Paper 014-2007

Integration of webAF™ and Table-Driven Methodology

Shelly Yiu-Morrison, Bureau of Labor Statistics, Washington DC

ABSTRACT

This paper will provide a general overview of SAS® webAF, targeting novice audiences. It is intended to provide information that is gathered from different resources. It will try to explain at a very high level what exactly webAF is and how SAS and Java technologies can be combined to gain momentum in a world where web-based technologies are increasingly becoming more popular.

This paper will try to explain what webAF is. It will provide a basic explanation of the Java technology used in webAF to create server-side web applications such as Servlets and Java Server Pages (JSP). This paper will also look into how Servlets and JSPs work together, their roles, and their functions. Additionally, it will also look at the object-oriented perspective of the Model-View-Controller (MVC) design and architecture, and examine two types of connections to SAS.

Consequently, this paper will look at the integration of webAF and the Table-Driven Methodology. It will explain what the Table-Driven Methodology is and how tables are utilized to drive and generate code. This paper will also study several issues currently faced in our Table-Driven Process and examine how these issues can be solved by integrating web-based application using webAF. This integration modernizes our current Table-Driven Process to make it more user-friendly, efficient, and less error prone.

INTRODUCTION

In the Division of Consumer Expenditure Surveys at the Bureau of Labor Statistics (BLS), we process large amounts of data on what American consumers purchase. This data is collected on an annual and quarterly basis in the form of surveys. The data is then processed and the results are published for public-use. As times change and technology advances, consumer purchases will change accordingly. This change affects our surveys and form processing, as old items are phased out and new items are introduced.

WEBAF EXPLAINED

WebAF is an Integrated Development Environment (IDE) for developing web applications using server-side Java technology. WebAF incorporates Servlet and Java Server Page (JSP) technology with SAS AppDev Studio API components to create web-based applications that can connect to SAS and use SAS functionalities.

WEBAF SOFTWARE

WebAF is software provided by SAS that can be used to create and deploy client-based Java applications, such as applets and applications, as well as server-based Java applications, such as Servlets and JSPs. It provides a visual environment that will generate code for the developer simply by dragging and dropping components or models into the web application, very similar to Visual Basic. It supplies Customizer and Property menus to assign or modify parameter values and attributes of components and models without going into the code. Figure 1-1 shows the JDBC Connection Object's Customizer menu which can be used to create the JDBC connection, modify its parameters, or test the connection. Figure 1-2, shows the Property menu of the JDBC Connection Object, which provides another way to modify the values of its parameters.

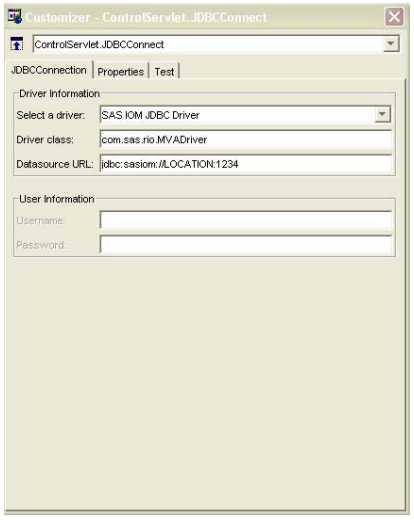


Figure 1-1. Customizer menu

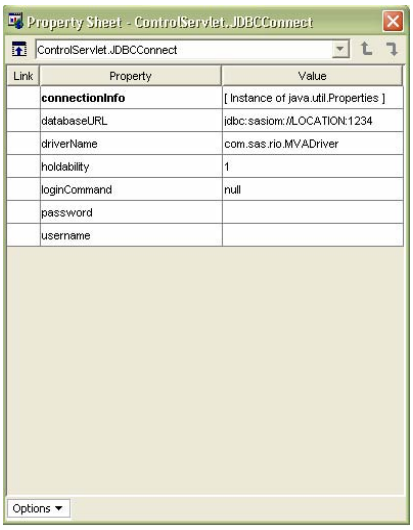


Figure 1-2. Property menu

The webAF environment also provides wizards and templates that generate code to perform common tasks, such as creating a blank template for a Servlet or JSP. Servlets and JSPs are essentially Java programs that can use pre-built components to access SAS data and procedures on a remote server from a thin client that does not have the SAS System installed.

JAVA

Java is a popular platform-independent, object-oriented programming language, meaning it can run on most platforms and allows solutions to be expressed in non-technical representations using Model-View separation. It can create applications that run on the web and generate dynamic web content on demand. Java can run on the client, the server, or both. Java is flexible in nature and therefore its popularity will continue to grow; hence, it is not surprising that SAS is integrating its technology with Java to make it even more prevalent.

SERVLET

Servlet is the Java platform technology of choice for extending and enhancing web servers. It is a popular choice for building interactive web applications. A Servlet is basically a Java class that runs on the server side within a servlet container. It has the ability to use any of Java's APIs, such as JDBC API to access enterprise databases and can also access a library of HTTP-specific calls. Servlet provides a component-based, efficient, platform-independent method for building web-based applications.

JAVA SERVER PAGE (JSP)

Java Server Page technology is an extension of the Servlet technology created to support authoring of HTML and XML pages. A JSP is compiled to a Servlet the first time it is called; hence, it recompiles automatically when the code is modified and eliminates the need to rebuild the page subsequently. This reduces development time for the JSP developer by allowing the developer to modify the appearance of the user-interface as often as needed, to attain the desired look.

MODEL-VIEW-CONTROLLER

The Model-View-Controller (MVC) is supported by both Servlets and JSPs. It creates a separation between the components that are used to view the data and those that are used to access the data. The model can be changed while leaving the view alone and vise versa. In addition, multiple viewers can use the same model.

