Paper 065-2012

Using Pre-Formatted Microsoft Excel Templates with SAS®

George Mendoza, CIGNA Healthcare, Bloomfield, CT
Subhashree Singh, The Hartford, Hartford, CT

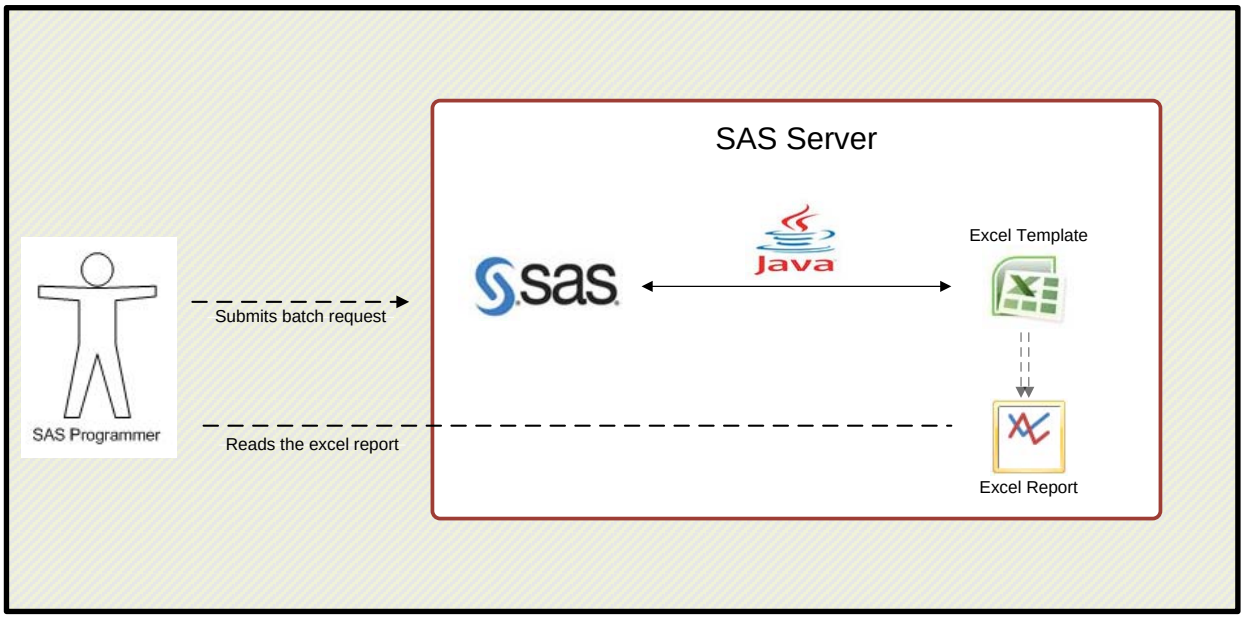
ABSTRACT

SAS® has several different options for transferring data to Excel. However, exporting data into a pre-formatted Excel template consisting of predesigned tables and charts can be challenging. Usually, there is some manual intervention required particularly in a Unix environment. The solution to leveraging pre-formatted excel templates in a server environment is the use of SAS Integration Technologies to serve as a bridge between base SAS and the excel template. This paper will walk you through the steps required to setup this bridge so that you can begin to use excel templates together with SAS in an automated server environment.

INTRODUCTION

When developing reports in Excel it is much easier to format the report template in native Excel and simply put dynamic SAS data in appropriate worksheet cells to build a dynamic report. There are a number of methods to programmatically transfer SAS data into Excel but there is no automatic method to programmatically transfer data into a pre-defined Excel report. Usually, a manual step is required and that prevents automatic excel report generation in a server environment. The NESUG paper titled “Excellent Ways of Exporting SAS Data to Excel” contributed by Ralph Winters (link given in the reference section) describes different methods for transferring SAS data to Excel but note that it does not have a method to transfer data to a pre-formatted Excel report template. But there is a solution. Thanks to SAS Integration Technologies, we can leverage the powerful features of Java to dynamically use SAS with pre-formatted Excel report templates. This paper will walk you through the process of using Java as a bridge between SAS and Excel, providing several code samples along the way. Note that other than SAS, all technologies used in this paper are open source and available free of cost, so no additional monetary investments are required to use this solution.

DIAGRAMMATIC OVERVIEW



SAS – Java (Bridge) - Excel

Figure 1: SAS – Java – Excel diagrammatic overview

THE GOAL

We will now look at an example step by step. We'll begin by defining what we want to accomplish. Take for example an excel workbook with 2 worksheets. The "Report" worksheet contains a bar chart and data table showing profits for a company across 4 years. The chart and data table references data on the "Input" worksheet. The complete report is as shown in the screen shots below.

