Paper 010-2011

Building a Self-Service SAS® Reporting Application - Soup to Nuts

George Mendoza, CIGNA Healthcare, Bloomfield, CT

ABSTRACT

SAS is widely used in a number of industries for producing different types of reports including financial reports, participation reports, outcomes reports, sales reports etc. Very often high volumes of reports are required with a very short turnaround time. Self-service is a key to success in such an environment. Self-service reports are generally of higher quality, less labor-intensive and instantaneous. They also have the potential to reach wider audiences over time. This paper will describe the process of building a self-service SAS reporting application using Base SAS® together with SAS® Integration technologies. The entire process will be described from start to finish with several code samples along the way. So sit back and enjoy the ride!

INTRODUCTION

SAS Integration Technologies allows web technologies such as Java and .NET to interact with the SAS system. Java and .NET can execute SAS programs as well as access SAS data and ODS output. One needs to know just a little bit of Java or .NET to share SAS reports as a self-service application. This paper will show you that it is surprisingly easy to build a self-service SAS reporting application using these technologies. We'll begin with a diagrammatic overview which will paint a picture of the major components of the application. We'll then review code snippets to develop a clear understanding of the SAS and web technology code that is required for such an application. Attached with this paper is a sample SAS self-service reporting application. The final section will be a step-by-step walkthrough of the implementation of the sample self-service reporting application on your own machine, which you can use as a starting point for sharing your own reports via a self-service application.

THE BIG PICTURE

The example provided in this paper will use Java as the web technology for the user interface and SAS programs on the server to produce the reports. The Java front end application will interact with the SAS server using SAS Integration Technologies. Figure A below illustrates the high level design of the self-service reporting application.

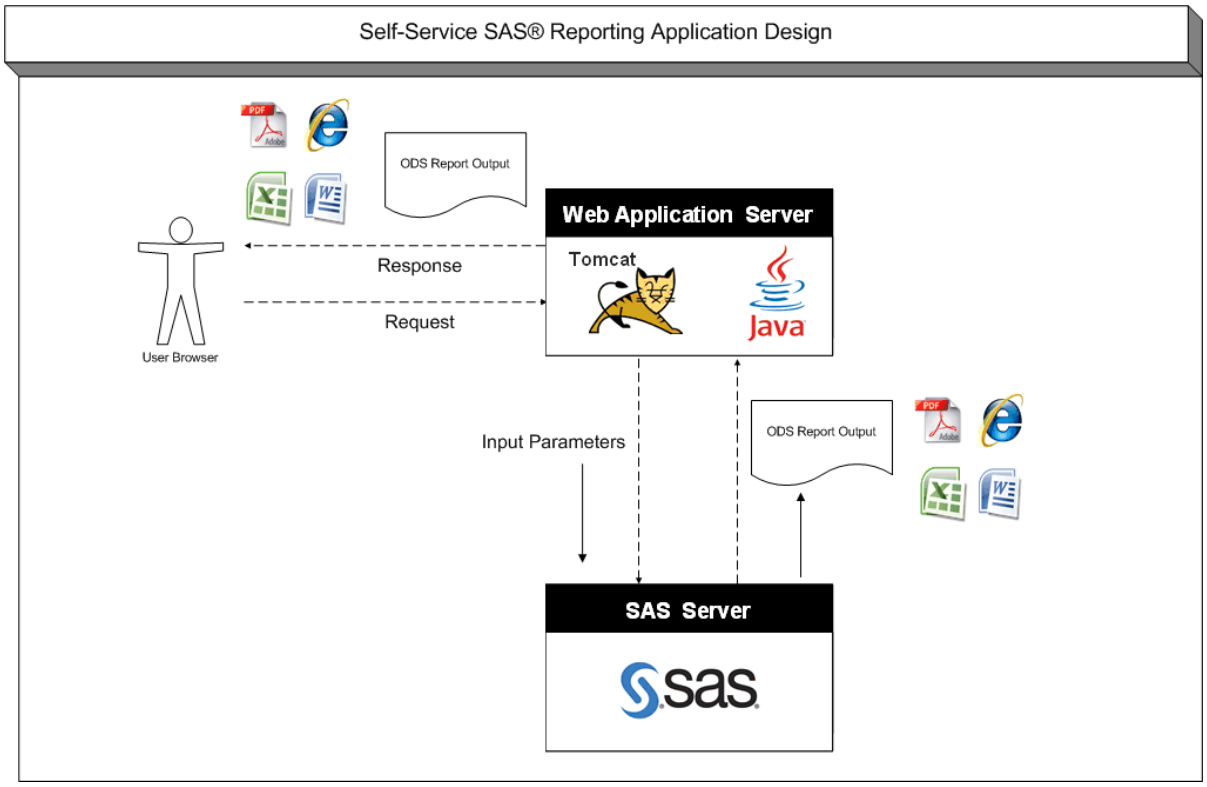


Figure A: Application Design

REQUIRED SOFTWARE

Software	Version	Operating System	Alternate Software
Base SAS	9.1 or higher	Windows/Unix	None
SAS Integration Technologies	9.1 or higher	Windows/Unix	None
Apache Tomcat (Web Application server)	4.1 or higher	Windows/Unix	Glassfish, Websphere
Eclipse (Java IDE)	3.4 and above	Windows	NetBeans, Rational Application Developer.