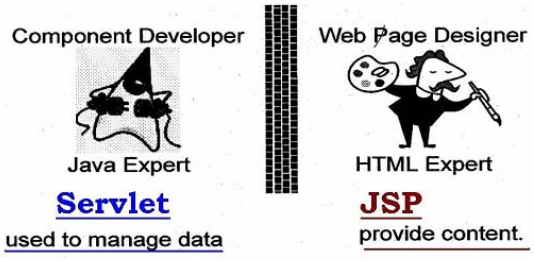
Figure 2-1. View

Figure 2-2. Model

One of the famous analogies of this concept is a clock. The view is the user interface, the outer appearance of the clock as shown in Figure 2-1, while the model is the mechanism inside the clock that keeps track of time, shown in Figure 2-2. The same mechanism or model can be reused inside different sizes, shapes, styles, colors, and types of clocks. By using MVC in Servlet/JSP, you can create a separation of application/business logic from web content delivery that encourages software reuse; thereby, simplifying deployment and maintenance.

JSP AND SERVLET

Together JSP and Servlet make the server-side components easier to create, modify, and maintain. They both have their distinct roles and functions within a web application (Figure 2-3).



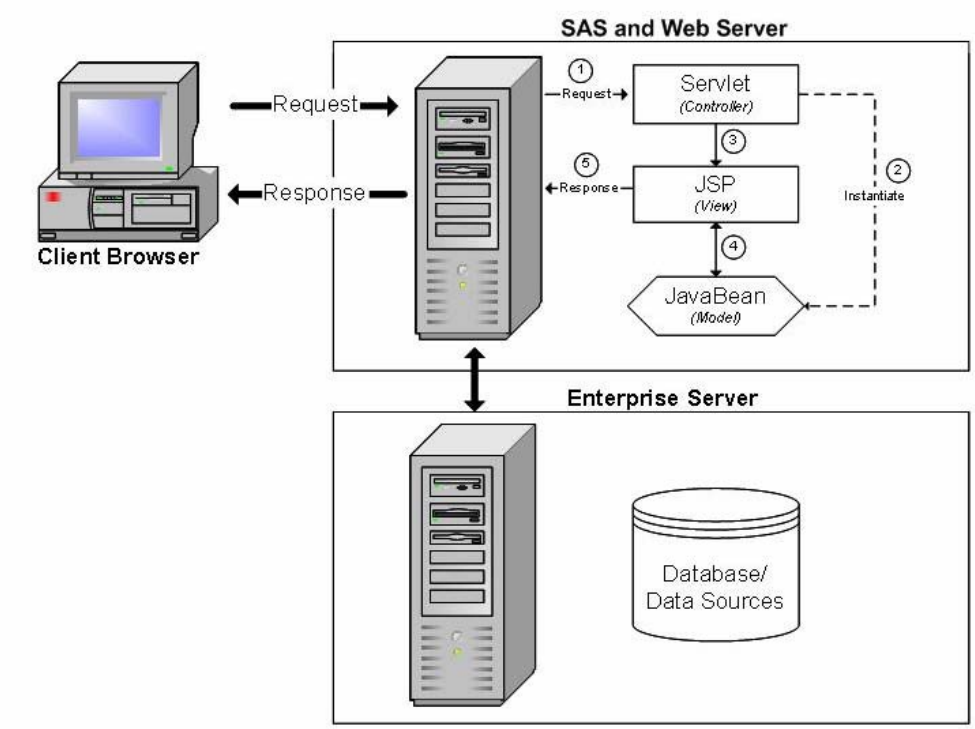
(Hillhouse 6/3/2006)

Figure 2-3. Servlet and JSP distinct roles and functions

The Servlet manages data, like a system designer, while the JSP controls the presentation of the look and feel of a web page, like an artist. The JSP relies on the Servlet to delegate or dispatch requests to another resource, while the Servlet relies on the JSP to present the result in an attractive format to the user. The JSP takes care of the view while the Servlet takes care of the model.

SERVLET/JSP AND MODEL-VIEW-CONTROLLER FITS TOGETHER

Figure 2-4 illustrates how Servlet and JSP fit into the MVC architecture within a web application setting.



(Girardin 7/25/2006)

Figure 2-4. Model-View-Controller Architecture

(1) The Client Browser, a user-web interface, sends a request to the Servlet.
(2) The Servlet performs all needed initialization and creates the model to be use by the View (JSP).
(3) The Servlet then forwards the request to the Java Server Page (JSP).
(4) The JSP may use JavaBeans which are reusable java class code, to further customize the view.
(5) Once the desired view or output is created, the JSP sends the response back to the requesting Client browser

ESTABLISH CONNECTION TO SAS

Depending on the needs of the application, there are two methods used to connect to SAS within a Java application. One method involves the Java Database Connectivity (JDBC) driver. This driver is used to access data from SAS and other sources. It makes it possible to establish a connection, send queries and update statements to the data source, and process the results using SQL. Another method used is the Integration Technologies, Integrated Object Model (IOM). This method is used when the strength of SAS is needed to manipulate the data prior to display and to perform some analytics or implement some special rules that SAS is needed for.

TABLE-DRIVEN METHODOLOGY APPLIED

Now equipped with some understanding of webAF and the technology involved, we can next look at how webAF can be used to improve the current Table-Driven Methodology used by the BLS's Division of Consumer Expenditure Surveys (DCES).

WHAT IS THE TABLE-DRIVEN METHODOLOGY?