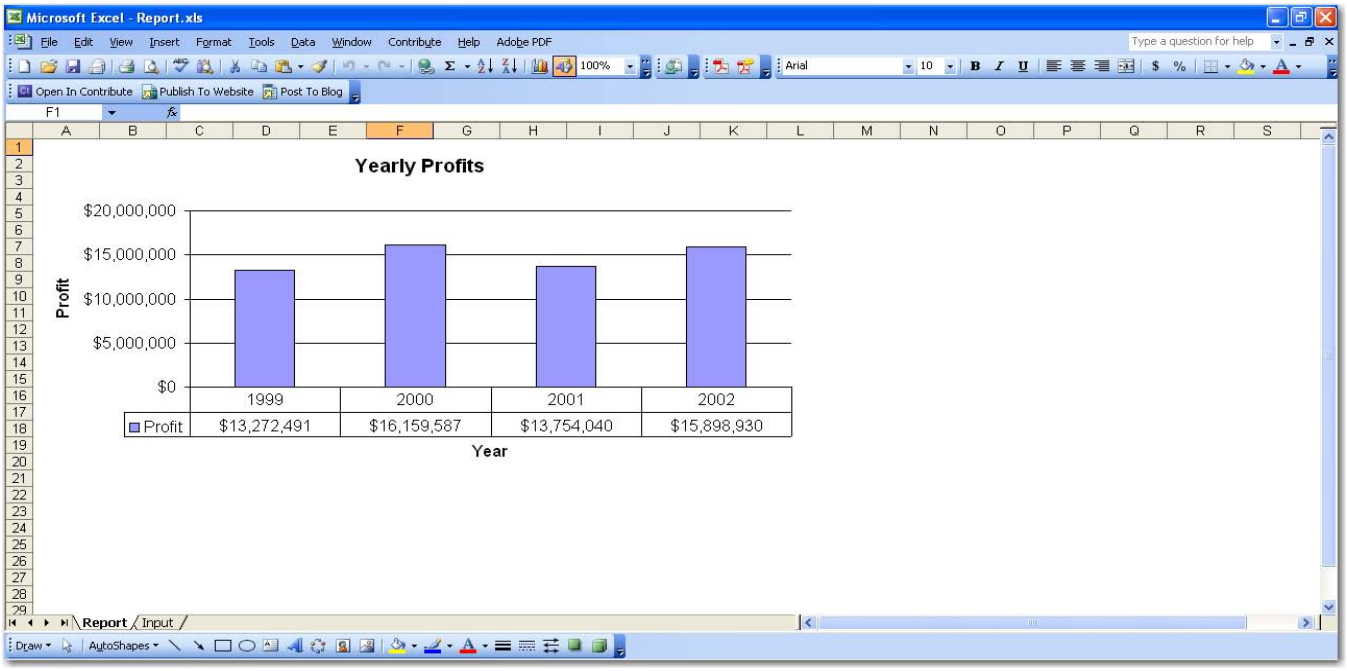


Figure 2: Report worksheet containing a chart and data table

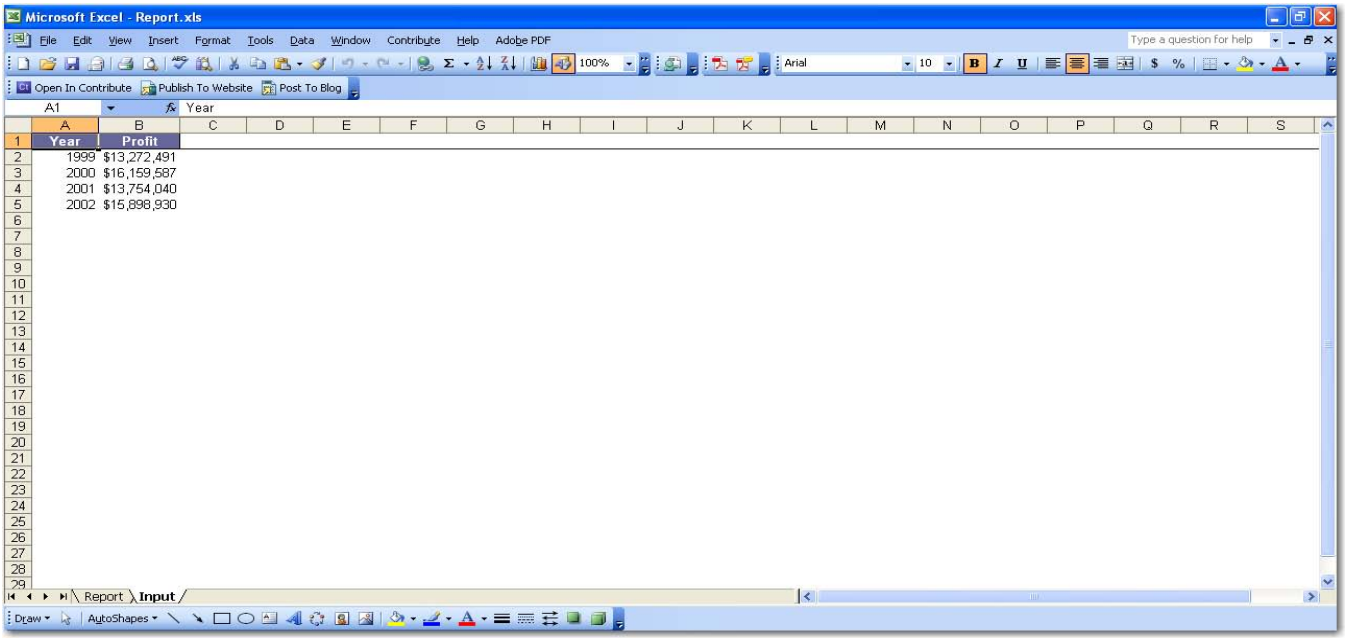


Figure 3: Input worksheet showing the raw data that is referenced by the chart and data table.