THE SAS CODE

Below is a simple SAS program which produces a tabular report in the user-specified ODS format (pdf/rtf/html/excel). There are two global macro variables defined. The “gender” macro variable enables filtering of the data by gender and the “reportFormat” macro variable will determine the ODS report format. The filename and ods statements establish a reference to the ODS output file. Java will use this reference to retrieve the ODS output file from the SAS system. Note that the Java front end will always pass valid parameters to the SAS code so error handling is not required in the SAS code.

```
*Define the global macro variables that will be passed from the Java app;
%global gender;
%global reportFormat;

*ProcessBody;
filename out&reportFormat 'report';

*Define ods output format;
ods &reportFormat file=out&reportFormat;
options nodate;

*Sample Report Code;
%macro report;

proc tabulate data= sashelp.class;
  title 'SAS Demo Report for Gender=' "&gender";
  class sex age;
  var height;
  table age all, sex*(height*mean);

  %if %length(&gender) > 0 and "&gender" ne "ALL" %then %do;
      where sex = "&gender";
  %end;

run;

%mend;

%report;

ods _all_ close;
ods listing;
```

Code Box 1: Sample SAS Code

OVERVIEW OF THE JAVA COMPONENTS

Below is a list of the main Java components together with a brief description of each of them.

1. Java IDE – The Java IDE or integrated development environment provides Java developers with a source code editor, compiler, build automation tools and a debugger. All these components help simplify Java code development. For my example I used Eclipse as the IDE which is available as a free download at http://www.eclipse.org/ .
2. Java Web Application Server – A web application server is a server side program that handles all communications between the server and the client/user. Its main function is to accept a client request, execute code based on the client request and send the results back to the client in the form of a response. Luckily, the developer does not have to code this program because there are many web application servers available for use. Tomcat, Websphere and GlassFish are some popular web application servers available. This example uses Tomcat which is available as a free download at http://tomcat.apache.org/. Choose Tomcat Version 6.0 and download the 32-bit/64-bit Windows Service Installer. Note down the port number and admin userid and password that you specify during installation.
3. Java Server Page (JSP) – This is the code which will generate the user interface HTML page for the user to enter in the report parameters and run the report. We can think of the JSP as a front-end to the reporting application. This component will need to be coded by the developer. It is called “default.jsp” in this example.
4. Servlet – The servlet is a Java class that is invoked by the JSP (front end) to execute SAS code, retrieve SAS ODS output and write the ODS output to the browser for the user to access. This component will need to be coded by the developer. In the example, I have called this component “getReport.java”.

THE MAIN JAVA SERVLET CLASS

The below servlet code (Figure C) instantiates a SAS session through the SAS object spawner, passes parameters and executes the SAS code. It then reads the ODS output file and returns it to the user as an attachment in the browser. I have added comments in green to explain each section of the code.

```
public class getReport extends HttpServlet {
 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

  ConnectionInterface cx = null;
  try {
       ServletContext context  = getServletContext();
       //Retrieve Parameters from the web.xml file
       String host      = context.getInitParameter("sasServer");
       int port         = Integer.parseInt(context.getInitParameter("sasPort"));
       String user      = context.getInitParameter("sasUserName");
       String password  = context.getInitParameter("sasPassword");
       String codePath  = context.getInitParameter("sasProgramPath");
       String codeName  = context.getInitParameter("sasProgramName");

       //Read user input (from the web page)
       String reportFormat    = request.getParameter("reportFormat");
       String gender          = request.getParameter("gender");

       //Connect to the SAS Server with the parameter values retrieved from web.xml
       String classID                              = Server.CLSID_SAS;
       Server server                               = new BridgeServer(classID, host, port);
       ConnectionFactoryConfiguration cxfConfig    = new ManualConnectionFactoryConfiguration(server);
       ConnectionFactoryManager cxfManager         = new ConnectionFactoryManager();
       ConnectionFactoryInterface cxf              = cxfManager.getFactory(cxfConfig);
       cx                                          = cxf.getConnection(user, password);
       org.omg.CORBA.Object obj                    = cx.getObject();
       IWorkspace sasWorkspace                     = IWorkspaceHelper.narrow(obj);
```