The Table-Driven Methodology is the concept of utilizing external tables to store information and using these tables to drive and generate code. In Figure 3-1, the first row of the excel sheet table defines the variable name and the second row defines the length associated with each variable. Majority of the columns within the excel table list the parameters needed by an in-house macro.

SURVY	RELN	RTYP	SECTION	MACRO	DBNAME	TABLIST	VAR	KEY	OUTLIB	OUTFILE	DATA	MATCH
9	4	8	2	20	8	50	800	50	20	90	50	50

NOMISS	OUTMISS	FIRSTROW	LASTROW	PARMCARD	DROPPARM	STAMO	STAYR	ENDMO	ENDYR
1	20	8	8	200	200	2	4	2	4

Figure 3-1. DBExtract table in Excel, column heading

By utilizing a table to store the values for all the parameters needed by a macro, hard-coded information can be removed from the code. This makes the code easier to maintain since a table can be changed instead of the actual code. It allows end-users to be involved in the data-entry process and provides parameter input indirectly to the program without going through the developer/programmer. This process gives the end-users more control over what data is being used and streamlines the data updating process.

Once the information is stored in the table, code can be dynamically generated by creating macro variables at each column and feeding them into a macro function as parameters. Figure 3-2 shows one way of using the excel table after it is converted to a SAS data set. Using the SQL procedure, selected macro variables are created for all the columns of the row that match the desired condition in the WHERE clause. Then the macro variables are fed into an in-house macro, %Sybsas, as parameters.

```
proc  sqlnoprint;
    select dbname, tablist, var, key, outlib, outfile, data, match,
           nomiss, outmiss, firstrow, lastrow, parmcard, dropparm
    into  : dbname, : tablist, : var, : key, : outlib, : outfile, : data, : match,
          : nomiss, : outmiss, : firstrow, : lastrow, : parmcard, : dropparm
    from  extractdb
    where (rectype = "&rectype" and (startdate <= &qtr5 and
          (enddate >= &qtr5 or enddate = . )));
quit;

%sybsas (dbname = &dbname, tablist = &tablist, var = %trim(&var),
         key = &key, outlib = &outlib, outfile = &outfile,
        data = &data, match = &match, nomiss = &nomiss,
        outmiss = &outmiss, firstrow = &firstrow, lastrow = &lastrow,
         parmcard = &parmcard, dropparm = &dropparm)
```

Figure 3-2. Create macro variables from the table and use them as parameters to an in-house macro

HOW TABLES ARE USED

One of the most popular uses of tables in DCES is to store formats that contain item codes of what consumers spend their money on. Every year there are form changes that need to be made with new item codes added and old item codes deleted. These changes often involve new technology replacing old technology. Figure 3-3 shows a program with hard coded item code format information and using if-statements to control the form changes of the affected year and quarter. Over time, the code can become increasingly lengthy and hard to read.

```
Proc format;
  value $misccod
  %if (&qtr5 >= 20032 ) %then %do;  /* 2003 quarter 2 form change  */
    "100", "110" = "120"
    "240", "250", "260" = "265"
    "290",
    %if (&qtr5 < 20052) %then %do;  /* 2005 quarter 2 form change */
      "300",
    %end;
    "330", "340" = "345";
  %end;
  %else %do;
    "100", "110" = "120";
  %end;
run;
```

Figure 3-3. SAS code with embedded hard-coded form changes

Thus, by moving the format information into a table and applying start and end dates to accommodate form changes every year, the code is easier to maintain and update. The table in Figure 3-4 contains the basic columns needed to create a format data set. The start and end date columns are used to control form changes. Consequently, the table is more manageable and easier to read and maintain than the code in Figure 3-3.

RECTYPE	FORMATNAME	STARTRANGE	ENDRANGE	LABEL	HLO	STARTDATE	ENDDATE
3	8	5	5	15	1	5	5
MIS	$MISCCOD	100	100	120		19941	
MIS	$MISCCOD	110	110	120		19941	
MIS	$MISCCOD	240	240	265		20032	
MIS	$MISCCOD	250	250	265		20032	
MIS	$MISCCOD	260	260	265		20032	
MIS	$MISCCOD	290	290	345		20032	
MIS	$MISCCOD	300	300	345		20032	20051
MIS	$MISCCOD	330	330	345		20032	
MIS	$MISCCOD	340	340	345		20032	
A11	$XVAL	A	B	VB		20052	
A11	$XVAL	C	G	IB		20052	
A11	$XVAL	R	R	IB		20052	
A11	$XVAL	0	0	ZERO		20052	
A11	$XVAL	OTHER	OTHER	VALUE	O	20052	
A11	XVAL	.A	.B	VB		20052	
A11	XVAL	.C	.G	IB		20052	
A11	XVAL	.R	.R	IB		20052	
A11	XVAL	0	0	ZERO		20052	
A11	XVAL	OTHER	OTHER	VALUE	O	20052	

Figure 3-4. Daformat table in Excel, another use of a table to store format information