We shall save a copy of this report template without the raw data on the SAS Unix server. The below screen shots show you what the excel template looks like without the raw data.

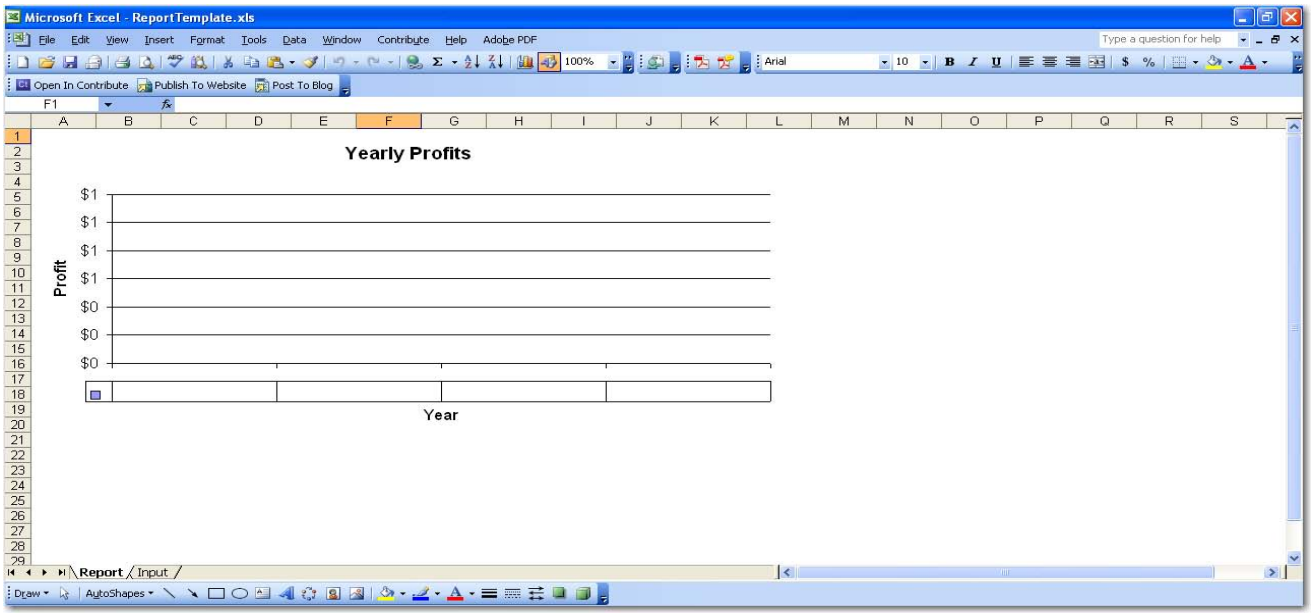


Figure 4: Report template without data.

