
```

We should add a data structure to store our Student objects, and a public method that allows SAS to add information to it. In this example we've used a separate Java class named "Classroom" with an ArrayList data structure for our students.

```
package classdemo;

import java.util.ArrayList;

public class Classroom {
    // list of student objects
    private ArrayList<Student> students = new ArrayList<Student>(19);

    // add a student object
    public void addStudent(String name, String sex, double age, double height,
                           double weight) {
        students.add(new Student(name, sex, age, height, weight));
    }
}
```

We'll add a method to this class that prints out the student information in the list, so we can verify that the data are added properly.

```
// print the information of each student in the list
public void listAllStudents() {
    for (Student student : students)
        System.out.println(student.name + ": Gender=" + student.sex
            + " Age=" + student.age + " Height=" + student.height
            + " Weight=" + student.weight);
}
```

We'll also add a main() method to test our Java class before we call it from SAS. (Java classes typically run via a main() method. We can use main() to add and print sample student data by calling our methods.)

```
// a main method for testing this class
public static void main(String[] args) {
     Classroom c = new Classroom();

     // add two students (name, sex, age, height, weight)
     c.addStudent("Student A", "M", 14, 65, 100);
     c.addStudent("Student B", "F", 13, 55, 80.0);

     System.out.println("");
     c.listAllStudents();
}
```

Output 1 shows the output from executing this Java program.

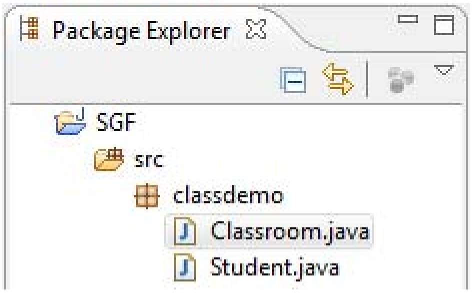
```
Student A: Gender=M Age=14.0 Height=65.0 Weight=100.0
Student B: Gender=F Age=13.0 Height=55.0 Weight=80.0
```

**Output 1. Java Program Output Using Two Sample Students**

## SETTING UP THE JAVA CLASSPATH IN SAS

In order to access the compiled Java classes, the Base SAS JVM needs to know where to find them. This can be accomplished for SAS 9.2 by setting the local CLASSPATH environment variable or by invoking SAS with the appropriate arguments.

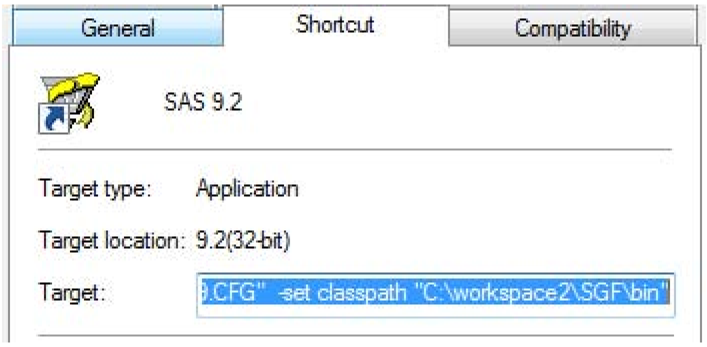
In this example the Java classes are saved and compiled in an Eclipse workspace.



**Display 1. Demonstration Project Structure in The Eclipse Package Explorer**

The project name is "SGF" and Eclipse compiles the Java classes into a "bin" directory. The actual Windows path to this example project is "C:\workspace2\SGF\bin".

To set the classpath in this example we can invoke Base SAS with the -set classpath options from the command line or by updating a Windows shortcut. Display 2 shows a sample Windows shortcut dialog.



SAS 9.2

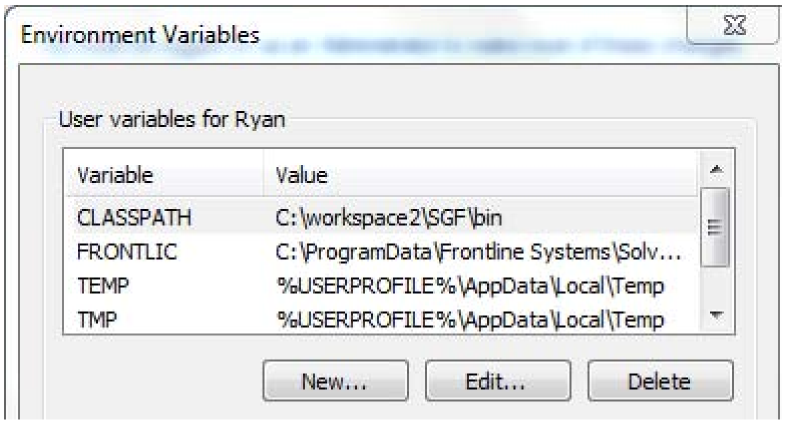
Target type: Application

Target location: 9.2(32-bit)

Target: 9.CFG" -set classpath "C:\workspace2\SGF\bin"

**Display 2. Windows Shortcut for Base SAS 9.2 with -set classpath Option**

The same effect can be achieved by setting a CLASSPATH environment variable. This can be located on Windows XP and Windows 7 by right-clicking “My Computer” → "Properties" → "Advanced" tab → "Environment Variables" and adding or updating a CLASSPATH environment variable. Display 3 shows a sample Windows Environment Variables dialog.


Environment Variables

User variables for Ryan

| Variable | Value |
|---|---|
| CLASSPATH | C:\workspace2\SGF\bin |
| FRONTLIC | C:\ProgramData\Frontline Systems\Solv... |
| TEMP | %USERPROFILE%\AppData\Local\Temp |
| TMP | %USERPROFILE%\AppData\Local\Temp |

**Display 3. CLASSPATH Windows User Environment Variable**

If SAS is unable to locate the Java class the following error may be displayed in the SAS log:

```
ERROR: Could not find class classdemo/Classroom at line 5 column 36.  Please ensure
that the
       CLASSPATH is correct.
ERROR: DATA STEP Component Object failure.  Aborted during the EXECUTION phase.
java.lang.ClassNotFoundException: classdemo.Classroom
```

**Output 2. SAS Error Message if Java Class not Found**

Referenced Java classes or third-party libraries must be included on the classpath as well (if any are used). For example, if the Classroom Java class referenced *core.jar* from the *Processing* graphics library, then the classpath might be "C:\workspace2\SGF\bin;C:\processing-1.5.1\lib\core.jar" instead.

## WRITING SAS CODE TO INTERFACE WITH THE JAVA CLASS

The final step in this method is to write SAS code that uses the SAS Javaobj construct.

Here is a simple example that instantiates the Java object, iteratively adds each student, then calls a method to print all of the students.

```
data _null_;
    set sashelp.class end=last;

    /* instantiate the Java object */
    if _n_=1 then declare javaobj j("classdemo/Classroom");

    /* add students to the Java data structure */
    j.callVoidMethod("addStudent", name, sex, age, height, weight);

    /* reports the students in the data structure */
    if last then j.callVoidMethod("listAllStudents");
run;
```

Output 3 shows part of the Java output displayed by the SAS log.