Continued on Next Page…

```
   //Keep a reference to the SAS report output
       IFileService fileService                    = sasWorkspace.FileService();
       StringHolder outstring                      = new StringHolder();
       IFileref fileRef                            = fileService.AssignFileref
                                                  ("out"+reportFormat, "TEMP", "", "",outstring);
       //Submit SAS Code to produce the report
       ILanguageService languageService            = sasWorkspace.LanguageService();
       IStoredProcessService sasSP                 = languageService.StoredProcessService();

       sasSP.Repository(codePath);
       sasSP.Execute(codeName,"gender='" + gender + "' reportFormat='" +  reportFormat + "'");

      //Read report
      IBinaryStream fileStream                      = fileRef.OpenBinaryStream
                                                   (StreamOpenMode.StreamOpenModeForReading);
      //Prepare to send report to user
      ServletOutputStream stream                    = null;
      stream                                        = response.getOutputStream();
     //Prepare the response application types. i.e. PDF, RTF, HTML or EXCEL
      if (reportFormat.equalsIgnoreCase("EXCEL")){
          reportFormat = "xls";
        }
      response.setContentType("application/" + reportFormat.toLowerCase());
      response.addHeader("Content-Disposition", "attachment; filename=Report." +
                         reportFormat.toLowerCase());
    //Write report to the browser
    boolean readMore                                = true;
    OctetSeqHolder tempData                         = new OctetSeqHolder();
    while (readMore) {
            fileStream.Read(1024, tempData);
            if (tempData.value.length == 0) {
              readMore = false;
             } else {
              stream.write(tempData.value);
        }
     }
    //Close response
    fileStream.Close();
    if (stream != null)
        stream.close();
    }
   catch (Exception e) {
       e.printStackTrace();
   }
   finally {
       //Close SAS connection
       cx.close();
   }
 }
}
```

Code Box 2: Main Java Servlet Class

THE USER INTERFACE (JSP)

The user interface (default.jsp) mainly consists of HTML tags and CSS styles to make the application aesthetically appealing to the user. The below code snippet (Figure D) is the part of the JSP code which allows the user to make report selections and invoke the servlet which will run the SAS code and return the ODS output to the user. Notice that the action attribute of the form tag specifies the servlet to be executed when the submit button is clicked.

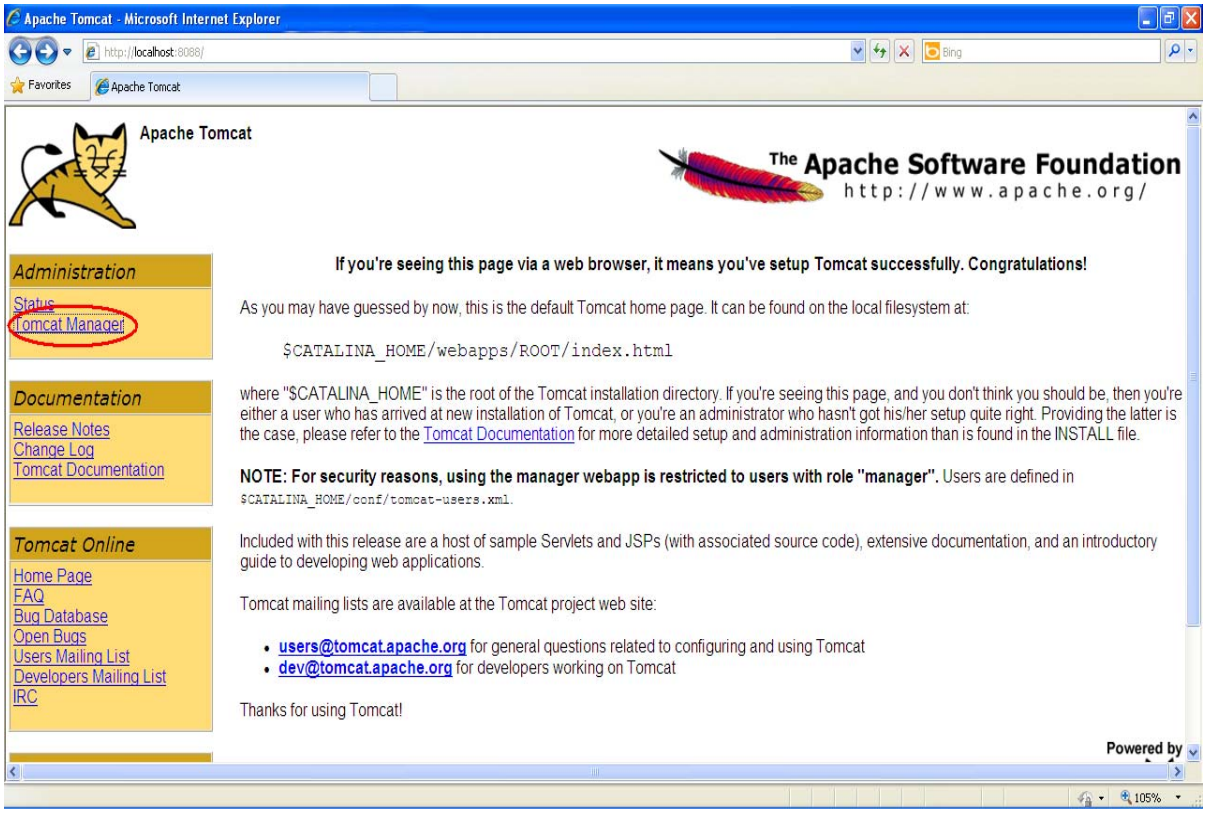
```
<form action="getReport" method="POST">
<table>
<tr><td>Gender : <select name="gender" class="select">
<option value="ALL" selected="selected">ALL</option>
       <option value="M">Male</option>
       <option value="F">Female</option>
       </select>
</td></tr>
<tr>
<td>Report Format : <select name="reportFormat" class="select">
       <option value="PDF" selected="selected">PDF</option>
       <option value="RTF">RTF</option>
       <option value="HTML">HTML</option>
       <option value="EXCEL">EXCEL</option>
       </select>
</td></tr>
<tr>
<td><INPUT type="submit" name="report" value="Get Report" class="submit" ></td>
</tr>
</table>
</form>
```