Our goal is prepare the raw data in SAS and send that raw data to the "Input" worksheet or the excel report template. The chart will reference the data on the "Input" worksheet automatically and the "Report" worksheet will be transformed from figure 3 to figure 1. All the files needed to implement this solution can be downloaded at **HTTP://WWW.SASCOMMUNITY.ORG/WIKI/FILE:PAPER_065-2012.ZIP .**

In order to accomplish this we need to understand a few basic concepts which are described next.

HOW DOES JAVA INTERACT WITH EXCEL?

The Apache Software Foundation is a community of Java developers and users that provide support for various open-source projects. One such open-source project is the Apache POI Project. This project is a Java application programming interface (API) which allows Java to both, create new Excel workbooks as well as manipulate pre-defined Excel report templates. The Apache POI API allows a Java to perform a number of Excel tasks such as opening an existing excel workbook, writing data in specific worksheet cells, creating a new worksheet, cell formatting etc. The java archive file (JAR) for manipulating excel is available as a free download at http://poi.apache.org/download.html . Below are java code snippets for manipulating excel.

Fig 5.

```
//Initialize reference to a new excel report template

InputStream reportTemplate  = null;

//If this is a new workbook (user input) then create a new excel
workbook otherwise open the existing excel workbook from the path
provided as input.

if (this.newExistingExcel.equalsIgnoreCase("NEW")){

   wb = new HSSFWorkbook();

   }

else {

      reportTemplate  = new FileInputStream(excelTemplatePath);

      this.wb         = new HSSFWorkbook(reportTemplate);

    }
```

Open an existing Excel workbook if it exists otherwise create a new excel workbook.

Fig 6.

```
//Check whether the worksheet specified in the input exists.If it
does not exist then create it.
this.worksheet     = wb.getSheet(excelWorksheetName);
if (worksheet == null){
    worksheet      = wb.createSheet(excelWorksheetName);
   }
```

Select a particular worksheet. Create the worksheet if it does not already exist.

Fig 7.

```
HSSFRow row   = worksheet.createRow(0);
HSSFCell cell = row.createCell(0);
cell.setCellValue(10);
```

Set the value of cell A1 to 10. Note that rows and columns are referenced by a numeric value starting at 0. e.g. Column A1 = 0,0; B2=1,1; C1=2,0 etc.

HOW CAN JAVA READ SAS DATA BEFORE SENDING IT TO EXCEL?

Java uses SAS integration technologies to read SAS datasets. The main tasks include connecting to the SAS server, querying SAS datasets and writing the results of the query to Java. Below are code samples to accomplish these tasks.

Fig 8.

```
//Connect to a remote SAS server using the parameters specified by the user
//Create a connection factory configuration
this.classID      = com.sas.services.connection.Server.CLSID_SAS;

//this.sasServer = SAS server name. e.g. sasserver.companyname.com
this.server       = new BridgeServer(this.classID, this.sasServer, 8591);

this.cxfConfig    = new ManualConnectionFactoryConfiguration(this.server);
this.cxfManager   = new ConnectionFactoryManager();
this.cxf          = this.cxfManager.getFactory(this.cxfConfig);

//this.sasId  = SAS User Id
//this.sasPwd = SAS Password
this.cx           = this.cxf.getConnection(this.sasId, this.sasPwd);
this.obj          = this.cx.getObject();
this.sasWorkspace = IWorkspaceHelper.narrow(this.obj);
```

Connect to the SAS server.

Fig 9.

```
//Simple select query on the SAS dataset
 String query  = "select * from '" + sasLib + "/" + sasDsname + ".sas7bdat'";

//Get a java database connection (JDBC) using the SAS connection that was
setup earlier (see figure 7)
this.jdbcConnection    = new MVAConnection(sasWorkspace, jdbcProps);
this.statement         = jdbcConnection.createStatement();
this.rs                = statement.executeQuery(query);
this.rsmeta            = this.rs.getMetaData();
```

Create a Java database connection (JDBC) and query SAS data.

PUTTING IT ALL TOGETHER

We shall now develop a sample Java code which combines the code snippets for reading SAS data and manipulating excel to create a complete solution to transferring SAS data to a pre-defined excel template. We make the design flexible so that you can use the Java code as is in your specific production environment. We'll do this by coding all site specific attributes such as SAS server name, SAS credentials, Excel report template path, SAS datasets to read and Excel worksheet cells to populate as input parameters to the SAS code. The generic Java code can then be used in any production environment without any customization. The main Java class file which can read a SAS dataset and transfer its contents to an excel workbook is given in the appendix. I have added comments in green color to explain each step. The input parameters are specified in comments at the top of the class.