Once the formats are moved to a table, an in-house macro, Gensastb, will convert the .dat file, created by saving the Excel table into a comma-delimited file, to a SAS data set. It will be used to drive the code in Figure 3-5. The embedded macro %DaCreateFormat will create a permanent or temporary format for the desired rectype (record type). The PROC SQL statement will create the output control data set called Format to contain information that describes the format for a particular rectype and date. The column headings will be renamed to the appropriate format variables to provide specific information about each range and value. The column, Type, will be added to indicate whether the format is numeric or character. Once all the desired format requirements are met, the output control data set will be used as the input control data set to create the desired format.

```
%macro DaFormat;

   /* Internal macro that will create the format */
   %macro DaCreateFormat(Fmt_Rectype = , Permanent = );

      %if (&permanent = Yes) %then
        %let library = %str(library = library);
      %else
        %let library = ;

      /* Create format for a specific rectype within the start and end date */
      proc sql noprint;
        create table Format as
          select formatname as fmtname, startrange as start, endrange as end, label,
                 hlo
           from Dataadj.daformat
           where (rectype = "&fmt_rectype" and startdate <= &qtr5 and
                 (enddate >= &qtr5 or enddate = .));
      quit;

      /* Assign the Type (Character or Numeric) Column */
      data Format;
        set Format;
        if (substr(fmtname, 1, 1) = '$') then
          type = 'C';
```

```
        else
          type = 'N';
      run;

      /* create the format from the format data set defined above */
      proc Format cntlin = Format &library;
      run;

   %Mend DaCreateFormat;

   /* create permanent format for rectype A11 */
   %DaCreateFormat(Fmt_Rectype = A11, Permanent = Yes)

   /* create temporary format for rectype MIS
   %DaCreateFormat(Fmt_Rectype = MIS)

%Mend DaFormat;
```

Figure 3-5. SAS code utilizing the Table-Driven Methodology to create permanent and temporary format

Utilizing tables allows the end-users to go directly into the table to make any required modifications to the system and provide any additional information the program needs to process. The code that is being driven by the table is reusable because, in this case, it can dynamically generate the format. Most notably, only the table would have to be modified and not the code; hence, by applying the concepts of tables, the process has been enhanced.

CONSUMER EXPEDITURES' CURRENT PROCESS

The purpose of storing the format information in Excel tables is ultimately to convert it into a SAS data set. Although this can be accomplished using PROC IMPORT to convert an Excel sheet to a SAS data set, it presents two major problems. First, by using PROC IMPORT, the Excel table is not converted to a SAS data set with the proper attribute requirement needed by our specifications/requirements. Second, there are firewall and security concerns over the communication between UNIX and the PC server. Due to these two reasons, systematic steps are taken to achieve a SAS data set that fits our requirements and needs (Figure 4-1).

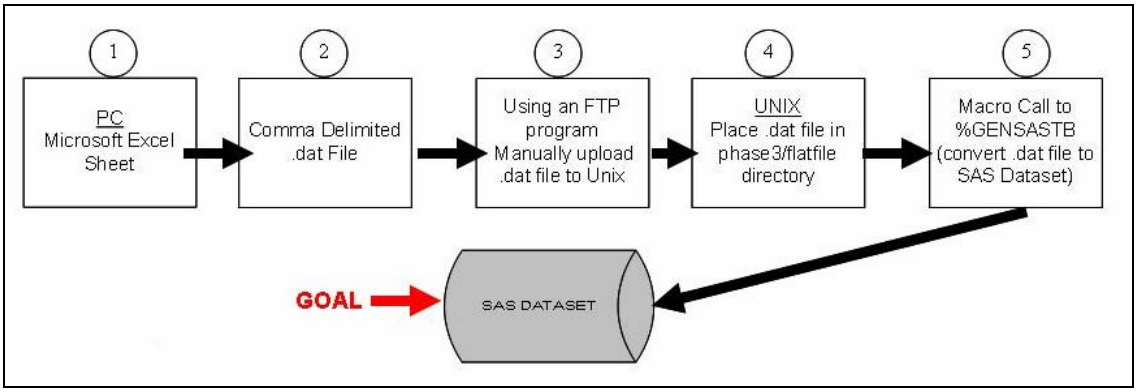


Figure 4-1. Sequence of steps required to attain a SAS data set from an Excel sheet

Step 1: Use an Excel sheet table on the PC server to modify the data.
Step 2: Save the Excel sheet as a comma delimited .dat file.
Step 3: Manually FTP the file to the UNIX server.
Step 4: Once the .dat file is in the UNIX server, place it into a specific directory.
Step 5: Call an in-house macro function to convert the comma delimited file to a SAS data set that satisfies our requirement, with the appropriate attributes.

Here the Table-Driven Methodology works well. It did speed up our process, but it also came with consequences.

ISSUES WITH CURRENT TABLE-DRIVEN PROCESS

The first issue currently facing our Table-Driven process is the number of manual steps needed to modify the Excel sheet table on PC, in order to get a SAS data set on the UNIX server. These manual steps can leave room for potential error. Second, there are no user-input validations when entering data into the Excel table to catch any typographical or data entry errors. Any minor changes needed to fix errors in the excel sheet would require the whole process to be repeated. This can be time consuming and again leaves room for more errors to occur. Even though the Table-Driven Process simplified the code, the steps required to achieve a SAS data set present some obstacles,

especially if there is an error with the user input or using an old .dat file. Hence, a web-based user-interface seems to provide a great solution to these problems that will make this process less painful and more efficient.

ISSUES RESOLVED BY USING WEBAF

Since the goal is a SAS data set, it is possible to use webAF to develop a user-interface that would allow the users to directly access or manipulate the SAS data set (Figure 4-2).