```
Alfred  : Gender=M Age=14.0 Height=69.0 Weight=112.5
Alice   : Gender=F Age=13.0 Height=56.5 Weight=84.0
Barbara : Gender=F Age=13.0 Height=65.3 Weight=98.0
  ⁞
William : Gender=M Age=15.0 Height=66.5 Weight=112.0
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: DATA statement used (Total process time):
      real time           0.05 seconds
      cpu time            0.06 seconds
```

**Output 3. Java Output in SAS Log**

### JAVAOBJ SUMMARY

This method has the advantages of being very easy to implement and it requires only Base SAS with no additional licenses. This can be a great way to test a Java function such as a graphics library with new data.

There are several limitations to this approach:

1. It only allows Java applications, not applets or other specific uses of Java.
2. SAS does not recognize certain Java data types and cannot accept returned Java objects.
3. If a Java class exits (by calling System.exit(), for example) then the SAS JVM terminates! This prevents the SAS session from working with any more Java objects. (This is a noticeable problem with Java GUI components, which tend to exit when a window is closed).

Additional details of the Javaobj construct are available in the SAS online documentation and in other SGF papers. See the listed references for more information.

## JAVA DATABASE CONNECTIVITY (JDBC)

JDBC is a Java technology designed to allow reusable code that offers read/write access to data. One set of Java code can be used to connect to different data sources. The examples below use the same Java code to connect to a SAS/SHARE server and a MySQL database with data loaded using SAS/ACCESS.

All JDBC connections require a client driver jar to be available in the Java project classpath.

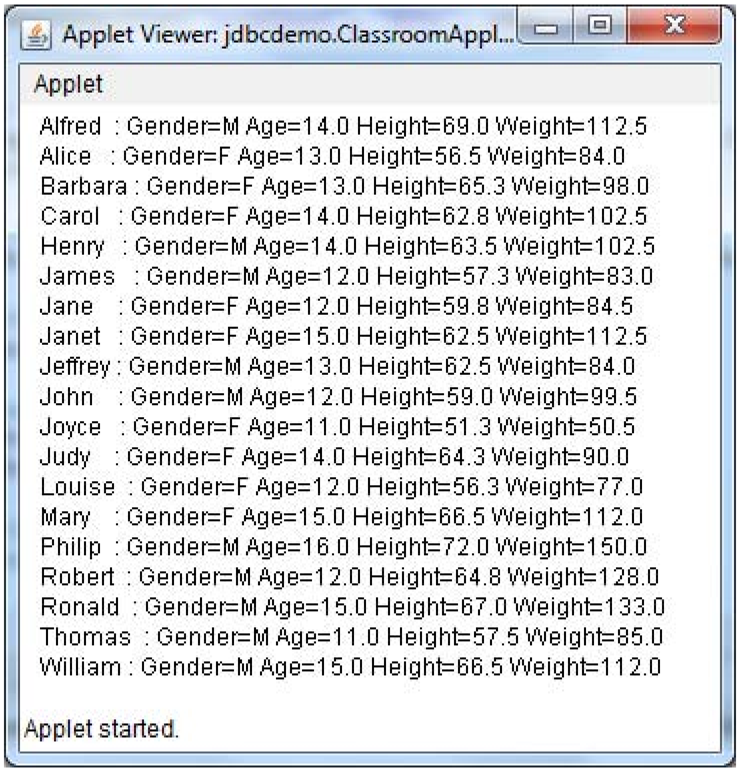
### CREATING A JAVA APPLET

To demonstrate these technologies, we will use a Java applet instead of a Java class. Java applets are a Web-friendly aspect of Java and are designed to be embedded in Web pages (within HTML tags).

The applet Java code uses a Student object and a data structure like the above example, but uses JDBC connection methods (highlighted in gray) to load the student data. The paint() method displays output to the applet screen.

```
package jdbcdemo;

import java.util.*;
import java.applet.*;
import java.awt.*;
import java.sql.*;

public class ClassroomApplet extends Applet {
    private static final long serialVersionUID = 1L;

    // list of student objects
    private ArrayList<Student> students = new ArrayList<Student>(19);

    // init() is called first
    public void init() {
        setSize(350, 300); // size the applet so all student information fits

        // create a connection and load student information
        try {
            Connection conn = getConnection();
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("select * from sashelp.class");

            // load student information from the connection
            while (rs.next()) {
                String name = rs.getString("Name");
                String sex = rs.getString("Sex");
                double age = rs.getDouble("Age");
                double height = rs.getDouble("Height");
                double weight = rs.getDouble("Weight");

                students.add(new Student(name, sex, age, height, weight));
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }
    }

    // standard method to draw on the applet
    public void paint(Graphics g) {
        int yPosition = 0;