IMPLEMENTATION

In this last section we'll look at the steps required to implement this solution in your specific production environment. We'll look at how to compile the java code and an example of how to call the Java "bridge" from SAS to produce the Excel report.

In order to compile this Java code you will need to perform the below steps:

1. Locate the Java installation directory on your unix server. Contact your server administrator if you are not able to find the Java installation.
2. Locate your .profile file on the server and add the below. The path highlighted in blue will need to be modified to your specific Java and JAR file path. Two JAR files are required – sas.svc.connection.jar (to connect to SAS from Java) and poi-3.7-20101029.jar (for manipulating excel from Java). Remember that the POI JAR file can be downloaded from **HTTP://POI.APACHE.ORG/DOWNLOAD.HTML**

```
PATH=/usr/java15/bin:$PATH
export PATH

JAVA_HOME=/usr/java15
export JAVA_HOME

CLASSPATH=/user/jars/poi-3.2-FINAL-20081019.jar:/user/jars/sas.svc.connection.jar:.
export CLASSPATH
```

3. Save the Java class file to a folder on the unix server. E.g. /projects/java/
4. To compile the code login to your unix server and execute the below commands
   a. cd /projects/java
   b. javac sasDataToExcel.java

   You should see a new file called sasDataToExcel.class created in the same folder as the .java class file.

Now we shall look at how to run the Java code to transfer data from SAS to the excel workbook.

1. First save the report template to a folder on unix. For example /projects/excel/template

2. Below are the parameters that you'll need to provide as input to the Java code
   a. SAS server name
   b. SAS user id
   c. SAS password
   d. SAS library containing the dataset to be transferred to excel
   e. SAS dataset name (in the library specified in part d)
   f. Excel template path. This is the path where you saved your excel report template (e.g. /projects/excel/template). Specify NA if you want to write the SAS data to a new workbook.
   g. Excel report (output) path. The resulting report conating SAS data will be saved at this location.
   h. Excel worksheet name. The SAS data will be written to this worksheet. The worksheet will be created if it does not already exist. In the example below, the worksheet name is "Input".
   i. Starting row number. SAS data will be written starting at this row number. In the example below, we will be writing the SAS data at row number 0.
   j. Starting column number. SAS data will be written starting at this column number. In the example below, we will be writing the SAS data at column number 0.
   k. Write headers (Y/N). Y means that the column labels will be written as table headers.

3. The below SAS code snippet demonstrates how to use the Java program. The PROC SQL statement creates a dataset with report data to be transferred to the excel report template. The %sysexec statements executes unix system commands to run the Java code specifying the required input parameters. The final report will be saved to the output path specified (in this case it is `/projects/excel/report/Report.xls).`

```
/*Define output library which will contain the SAS dataset to be used for reporting*/
libname out '/projects/sas/reportingdata';run;

/*Create the SAS dataset to be used for reporting.
  The data from this dataset will be transferred to the excel report template
*/
proc sql;
create table out.orsales as
select year                  label='Year',
       sum(profit) as profit label='Profit'
from sashelp.orsales
group by year;
Quit;

/*Change directory to the path containing the Java class file*/
%sysexec %str(cd /projects/java);

/*Execute the Java program specifying all input parameters*/
%sysexec %str(java sasDataToExcel
                  servername
                  userid
                  password
                  /projects/sas/reportingdata
                  orsales
                  /projects/excel/template/ReportTemplate.xls
                  /projects/excel/report/Report.xls
                  Input
                  0
                  0
                  Y
              );
```

CONCLUSION

We have seen how to use SAS integration technologies to create fully formatted Excel reports using pre-formatted Excel report templates. This solution makes it very easy to create highly customized Excel reports consisting of charts, tables and other formatting because we are able to build the charts and apply formats through Excel prior to runtime and only send SAS data to the excel workbook at runtime.

REFERENCES

Excellent Ways of Exporting SAS Data to Excel, Ralph Winters.
http://www.nesug.org/proceedings/nesug04/io/io09.pdf

Apache Excel POI developer guide.
http://poi.apache.org/spreadsheet/quick-guide.html

SAS Online Support

ACKNOWLEDGMENTS

Thanks to the SAS online support for prompt responses to our questions.

DISCLAIMER

All code provided in this paper is provided on an "AS IS" basis, without warranty. Neither the author nor CIGNA nor The Hartford make any representation, or warranty, either express or implied, with respect to the programs, their quality, accuracy, or fitness for a specific purpose. Therefore, neither the author nor CIGNA nor The Hartford shall have any liability to you or any other person or entity with respect to any liability, loss or damage caused or alleged to have been caused directly or indirectly by the programs provided in this paper. This includes, but is not limited to, interruption of service, loss of data, loss of profits, or consequential damages from the use of these programs.