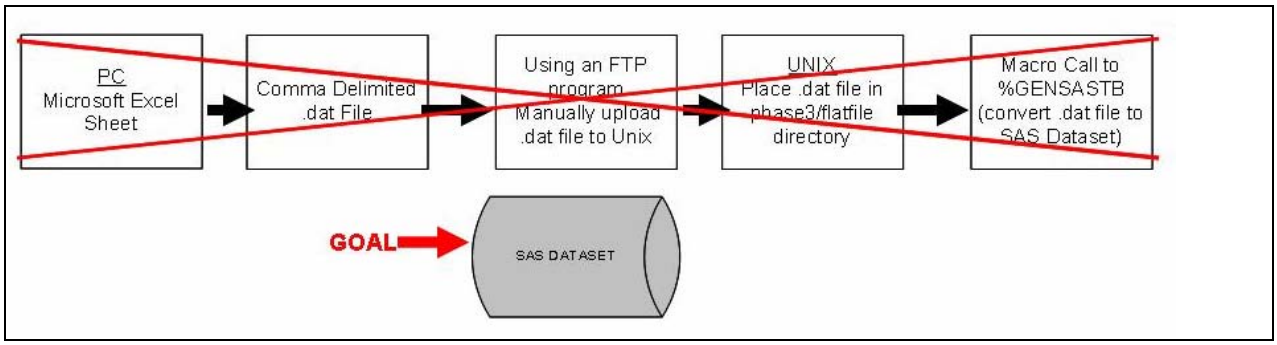


Figure 4-2. Directly access the SAS data set to eliminate all manual steps needed to achieve a SAS data set

It will eliminate the need for an external source, such as Excel sheets, Access or any type of table because now the user-interface will be the interface to the SAS data set. It will allow the user to directly modify the SAS data set without going through some other software. This will streamline the process of modifying the SAS data set. It will also eliminate all the manual steps required to create a SAS data set and decrease the possibility of working with an old .dat file, in case the step to replace the new file with the old file was accidentally skipped. The user-interface can have user-friendly screens with built-in user input validations. It will reduce the error and time involved in running the process with the inputs already checked prior to updating the data set. Building a web-based application allows the flexibility to add desired features if needed. Another advantage of using a web-based interface is that it allows end-users to update or modify a SAS data set without having the SAS System on their machine.

INTEGRATING WEBAF AND THE TABLE-DRIVEN METHODOLOGY

The web application TableDrivenEdit was developed using webAF (Figure 5-1). It is an application that provides a user-interface to directly access the SAS data set icccr that is stored on the SAS/Share Server. The SAS/Share server is configured to use the authentication of the UNIX userid and password. WebAF software makes deploying and testing the web application fast and easy. To deploy, simply start the Java Web Server that comes packaged with webAF and execute the application in the web browser.

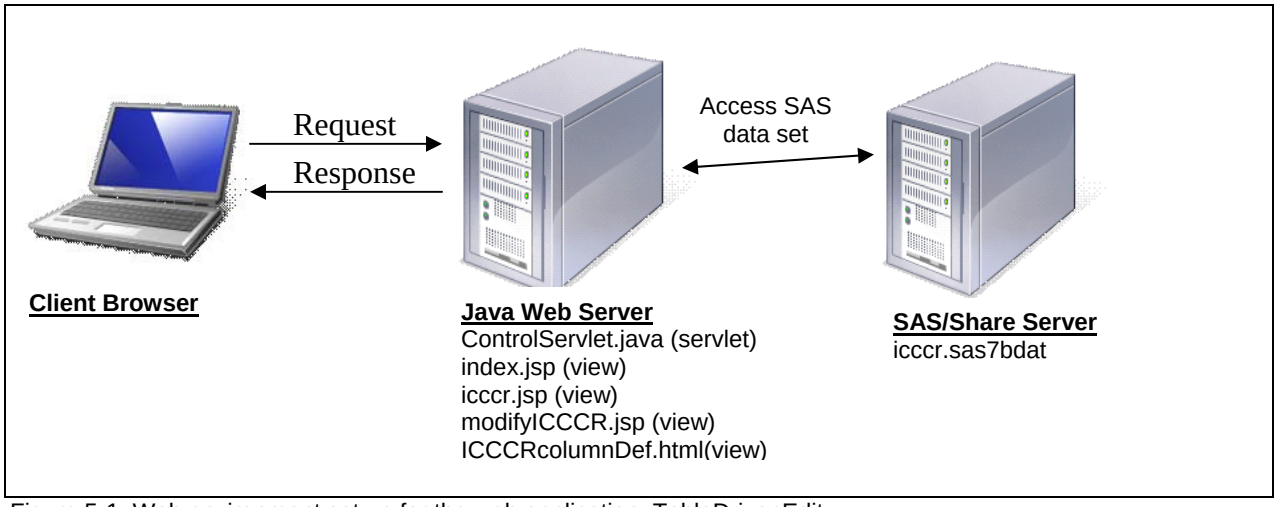


Figure 5-1. Web environment set up for the web application, TableDrivenEdit

The user will use the client browser to access the web application that is loaded to the web server. The initial page the user will see in this application is the index.jsp page (Figure 5-2). In order to enter the system, the user has to input his/her UNIX userid and password and select a table to view. Once the user clicks on the “Submit” button, the request is sent to the ControlServlet. If the UNIX login information is correct, the ControlServlet will then redirect the user to the icccr.jsp page; otherwise, it will redirect the user back to the login screen with an error message, Figure 5-3. Note the URL is now pointing to the ControlServlet.