        for (Student student : students) {
            yPosition += 15;
            g.drawString(student.name + ": Gender=" + student.sex +
                         " Age=" + student.age + " Height=" + student.height +
                         " Weight=" + student.weight, 10, yPosition);
        }
    }
}
```

Display 4 shows the applet view after the data is loaded.



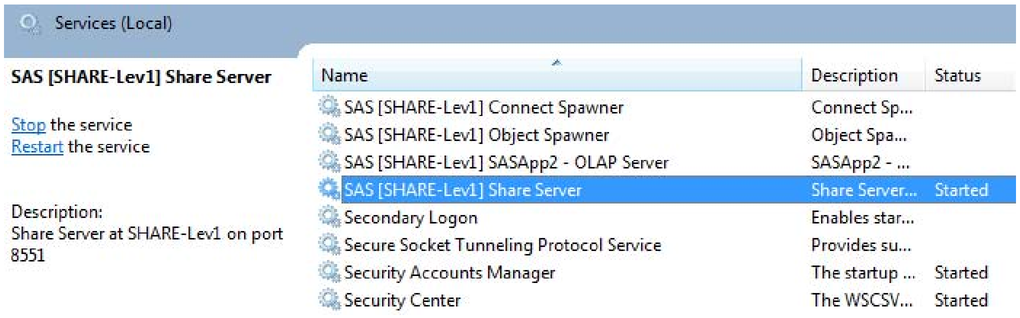
```
Alfred  : Gender=M Age=14.0 Height=69.0 Weight=112.5
Alice   : Gender=F Age=13.0 Height=56.5 Weight=84.0
Barbara : Gender=F Age=13.0 Height=65.3 Weight=98.0
Carol   : Gender=F Age=14.0 Height=62.8 Weight=102.5
Henry   : Gender=M Age=14.0 Height=63.5 Weight=102.5
James   : Gender=M Age=12.0 Height=57.3 Weight=83.0
Jane    : Gender=F Age=12.0 Height=59.8 Weight=84.5
Janet   : Gender=F Age=15.0 Height=62.5 Weight=112.5
Jeffrey : Gender=M Age=13.0 Height=62.5 Weight=84.0
John    : Gender=M Age=12.0 Height=59.0 Weight=99.5
Joyce   : Gender=F Age=11.0 Height=51.3 Weight=50.5
Judy    : Gender=F Age=14.0 Height=64.3 Weight=90.0
Louise  : Gender=F Age=12.0 Height=56.3 Weight=77.0
Mary    : Gender=F Age=15.0 Height=66.5 Weight=112.0
Philip  : Gender=M Age=16.0 Height=72.0 Weight=150.0
Robert  : Gender=M Age=12.0 Height=64.8 Weight=128.0
Ronald  : Gender=M Age=15.0 Height=67.0 Weight=133.0
Thomas  : Gender=M Age=11.0 Height=57.5 Weight=85.0
William : Gender=M Age=15.0 Height=66.5 Weight=112.0

Applet started.
```

**Display 4. ClassroomApplet with Data Loaded from JDBC Connection**

## SAS/SHARE CONNECTION

A running SAS/SHARE server supports database-like read and write access to shared SAS libraries and data sets. Connecting to a SAS/SHARE server requires at least the *sas.intrnet.javatools.jar* and *sas.core.jar* client driver jars. These jars may be found in your local SASHOME directory or downloaded from SAS.

This example assumes the basic scenario that a SAS/SHARE server is installed and running locally as a Windows service on the default port, 8551. The Windows Services tools or the SAS Management Console can retrieve information about this service.



Description:
Share Server at SHARE-Lev1 on port 8551

| Name | Description | Status |
|---|---|---|
| SAS [SHARE-Lev1] Connect Spawner | Connect Sp... | |
| SAS [SHARE-Lev1] Object Spawner | Object Spa... | |
| SAS [SHARE-Lev1] SASApp2 - OLAP Server | SASApp2 - ... | |
| SAS [SHARE-Lev1] Share Server | Share Server... | Started |
| Secondary Logon | Enables star... | |
| Secure Socket Tunneling Protocol Service | Provides su... | |
| Security Accounts Manager | The startup ... | Started |
| Security Center | The WSCSV... | Started |

**Display 5. Windows Services Tool with SAS Share Server Highlighted**

Creating and managing SAS/SHARE servers is outside of the scope of this paper, but the references section includes a link to the SAS documentation.

### JDBC CONNECTION CODE TO SAS/SHARE

The following code added to the Java applet class above will connect to the default local SAS/SHARE server. (*Be sure to update the user name and password information.*)

```
public static Connection getConnection() throws Exception {
    return getShareConnection("localhost", "8551", "PC\\user", "password");
}

public static Connection getShareConnection(String serverHost, String serverPort,
    String user, String password) throws Exception {
    // JDBC connection url
    String URL = "jdbc:sharenet://" + serverHost + ":" + serverPort;