CONTACT INFORMATION

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

George Mendoza
CIGNA Healthcare
900 Cottage Grove Road
Bloomfield, CT 06152
Work Phone: 860-226-1960
Email: mendoza_george@yahoo.com

Subhashree Singh
The Hartford
One Hartford Plaza
Hartford, CT 06155
Work Phone: 860-547-2398
Email: subhashree_s123@yahoo.com

APPENDIX

```
/*
* Created On  : Dec 27, 2011.
*
* Description : This program transfers SAS data to a predefined/new
*               Excel workbook.
*
* Author      : George G Mendoza and Subhashree Singh
*
* Inputs      : arg[0]  - SAS Server
*               arg[1]  - SAS Id
*               arg[2]  - SAS Pwd
*               arg[3]  - SAS Lib
*               arg[4]  - SAS dataset name
*               arg[5]  - Excel Template path. Use NA for new workbook.
*               arg[6]  - Excel Report (Output) Path
*               arg[7]  - Excel Worksheet
*               arg[8]  - Starting Row Number
*               arg[9]  - starting Column Number
*               arg[10] - Write Headers ? (Y/N)
*/

/*Java IO Related Packages**/
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

/*Java SQL Packages***/
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

/*Apache POI Excel Packages**/
import org.apache.poi.hssf.model.Workbook;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFCellStyle;
import org.apache.poi.hssf.usermodel.HSSFFont;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.hssf.util.HSSFColor;

/*SAS Packages for Java Development**/
import com.sas.iom.SAS.IWorkspace;
import com.sas.iom.SAS.IWorkspaceHelper;
import com.sas.rio.MVAConnection;
import com.sas.services.connection.BridgeServer;
import com.sas.services.connection.ConnectionFactoryConfiguration;
import com.sas.services.connection.ConnectionFactoryException;
import com.sas.services.connection.ConnectionFactoryInterface;
import com.sas.services.connection.ConnectionFactoryManager;
import com.sas.services.connection.ConnectionInterface;
import com.sas.services.connection.ManualConnectionFactoryConfiguration;
import com.sas.services.connection.Server;

                                                    //Continued on next page…
```

```
public class sasDataToExcel {

        //SAS connection variables
        IWorkspace sasWorkspace                  = null;
        ConnectionInterface cx                   = null;
        String classID                           = null;
        Server server                            = null;
        ConnectionFactoryConfiguration cxfConfig = null;
        ConnectionFactoryManager cxfManager      = null;
        ConnectionFactoryInterface cxf           = null;
        org.omg.CORBA.Object obj                 = null;

        //SQL attributes
        MVAConnection jdbcConnection = null;
        Properties jdbcProps = new Properties();
        Connection connection                    = null;
        Statement statement                      = null;
        ResultSet rs                             = null;
        ResultSetMetaData rsmeta                 = null;

        //Excel attributes
        HSSFWorkbook wb                          = null;
        HSSFSheet worksheet                      = null;

        //Input parameters
        String sasServer                         = "";
        String sasId                             = "";
        String sasPwd                            = "";
        String sasLib                            = "";
        String sasDsname                         = "";
        String excelTemplatePath                 = "";
        String excelOutputReportPath             = "";
        String newExistingExcel                  = "";
        String excelWorksheetName                = "";
        String startRowNum                       = "";
        String startColumnNum                    = "";
        String writeColumnHeaders                = "";

                                                        //Continued on next page…
```

```
private void connectToSAS() throws ConnectionFactoryException{
//Connect to a remote SAS server using the parameters specified by the user
System.out.println("In connect To SAS");
this.classID                            = com.sas.services.connection.Server.CLSID_SAS;
this.server                             = new BridgeServer(this.classID, this.sasServer, 8591);
this.cxfConfig                          = new ManualConnectionFactoryConfiguration(this.server);
this.cxfManager                         = new ConnectionFactoryManager();
this.cxf                                = this.cxfManager.getFactory(this.cxfConfig);
this.cx                                 = this.cxf.getConnection(this.sasId, this.sasPwd);
this.obj                                = this.cx.getObject();
this.sasWorkspace                       = IWorkspaceHelper.narrow(this.obj);
}

private void readSASData() throws SQLException{
//Read SAS dataset specified by the user
System.out.println("In read SAS Data");
//Simple select query on the SAS dataset
String query           = "select * from '" + sasLib + "/" + sasDsname + ".sas7bdat'";
System.out.println(query);
//Get a java database connection (JDBC) using the SAS connection that was setup earlier (see
connectToSAS())
this.jdbcConnection    = new MVAConnection(sasWorkspace, jdbcProps);
this.statement         = jdbcConnection.createStatement();
this.rs                = statement.executeQuery(query);
this.rsmeta            = this.rs.getMetaData();
}

private void createExcelWorkbook() throws IOException{
//Open or create excel workbook
System.out.println("In createExcelWorkbook");
//Initialize reference to a new excel report template
InputStream reportTemplate  = null;
//If this is a new workbook (user input) then create a new excel workbook
//otherwise open the existing excel workbook from the path provided as input.
if (this.newExistingExcel.equalsIgnoreCase("NEW")){
    wb = new HSSFWorkbook();
}
else {
     reportTemplate          = new FileInputStream(excelTemplatePath);
     this.wb                 = new HSSFWorkbook(reportTemplate);
}
//Check whether the worksheet specified in the input exists.
//If it does not exist then create it.
this.worksheet              = wb.getSheet(excelWorksheetName);
if (worksheet == null){
    worksheet               = wb.createSheet(excelWorksheetName);
}
}
```