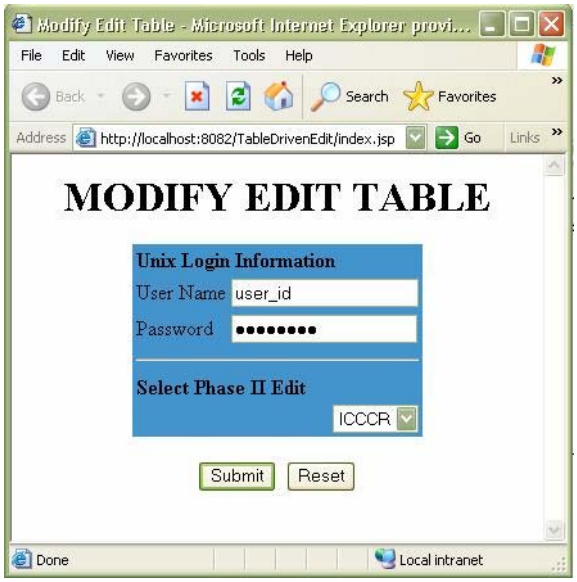


Figure 5-2. index.jsp. Initial Log-In Screen

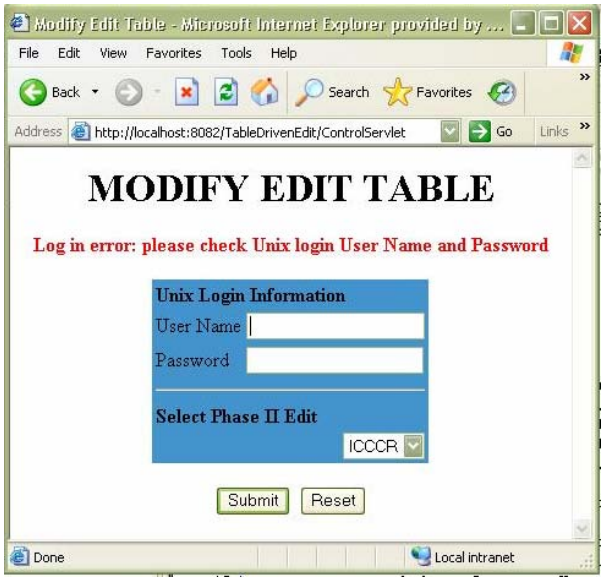


Figure 5-3. index.jsp. ControlServlet redirect back to the login screen due to login error

Upon successful login, the ControlServlet redirect the user to the icccr.jsp page (Figure 5-4), which will display the icccr SAS data set as a table using the JDBCToTableModel View that is set up in the ControlServlet. Here the user will be able to filter the table view of the SAS data set by selecting the appropriate radio options. To add a new row to the data set, the user can click on the "Add New Row" button. To modify the data set, the user can click on the Order link in the table or use the drop down menu to modify the desired Order row in the data set.

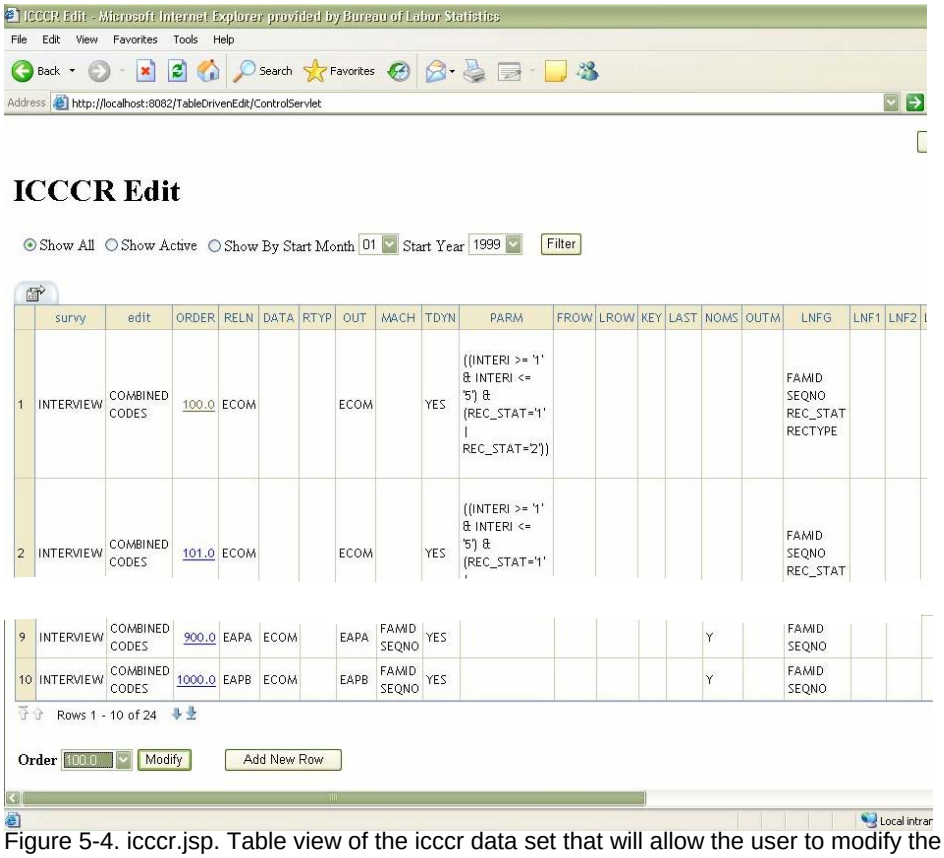
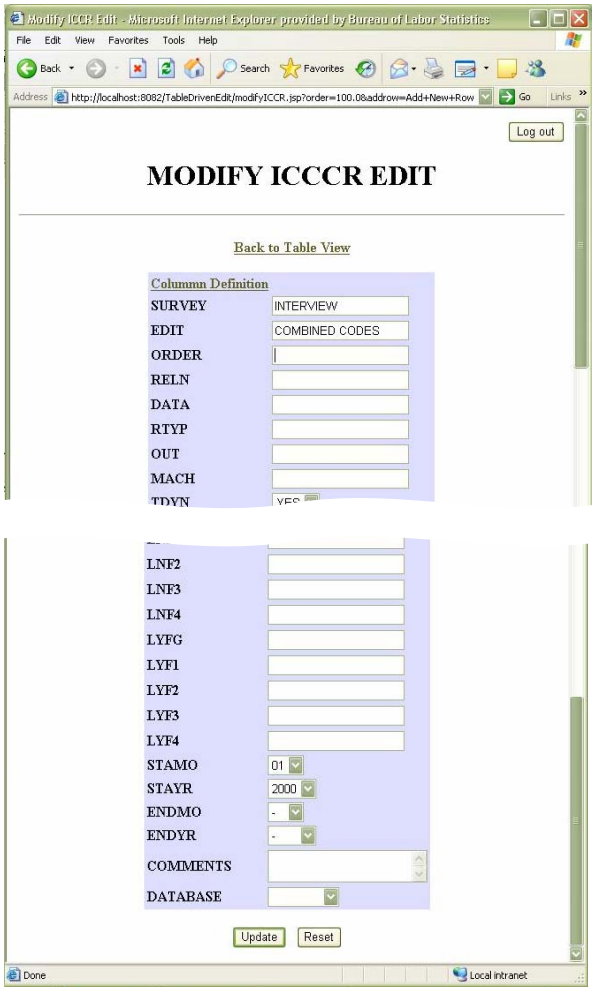