Code Box 3: JSP Code

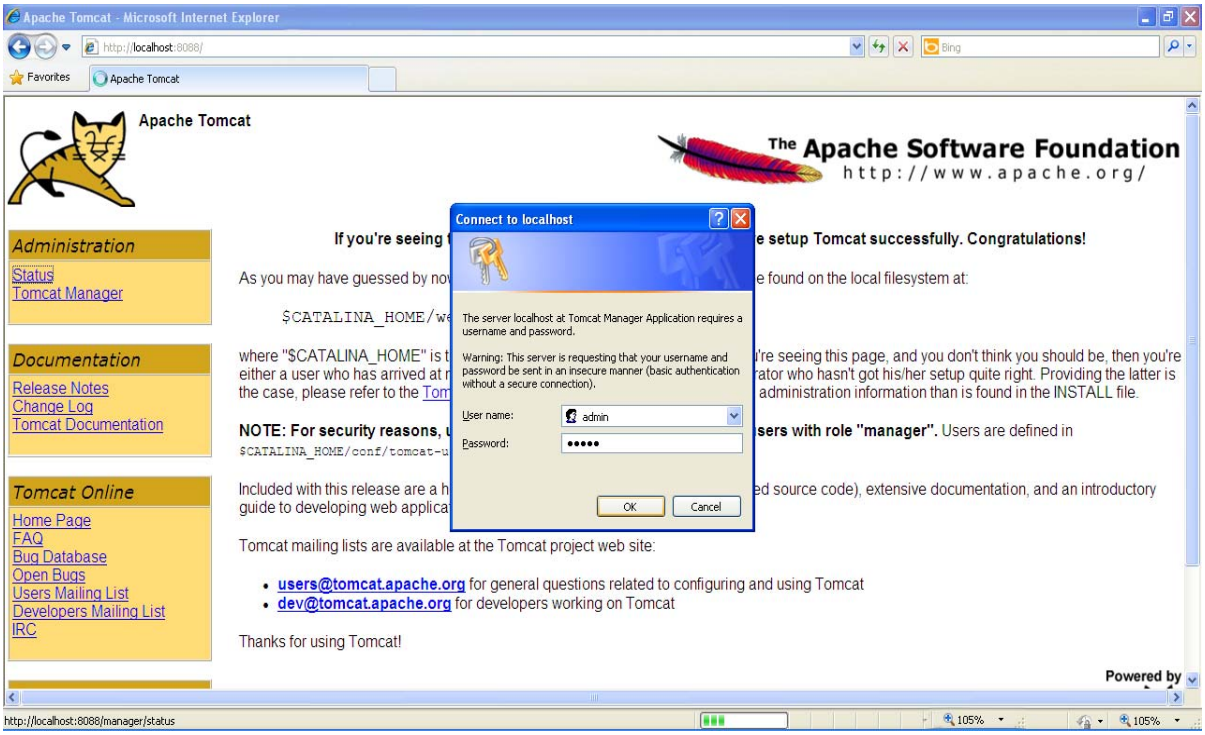
IMPLEMENTING THE SAMPLE APPLICATION ON YOUR MACHINE

Attached is the SAS and Java code described in this example. Below are the instructions to run the attached sample. We will use the Tomcat web application server for this walkthough.