    // Connection properties
    Properties prop = new Properties();
    prop.setProperty("shareUser", user);
    prop.setProperty("sharePassword", password);

    com.sas.net.sharenet.ShareNetDriver snd = new
       com.sas.net.sharenet.ShareNetDriver();
    Connection conn = snd.connect(URL, prop);

    return conn;
}
```

## SAS/ACCESS CONNECTION (VIA MYSQL)

This method adds a database layer between the Java code and the SAS data. SAS/ACCESS may be used to upload data to a third-party database. JDBC connection code can then read and write to and from a database much like using SAS/SHARE.

Like SAS/SHARE (and all JDBC connections), MySQL requires a client driver jar. This example uses the *mysql-connector-java-5.1.18-bin.jar*, available from MySQL's webpage.

This example assumes the basic scenario that a MySQL server is installed and running locally on the default port, 3306. This example is based on SAS 9.2 (32-bit) with the SAS/ACCESS hotfix and MySQL 5.1 (32-bit).

### CREATE DATABASE IN MYSQL

After the MySQL database is installed and running as a Windows service (like the SAS/SHARE server), we need to create a database so SAS can upload data.

The following code submitted in the MySQL command line client will create a new database named "sgf" that we can upload data to.

```
mysql> create database sgf;
mysql> commit;
mysql> show databases;
```

```
Query OK, 1 row affected (0.01 sec)
Query OK, 0 rows affected (0.00 sec)
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| sgf                |
| test               |
+--------------------+
4 rows in set (0.00 sec)
```

**Output 4. MySQL Output from Creating Database "SGF"**

## LOAD DATA TO MYSQL

The following SAS code will upload the *sashelp.class* data to MySQL using a database named "SGF" and a table named "class".

```
libname mysqllib mysql user=root password=password database=SGF server=localhost
         port=3306;

data mysqllib.class;
    set sashelp.class;
run;
```

This code uses SAS/ACCESS to connect directly to MySQL via a libname statement. It connects to the "SGF" database we created above using a server installed on the local machine and default port (3306).

The uploaded data can be verified from within the MySQL command line client.

```
mysql> use sgf;
mysql> show tables;
mysql> select * from class;
```

```
Database changed
+----------------+
| Tables_in_sgf |
+----------------+
| class          |
+----------------+
1 row in set (0.00 sec)

+---------+------+------+--------+--------+
| Name    | Sex  | Age  | Height | Weight |
+---------+------+------+--------+--------+
| Alfred  | M    |   14 |     69 |  112.5 |
| Alice   | F    |   13 |   56.5 |     84 |
| Barbara | F    |   13 |   65.3 |     98 |
  ⁞
| William | M    |   15 |   66.5 |    112 |
+---------+------+------+--------+--------+
19 rows in set (0.00 sec)
```

**Output 5. MySQL Output from Full Query of Database "SGF" (After Loading *sashelp.class* Data)**

## JDBC CONNECTION CODE TO SAS/ACCESS

The SAS/ACCESS Java connection code is very similar to the SAS/SHARE Java code above.

```
public static Connection getConnection() throws Exception {
    return getMySQLConnection("localhost", "sgf", "root", "password");
}

public static Connection getMySQLConnection(String serverHost, String db,
    String user, String password) throws Exception {

    Connection conn = null;
    String url = "jdbc:mysql://" + serverHost + "/" + db;
    conn = DriverManager.getConnection(url, user, password);
    System.out.println("Database connection established");

    return conn;
}
```

There are a few subtle differences between this code and the SAS/SHARE code. The most important to note is that this code uses a different URL string that requires the database name ("sgf") rather than the port number.

We also need to change our query to match the MySQL query (since we don't have the *sashelp* table name).