Figure 5-4. icccr.jsp. Table view of the icccr data set that will allow the user to modify the table

Whether the user chooses to add or modify a row, the request will be routed to the modifyICCCR.jsp page. The modifyICCCR.jsp page is an html form that will dynamically create the appropriate page based on the url parameter that is passed to it by the icccr.jsp page, either to add a new row, Figure 5-5, or to modify an existing row, Figure 5-6.



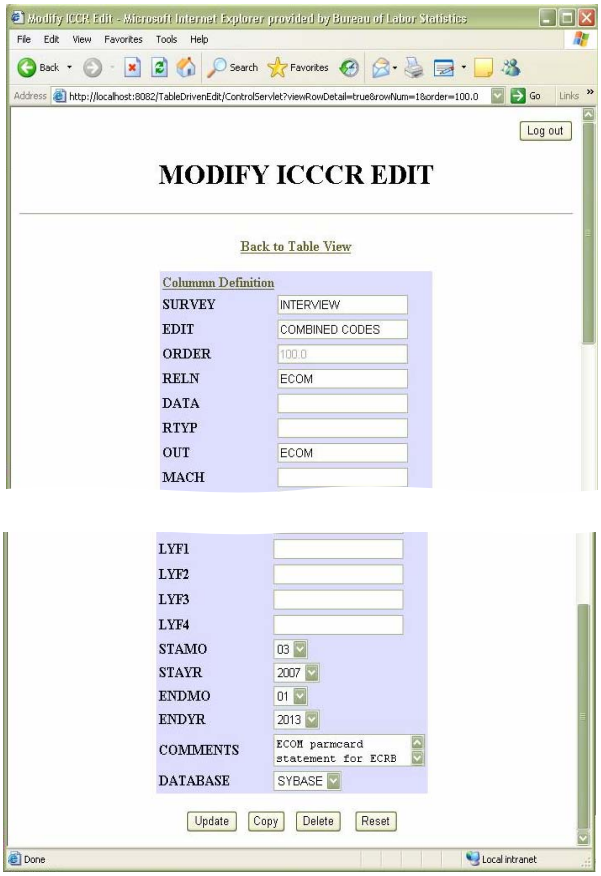


Figure 5-6. modifyICCCR.jsp. To modify a row

Figure 5-5. modifyICCCR.jsp. To add a new row

The form also contains data validation rules to eliminate user input errors and utilizes drop down menus whenever possible to decrease typing errors. One of the validation rules in place for this application is for the start date and end date field. If the end date is greater than the start date, then an error window, Figure 5-7a & b, will pop up with respective error message. Once the user input satisfies all the conditions of the validation rules in the form, the page request will be sent to the ControlServlet to add a new row or update the SAS data set.

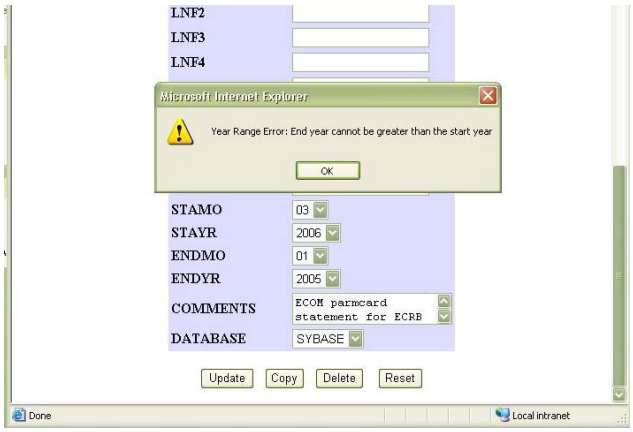


Figure 5-7a. modifyICCCR.jsp. User-validation for the ENDYR (end year) field

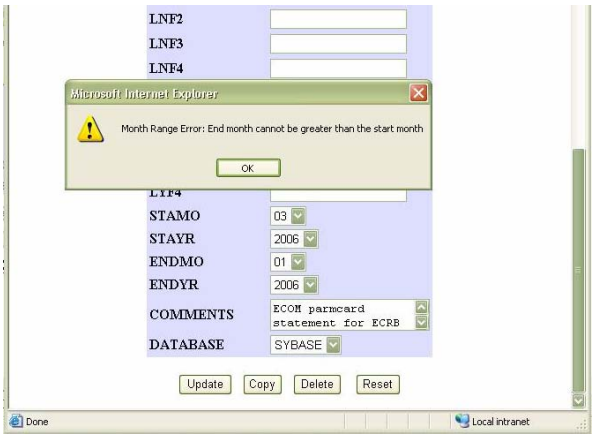


Figure 5-7b. modifyICCCR.jsp. User-validation for the ENDMO (end month) field

The validation rules were added to the form to satisfy user requirements. A user-friendly feature was also added to the form with the embedded link "Column Definition" above the column heading. The user can click on this link to view the description of the field names (Figure 5-8). It will help the user update the SAS data set by providing the definition of what each field means.