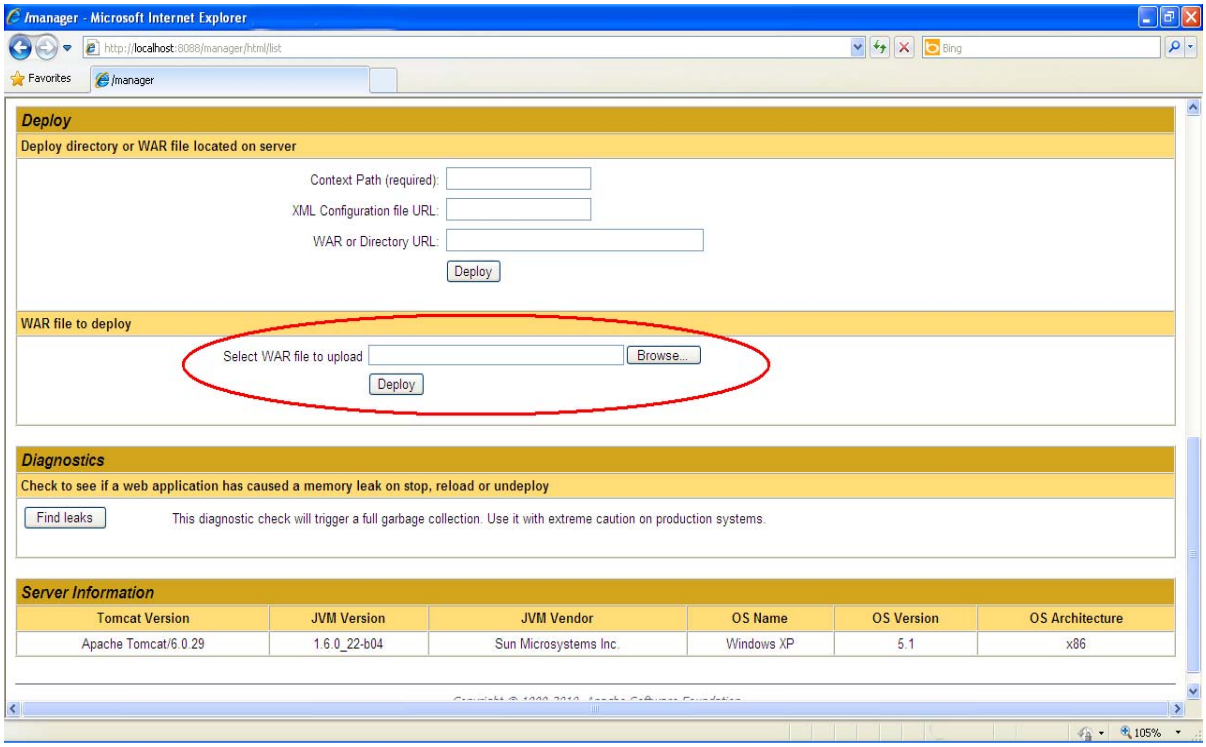
1. Unzip the sample code 0102011.zip attached with this paper to a drive on your machine. The sample code is also available for download at http://www.sascommunity.org/wiki/File:0102011.zip#filelinks .
2. Start the Tomcat server from Start>>All Programs>>Apache Tomcat 6.0>>Configure Tomcat. Click the Start button on the General tab of the window that opens up.
3. Deploy the Web Archive or WAR file (SASSelfServiceReporting.war) in Tomcat.
    a. If Tomcat was installed to run on port 8088 then access the Tomcat Manager at http://localhost:8088/



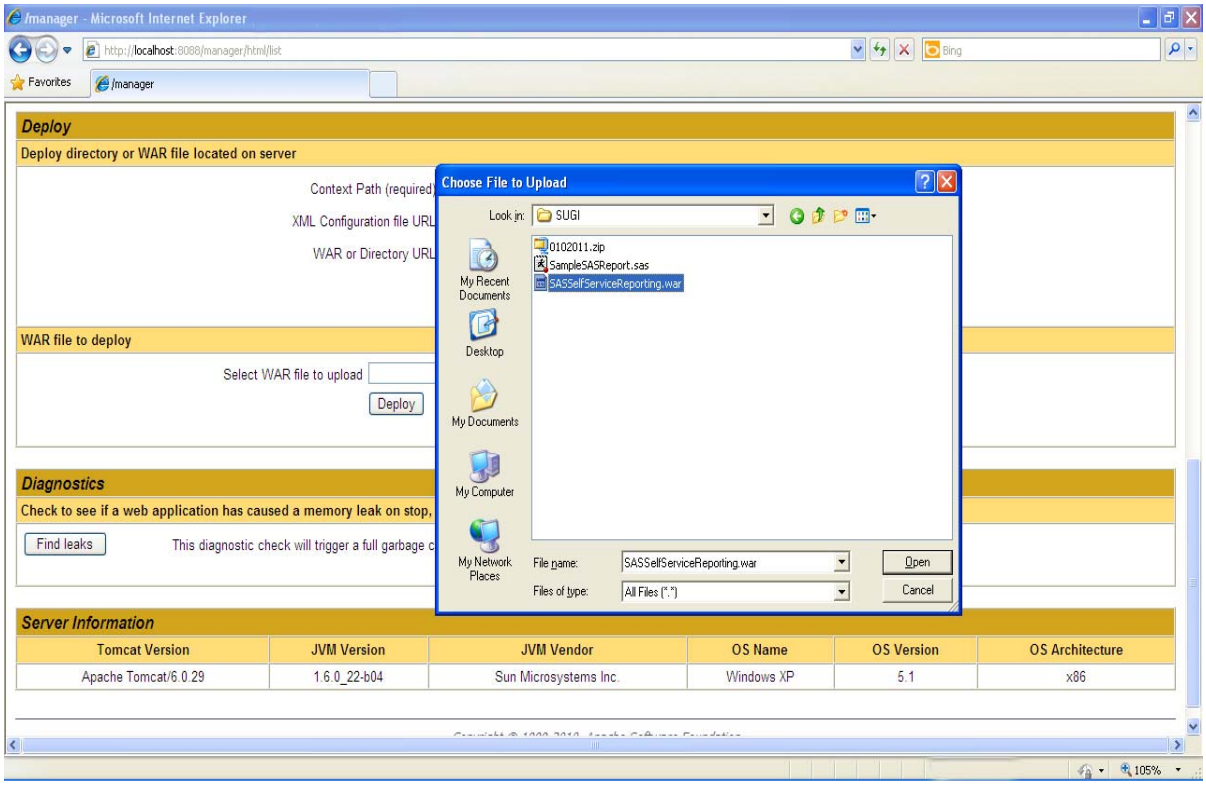
b. Login to the Tomcat Web Application Manager using the id and password you specified at the time you installed the server.