//Continued on next page…

```
private void writeTableHeaders() throws SQLException{

//Write table headers. Table headers default to the label of the variable
System.out.println("In writeTableHeaders");

//Initialize variables
int colnum               = new Integer(this.startColumnNum).intValue();
int rownum               = new Integer(this.startRowNum).intValue();
int colLabelCnt             =0;
HSSFRow   rowHeader      = null;
HSSFCell  cellHeader     = null;
HSSFCellStyle styleTitle = null;
HSSFFont font            = null;

//Create excel cell styles for formating excel
styleTitle = wb.createCellStyle();
font = wb.createFont();
styleTitle.setFont(font);
font.setFontHeightInPoints((short) 10);
font.setFontName("Arial");
font.setColor(HSSFColor.WHITE.index);
font.setBoldweight(HSSFFont.BOLDWEIGHT_BOLD);
styleTitle.setFont(font);
styleTitle.setAlignment(HSSFCellStyle.ALIGN_CENTER);
styleTitle.setBorderBottom(HSSFCellStyle.BORDER_THIN);
styleTitle.setBorderTop(HSSFCellStyle.BORDER_THIN);
styleTitle.setBorderLeft(HSSFCellStyle.BORDER_THIN);
styleTitle.setBorderRight(HSSFCellStyle.BORDER_THIN);
styleTitle.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
styleTitle.setFillForegroundColor(HSSFColor.BLUE_GREY.index);

//For existing excel report templates don't create new rows to add data because you will loose the
//preformatting.
if(newExistingExcel.equalsIgnoreCase("NEW")){
  rowHeader  = worksheet.createRow((short) rownum);
}
else {
      rowHeader  = worksheet.getRow((short) rownum);
      if(rowHeader==null){
         rowHeader  = worksheet.createRow((short) rownum);
      }
     }

//Write out label names of all variables
while(colLabelCnt < this.rsmeta.getColumnCount()) {
  if(newExistingExcel.equalsIgnoreCase("NEW")){
     cellHeader = rowHeader.createCell((short) colnum);
     }
  else {
     cellHeader = rowHeader.getCell((short) colnum);
     if(cellHeader==null){
       cellHeader = rowHeader.createCell((short) colnum);
     }
    }
  cellHeader.setCellValue(rsmeta.getColumnLabel(colLabelCnt+1));
  cellHeader.setCellStyle(styleTitle);
  colLabelCnt +=1;
  colnum +=1;
 }
}
```