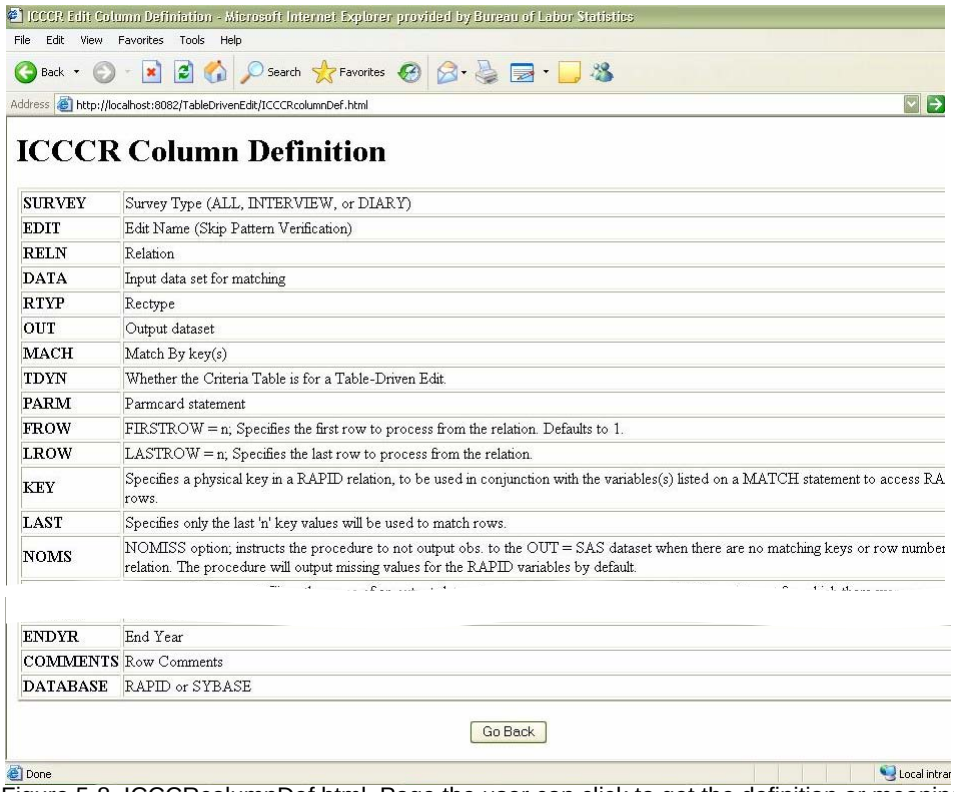


Figure 5-8. ICCCRcolumnDef.html. Page the user can click to get the definition or meaning of the fields in the form.

CONCLUSION

Using webAF to develop server-side web applications prove to have numerous benefits. By incorporating Java, it has made SAS even more powerful and prevalent. Now SAS developers have the choice to bring their SAS application to a whole new level by incorporating object-oriented programming and web-based technology. Using webAF to develop a web-based user-interface to directly access the SAS data set will effectively execute the Table-Driven Methodology.

REFERENCES

Chopra, Vivek, et al. 2005. *Beginning JavaServer Pages*. Indianapolis,IN: Wiley Publishing Inc.

Hillhouse, Anita, Kari Richardson, and Eric Rossland. 2004. *Server-Side Web Application Development Using webAF Course Notes*. Cary,NC: SAS Institute Inc.

Wright, Jeff and Greg B. Nelson. "Developing and Deploying Java Applications Around SAS: What they didn't tell you in class" *Proceedings of the twenty-ninth Annual SAS® Users Group International Conference*. May 2004. <http://www2.sas.com/proceedings/sugi29/030-29.pdf> (July 2, 2006).

Waxman, Mickey and Larry Hoyle. "SAS webAF for Java Application Development, a First Sip" Proceedings *of the twenty-forth Annual SAS® Users Group International Conference*. April 1999. <http://www2.sas.com/proceedings/sugi24/Handson/p155-24.pdf> (June20, 2006).

Girardin, Robert. "Introduction to the SAS Custom Tag Library" Proceedings *of the twenty-eigth Annual SAS® Users Group International Conference*. April 2003. <http://support.sas.com/rnd/appdev/doc/sugi28p60-28.pdf> (July 25, 2006).

SAS Institute Inc. 2006. "Welcome to the SAS AppDev Studio Developer's Site." http://support.sas.com/rnd/appdev/webAF/index.htm.

Sun Microsystems, Inc. 2006. "Java Servlet Technology." http://java.sun.com/products/servlet.

ACKNOWLEDGMENTS
I want to thank Scott Ankers who thought of the idea of using a web interface to directly access a SAS data set and William Chan for his knowledge of the table-driven methodology. I also want to thank David Caveney and William Chan for helping me to review and edit this paper. The success and outcome of this paper would not be possible without their ideas and inspiration.

CONTACT INFORAMTION
Your comments and questions are valued and encouraged. Contact the author:

Shelly Yiu-Morrison
Bureau of Labor Statistics, DOL
2 Massachusetts Ave, NE Suite 5250
Washington, DC 20212
(202) 691-5148
morrison.shelly@bls.gov