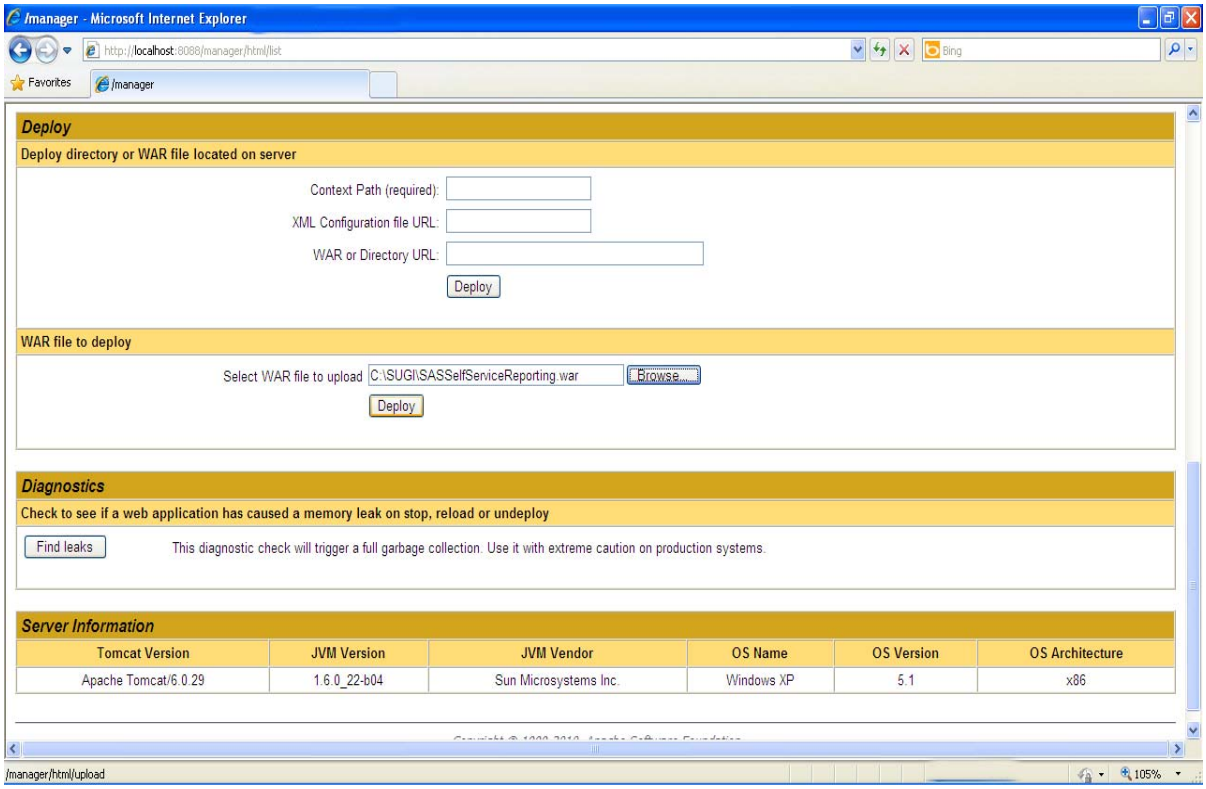


c. Scroll down to the section labeled “WAR file to deploy”.