//Continued on next page…

```
private void writeSASDataToExcel() throws NumberFormatException, SQLException, IOException{

 //Write SAS data to excel
  System.out.println("In writeSASDataToExcel");
  //Initialize variables
  int colnum               = new Integer(this.startColumnNum).intValue();
  int rownum               = new Integer(this.startRowNum).intValue();
  int colData              = 0;
  HSSFRow row              = null;
  HSSFCell cell            = null;
  HSSFCellStyle styleData  = null;
  HSSFFont fontdata        = null;
  //Create a cell style
  styleData = wb.createCellStyle();
  fontdata = wb.createFont();
  styleData.setFont(fontdata);
  fontdata.setFontHeightInPoints((short) 10);
  fontdata.setFontName("Arial");
  styleData.setFont(fontdata);
  styleData.setAlignment(HSSFCellStyle.ALIGN_CENTER);
  styleData.setBorderBottom(HSSFCellStyle.BORDER_THIN);
  styleData.setBorderTop(HSSFCellStyle.BORDER_THIN);
  styleData.setBorderLeft(HSSFCellStyle.BORDER_THIN);
  styleData.setBorderRight(HSSFCellStyle.BORDER_THIN);
  styleData.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
  rownum+=1;
  //Write SAS data records to the excel worksheet
  while (rs.next()){
  //Access or create a row
  if(newExistingExcel.equalsIgnoreCase("NEW")){
     row = worksheet.createRow(rownum);
  }
  else {
    row = worksheet.getRow(rownum);
    if(row==null){
      row = worksheet.createRow(rownum);
    }
  }
  //Access or create a cell
  colnum  = new Integer(this.startColumnNum).intValue();
  colData = 0;
  while(colData < rsmeta.getColumnCount()){
      if(newExistingExcel.equalsIgnoreCase("NEW")){
         cell = row.createCell((short) colnum);
      }
      else {
        cell = row.getCell((short) colnum);
        if(cell==null){
          cell = row.createCell((short) colnum);
      }
     }
 //Set cell value using data from the SAS dataset based on the datatype
 if (rsmeta.getColumnType(colData+1)==12) {
    cell.setCellValue(rs.getString(colData+1));
 }
 else {
   cell.setCellValue(rs.getDouble(colData+1));
 }

 colData +=1;
 colnum +=1;
 }
 rownum+=1;
 }
 //Turn off the excel gridline display
 worksheet.setDisplayGridlines(false);
 //Reevaluate all formulas and chart data links whenever excel is opened
 for(int sheetNum = 0; sheetNum < wb.getNumberOfSheets(); sheetNum++) {
    this.worksheet = wb.getSheetAt(sheetNum);
    this.worksheet.setForceFormulaRecalculation(true);
          }
 //Create a permanent copy of the report
 writeToExcelWorkbook();
}}}
                                                          //Continued on next page…
```

```
private void writeToExcelWorkbook() throws IOException{
//Save the report to the path specified by the user
System.out.println("Writing physical file");
FileOutputStream out = new FileOutputStream(excelOutputReportPath);
wb.write(out);
out.close();
}

private void closeSASConnection() throws SQLException{
//Close the JDBC and SAS connection
jdbcConnection.close();
cx.close();
}

public static void main(String[] args) {
//This program writes the contents of a SAS dataset to a specified worksheet cell of a
//preformatted excel report
System.out.println("Begin SAS To Excel");
//Instantiate object
sasDataToExcel sas2excel = new sasDataToExcel();
try{
 //Retrieve input parameters
 sas2excel.sasServer             = args[0];
 sas2excel.sasId                 = args[1];
 sas2excel.sasPwd                = args[2];
 sas2excel.sasLib                = args[3];
 sas2excel.sasDsname             = args[4];
 sas2excel.excelTemplatePath     = args[5];
 sas2excel.excelOutputReportPath = args[6];
 sas2excel.excelWorksheetName    = args[7];
 sas2excel.startRowNum           = args[8];
 sas2excel.startColumnNum        = args[9];
 sas2excel.writeColumnHeaders    = args[10];
 if(sas2excel.excelTemplatePath.trim().equalsIgnoreCase("NA")) {
   sas2excel.newExistingExcel = "NEW";
 }
 else {
  sas2excel.newExistingExcel = "EXISTING";
 }
 //Connect To SAS
 System.out.println("Connecting to SAS...");
 sas2excel.connectToSAS();
 //Read SAS Dataset
 System.out.println("Reading SAS Data...");
 sas2excel.readSASData();
 //Open/Create Excel Workbook
 System.out.println("Creating Excel Workbook...");
 sas2excel.createExcelWorkbook();
 //Write table headers
 System.out.println("Writing Table Headers...");
 if(sas2excel.writeColumnHeaders.equalsIgnoreCase("Y")){
    sas2excel.writeTableHeaders();
 }
 //Write SAS data to excel
 System.out.println("Writing SAS datarecords...");
 sas2excel.writeSASDataToExcel();
 }catch(Exception e){System.out.println("Error - " + e.getMessage());}
 finally{
  //Close SAS Connection
  System.out.println("Closing SAS Connection...");
  try {
        sas2excel.closeSASConnection();
  } catch (SQLException e1) {e1.printStackTrace();}
}}}

                                                                    //End of Java code.
```

****