```
ResultSet rs = stmt.executeQuery("select * from class");
```

## JDBC SUMMARY

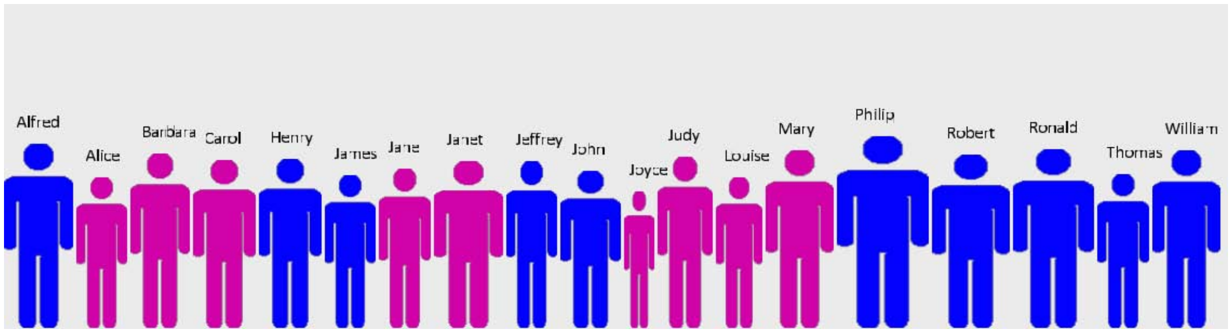
JDBC technology is a versatile method for accessing data. It supports reading different data types and is available for specialized uses of Java, such as Web applets. Additionally, most of the Java code is reusable between different third-party data sources.

This method requires additional work, both to set up client driver jars and to write Java connection code. The examples described both require a server to be set up and running. Also both require a special license, either for SAS/ACCESS or SAS/SHARE, to access data managed by SAS.

## CONCLUSION

The examples provided are a starting point for accessing SAS data from Java code. Once the connection is made, it opens up the flexibility of Java to the powerful data and statistical tools of SAS.

The graphic in Display 6 was generated for the *sashelp.class* data using the Java-based *Processing* graphics library; additional applications of Java will be left to the creativity of the reader.



**Display 6. Custom Java Graphic for *sashelp.class* Data**

(Source code and Java projects describes in this paper, including code for this graphic, are available through the authors' website listed in the references section.)

## REFERENCES

- Hall, Patrick and Snyder, Ryan. "Data Visualization Using SAS® and Processing." March 7, 2012. Available at https://sites.google.com/site/stamb2012/.
- "Base SAS: The Java Object and the DATA Step Component Interface." Jan 4, 2012. Available at http://support.sas.com/rnd/base/datastep/dot/javaobj.html.
- DeVenezia, Richard A. "Greetings from the Edge: Using javaobj in DATA Step." *SUGI 29*. Cary, NC: SAS Institute. Available at http://www2.sas.com/proceedings/sugi29/033-29.pdf.
- DeVenezia, Richard A. "Java in SAS®: JavaObj, a DATA Step Component Object." *SUGI 30*. Cary, NC: SAS Institute. Available at http://www2.sas.com/proceedings/sugi30/241-30.pdf.
- "SAS Drivers for JDBC." Jan 4, 2012. Available at http://support.sas.com/documentation/onlinedoc/jdbc/index.html.
- "SAS/SHARE® 9.2 User's Guide ". April 15, 2008. Available at http://support.sas.com/documentation/cdl/en/shrref/59595/HTML/default/a000585192.htm.
- "MySQL :: The world's most popular open source database." Jan 21, 2012. Available at http://www.mysql.com/.
- "Usage Note 36146: SAS® 9.2 does not support the use of Sun Microsystem's MySQL 5.1 client with SAS/ACCCESS® Interface to MySQL". Jan 4, 2012. http://support.sas.com/kb/36/146.html.

## ACKNOWLEDGMENTS

Acknowledgements go to:

- Patrick Hall, who had the initial inspiration of trying to work with data from Java. He also worked with SAS tech support to fix our Java classpath woes.
- The staff at the Institute for Advanced Analytics who encouraged us to work on this paper and provided advice, resources and proofreading.
- The friendly folks at SAS, including instructors, tech support and personal friends.

## RECOMMENDED READING

Hall, Patrick and Snyder, Ryan. "Let the Data Paint the Picture: Data-Driven, Interactive and Animated Visualizations Using SAS®, Java and the Processing Graphics." *SAS Global Forum 2012*. Cary, NC: SAS Institute.

## CONTACT INFORMATION

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

Ryan Snyder
ryancsnyder@gmail.com
http://www.linkedin.com/in/ryancsnyder

Patrick Hall
jpatrickhall@gmail.com
http://www.linkedin.com/in/jpatrickhall

Institute for Advanced Analytics
920 Main Campus Drive, Suite 530
Raleigh, NC 27606
http://analytics.ncsu.edu/