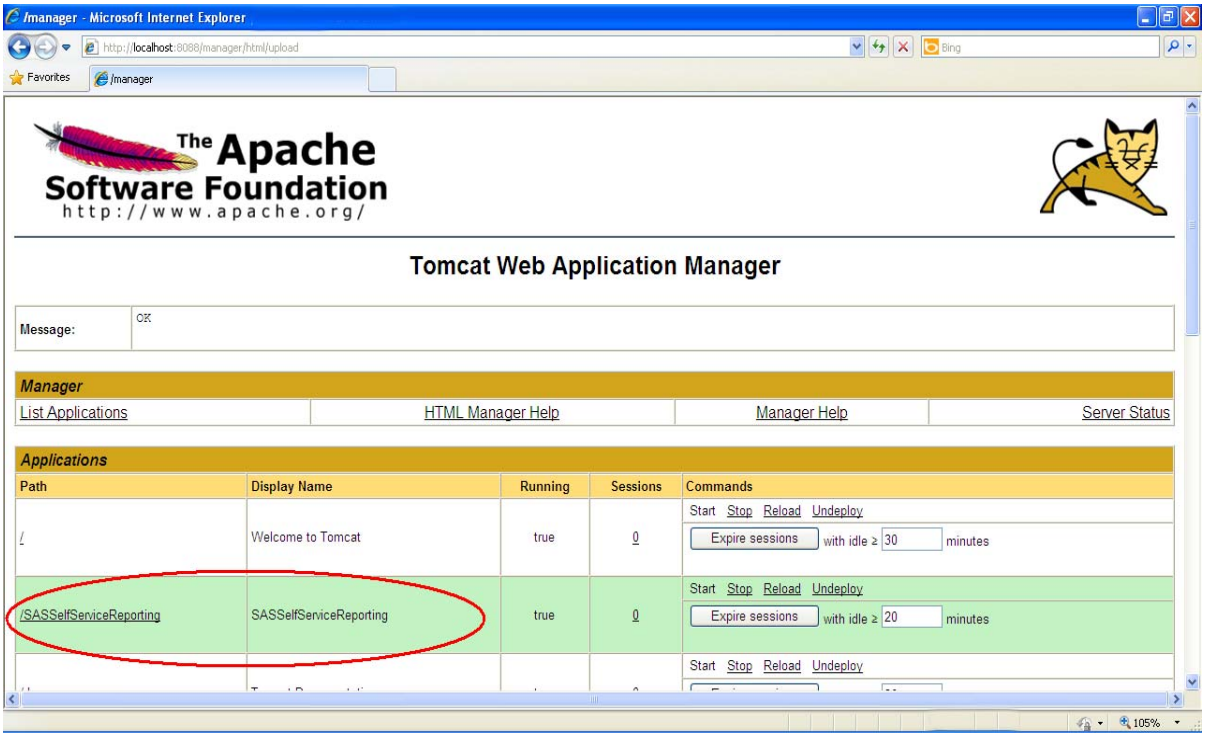


d. Browse to the folder containing the unzipped contents from step 1. Select the file named "SASSelfServiceReporting.war" and click "Deploy".



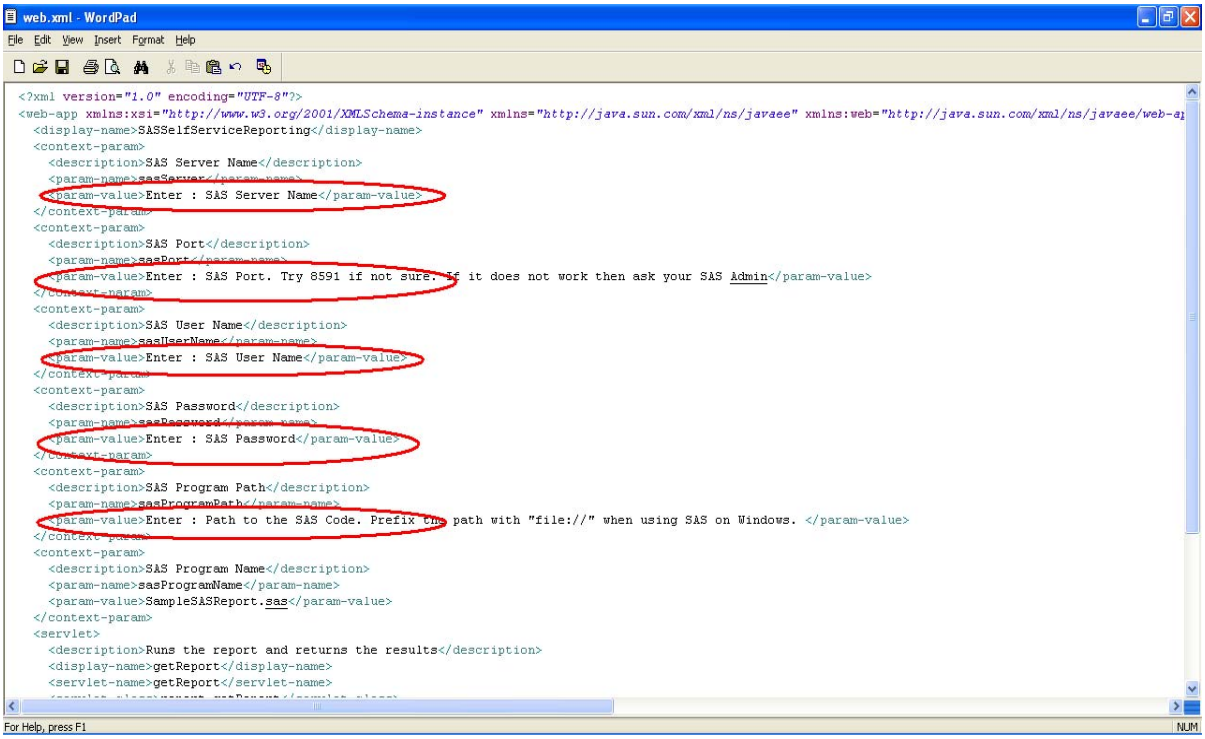


e. You will see the "SASSelfServiceReporting" application listed on the screen.

