

Abstract

In a modern world where web is becoming increasingly significant personally and professionally, more and more, **web applications** are replacing old-fashioned softwares installed on computer.

Main benefits are an easy and a common way to access for end-users and the possibility to externalize your IT in dedicated infrastructure like **Cloud Computing**.

With that in mind and focusing on study biostatistical programming needs and how it could be possible to offer full services through web technologies, an important first question will have to be raised: What are the technical possibilities to **render a SAS dataset in a web-browser** ? Is that even possible ?

The aim of this poster is to answer to this question, analyze different ways to succeed and compare them.

Java



We decided to use Java technology to **read SAS datasets**.

Why Java ? Because it is a **polyvalent** programmatic language, very **popular** for server side management when it comes to work on web application development.

Moreover, there are a lot of **bridges** between SAS and Java, like Java material provided by SAS (jar files) or SAS Stored Process (exposed as web services).

Challenge is to be able to read SAS datasets structure and data through Java and store that in Java objects.

JSON



When SAS datasets data will be stored in Java objects, it will be necessary to communicate with client side of your web application (interface) to load these data and display it.

Most of the web application interfaces are now implemented using **HTML 5 and JavaScript framework** (like AngularJS). The most common way to do that is to convert your Java data in **JSON** (JavaScript Object Notation) format, which is an open standard format pretty well adapted for JavaScript environment. There are multiple ways to convert your data from Java to JSON, like **Jackson** library.



- Doesn't work with most of the SAS datasets we test with, especially those generated with last SAS Releases (v9.2, v9.3 and v9.4)
- Doesn't seem to be upgraded or maintained anymore



- Lot of options to specify what kind of information you want to retrieve
- Open source Java library

Example of Java code using SassyReader:

```
File file = new File("C:/test/datasets/dataset.sas7bdat");
CountingSasReaderCallback callback = new CountingSasReaderCallback(true);

//Read SAS dataset
SasReader sasReader = new SasReader(file);
sasReader.read(callback);

List<Object[]> data = callback.getSampleRows();

//Retrieve data
for (Object[] objs : data) {
    for (int i = 0; i < objs.length + 1; i++) {
        Object obj = objs[i];
        System.out.println(obj.toString());
    }
}
```

SassyReader is an Open source Java library that simply offer a way of reading the data that is stored in SAS dataset files.

It is a portage of the **SAS7BDAT** project that have been implemented in R language.

Though a unique interface, you can precisely define what you want to retrieve from the SAS dataset : structure, columns type, row data, ...

Unfortunately, SassyReader **was not working** correctly when we test it on multiple recent SAS datasets.

SassyReader

SAS/SHARE software is a data server that allows multiple users to gain simultaneous access to SAS files.

SAS provides Java DataBase Connectivity (JDBC) drivers to connect and interacts with SAS datasets stored on SAS/SHARE servers.

Using the com.sas.net.sharenet.ShareNetDriver Java class, you can **create a connection** to your SAS/SHARE server and use **SQL queries** to retrieve information from your SAS datasets.

With access to the complete features of JDBC, it is easy to create Java programs to store SAS dataset data in Java objects.

Example of Java code using JDBC connection on SAS/SHARE:

```
try {
    //JDBC URL connection
    String jdbcUrl = "jdbc:sharenet://your_server_host:" +
        "your_server_ports";

    Properties prop = new Properties();
    prop.setProperty("shareuser", "your_user");
    prop.setProperty("sharepassword", "your_password");
    prop.setProperty("libref", "saslib c:\\test\\datasets");

    ShareNetDriver end = new ShareNetDriver();
    Connection conn = end.connect(jdbcUrl, prop);

    //Create and execute the query to read SAS datasets
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT sup_id, " +
        "sup_name FROM saslib.suppliers");

    //Browse the results and store them in Java
    while (rs.next()) {
        Double id = rs.getDouble(1);
        String name = rs.getString(2);
        System.out.println(id + " " + name);
    }

    stmt.close();
    conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Using SAS/SHARE



- Need to have and to use SAS/SHARE in your IT environment
- Consume SAS resources



- JDBC drivers (jar files) provided by SAS
- Well known framework (JDBC) by Java developers
- Flexibility to use SQL queries to retrieve the SAS dataset data
- Good performance of the SAS dataset reading operations
- Should work correctly for future SAS Releases because it is provided by the editor

Explore ways to render SAS datasets through Web

Pascal Verrecchia

pascal.verrecchia@sokaris.fr
Sokaris Informatique



SOKARIS

Parso is a Java **lightweight library** to parse SAS dataset files.

It offers a lot of functionalities to access to the SAS dataset properties like row and columns count, OS type, compression methods, SAS Release version, ...

To read the rows, you have two main methods:

- Read all rows at the same time and store everything in Java objects
- Read data row by row

In case you're working on **big datasets** (more than 1 GB), you could have **troubles with Java memory** if you try to read all rows in one step.

We have **successfully tested** Parso with datasets generated from different OS and from several SAS Releases.

Example of Java code using Parso:

```
File f = new File("C:/test/datasets/dataset.sas7bdat");

try {
    InputStream is = new FileInputStream(f);
    SasFileHeader sfr = new SasFileHeader(is);
    SasFileProperties sfp = sfr.getSasFileProperties();
    System.out.println("Name : " + sfp.getName());
    System.out.println("File type : " + sfp.getFileType());
    System.out.println("Row count : " + sfp.getRowCount());
    System.out.println("Row length : " + sfp.getRowLength());
    System.out.println("OS type : " + sfp.getOsType());
    System.out.println("Page count : " + sfp.getPageCount());
    System.out.println("Page length : " + sfp.getPageLength());
    System.out.println("Creation date : " + sfp.getCreationDate());
    System.out.println("Modification date : " + sfp.getModificationDate());
    System.out.println("Server type : " + sfp.getServerType());
    System.out.println("SAS release : " + sfp.getSasRelease());
    System.out.println("Columns count : " + sfp.getColumnCount());
    System.out.println("Compression method : " + sfp.getCompressionMethod());
    List<Column> columns = sfr.getColumns();
    int i = 1;
    System.out.println("Column list : ");
    for (Column c : columns) {
        System.out.println("    Column " + i + " : ");
        i++;
        System.out.println("        Format : " + c.getFormat());
        System.out.println("        Id : " + c.getId());
        System.out.println("        Label : " + c.getLabel());
        System.out.println("        Length : " + c.getLength());
        System.out.println("        Name : " + c.getName());
    }

    Object[][] allRows = sfr.readAll();
    for (int j = 0; j <= allRows.length - 1; j++) {
        Object obj = allRows[j];
        System.out.println(obj);
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

Parso

Using IOM server

Integrated Object Model (IOM) in SAS Integration Technologies provides distributed object interfaces to SAS software features.

There are four types of IOM servers:

- SAS Workspace server
- SAS Stored Process server
- SAS OLAP server
- SAS Metadata server

SAS provides Java DataBase Connectivity (JDBC) drivers to connect and interacts with SAS datasets through IOM servers.

Using the com.sas.rio.MVADriver Java class, you can **create a connection** to your IOM server and use **SQL queries** to retrieve information from your SAS datasets.

With access to the complete features of JDBC, it is easy to create Java programs to store SAS dataset data in Java objects.

Example of Java code using JDBC connection on IOM server:

```
try {
    //JDBC URL connection
    String jdbcUrl = "jdbc:sasrio://your_server_host:" +
        "your_server_ports";

    Class.forName("com.sas.rio.MVADriver");
    Properties prop = new Properties();
    prop.setProperty("shareuser", "your_user");
    prop.setProperty("sharepassword", "your_password");
    prop.setProperty("libref", "saslib c:\\test\\datasets");

    Connection conn = DriverManager.getConnection(jdbcUrl, prop);

    //Create and execute the query to read SAS datasets
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT sup_id, " +
        "sup_name FROM saslib.suppliers");

    //Browse the results and store them in Java
    while (rs.next()) {
        Double id = rs.getDouble(1);
        String name = rs.getString(2);
        System.out.println(id + " " + name);
    }

    stmt.close();
    conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
```



- JDBC drivers (jar files) provided by SAS
- Well known framework (JDBC) by Java developers
- Flexibility to use SQL queries to retrieve the SAS dataset data
- Good performance of the SAS dataset reading operations
- Should work correctly for future SAS Releases because it is provided by the editor



- Need to have and to use IOM server in your IT environment
- Consume SAS resources



- Easy to retrieve a lot of information regarding SAS dataset
- Works very well on SAS datasets from different OS and SAS Release version
- Don't use any SAS process or SAS connection
- Open source Java library (GNU Licence 3)

- Nothing available to retrieve row data with delimiters (row line ... to line ...), not possible to use lazy loading