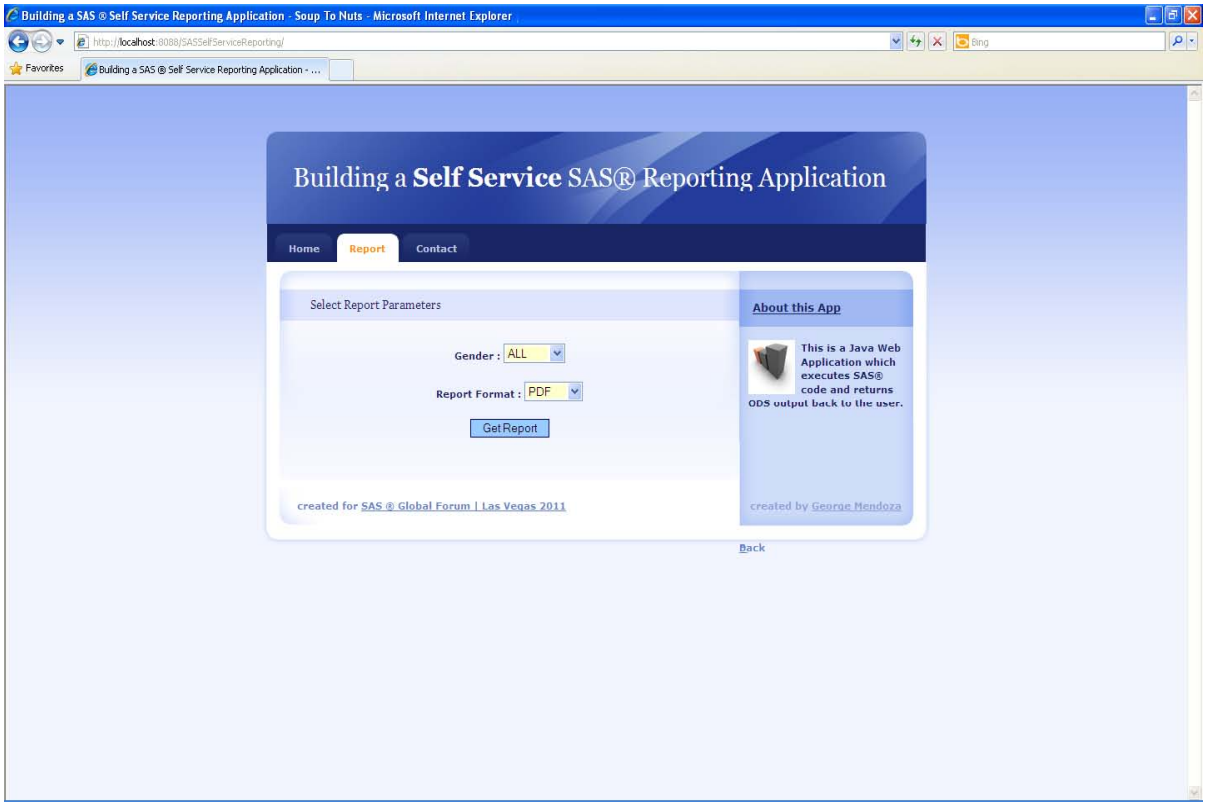


4. Open the Web.xml file of the application. You should find it at the path below if Tomcat was installed under C:\Program Files\. Update the Web.xml with the values for sasServer, sasPort, sasUserName, sasPassword and sasProgramPath. The user inputs are prefixed with “Enter : “

   Path to the web.xml file - C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\SASSelfServiceReporting\WEB-INF



5. Make sure the attached SAS code (SampleSASReport.sas) is saved at the correct path (reportSASProgramPath) on the server.
6. Click the "SASSelfServiceReporting" application link in the Tomcat Web Application Manager to access the application.



7. Click the "Get Report" button to run the report.

EXPANDING THE APPLICATION WITH NEW REPORTS

In order to add new reports to the self-service application you will need to.

1. Write the SAS code.
2. Update the Web.xml file with the new SAS report code name.
3. Add a new JSP page which will allow the user to make report selections and run the report.
4. Clone the servlet in the example (getReport.java) to create a new servlet which will accept the request from the JSP, create a SAS session, execute the SAS code and return the ODS output to the browser.

CONCLUSION

We have gone through the steps required to share SAS reports as a self-service application. Once the application is installed, the time and effort spent on processing requests will decrease drastically, and programming resources can be used for decision support work instead of manually running reports. The same framework can also be leveraged to share additional self-service reports. Hope you found the content of this paper useful. My contact details are provided at the end of this paper in case you have questions.

ACKNOWLEDGMENTS

A special thank you to Mike Rhoads and George Fernandez for their valuable tips and suggestions which added a lot of value to this paper. Thanks also to the SAS online support for prompt responses to my questions and to Nuvio Webdesign (http://www.nuvio.cz) for sharing a free web template at http://www.oswd.org/

RECOMMENDED READING

- George Mendoza 2011. “Self-Service and Automation Using SAS® with Java.” Cary, NC: SAS Press. (Anticipated publication date: Late 2011).
- SAS® Integration Technologies White Paper. Cary, NC: SAS Institute Inc. <http://www.sas.com/reg/wp/corp/3578>

DISCLAIMER

All code provided in this paper is provided on an "AS IS" basis, without warranty. Neither the author nor CIGNA 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 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 author at:

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

****