

Stored Process Tips and Tricks

Faron Kincheloe, Baylor University, Waco, TX

ABSTRACT

The stored process can be a powerful tool. Learn how to use it to its full potential. Among the topics discussed will be how to use the stored process to pre-filter data for an information map as a way of enforcing strict security rules. This paper will discuss how to use a stored process to dynamically create a menu that supplies user selections to a second stored process and how to change the type of output that is delivered to the web browser. It will also discuss the ramifications of assigning the stored process to the workspace server as opposed to the stored process server when the stored process is created.

INTRODUCTION

The stored process is nothing more than a SAS® program written to perform a specific task. However, since the stored process is executed on a server, the user does not need a Base SAS® installation to run the program. Stored processes can be accessed from a number of BI tools including Enterprise Guide and the Information Delivery Portal. The SAS® documentation for stored processes recommends using Enterprise Guide to develop stored processes because this allows you to test the stored process and register it in metadata using the same application. However, this paper will discuss editing stored process programs manually using the standard SAS® Display Manager interface and the SAS® Management Console. At the writing of this paper, BI applications have not been available in version 9.2 long enough to determine what changes have been made in the area of stored processes. Therefore, this paper is based upon the operation of stored processes under the SAS® version 9.1.3 platform.

This paper assumes the reader has a working knowledge of the SAS® Management Console and has the ability to create libraries, import tables, and assign security within the Management Console. The reader should also have a working knowledge of the Information Delivery Portal, Data Integration Studio, Information Map Studio and Base SAS® programming.

REGISTERING IN MANAGEMENT CONSOLE AND CHOOSING A SERVER

In order for a stored process to be available for BI client applications, it must be registered in metadata. To register the stored process in the Management Console, log in with an administrator account or other account that has sufficient rights to manage metadata. Go to Environment Management > BI Manager > Stored Process. It is recommended that you create sub-folders under the Stored Process folder to allow you to organize your stored processes and manage security for them. Right Click on the folder and select “New Stored Process”.

You will be prompted to enter a name by which the stored process will be identified and, if desired, a description of the stored process. The name and description entered here are displayed in the portal. The name can contain spaces, but if you want to use a custom menu from the portal, it is recommended that you avoid special characters as they may cause difficulties with the web application server.

The next step in the registration process is to choose the server that will execute the stored process. You must choose between a Workspace Server and a Stored Process Server. The decision you make depends upon the purpose of the stored process and the type of output you desire. If you want the stored process to send streaming output to your web browser from the portal, you must use the Stored Process Server. If you want the stored process to pre-filter data used by an information map, you must use the Workspace Server. Both of these choices have security implications. The SAS® Workspace Server is launched using the end-user's credentials so access to data will be automatically restricted by the end user's permissions. However, the SAS® Stored Process Server is launched using the login credentials of the SAS General Servers group (that is, the sassrv account). This account has widespread access to data within BI. Therefore, the stored process must be registered in a folder that is restricted to only those users who should have access to the data read by the stored process. By locking down these folders with Access Control Templates you can ensure that the stored process is available only to those users who should have access to the information provided by the stored process. **Figure 1** below shows an example of stored process folders set up for various groups.

Name	Description	Type	Last Modified
Undergraduate	Stored Processes related to Undergraduates	Folder	Mar 22, 2007 3:24:42 PM
HRCompBenefits	Stored processes to replace HRVision Product. 7/6/07 FDK	Folder	Sep 11, 2007 2:41:04 PM
UG Admissions	Stored Processes for UG Admissions 8/2/07 FDK	Folder	Aug 2, 2007 4:01:34 PM
Parents League	Stored Process for Parents League List 8/20/07 fdk	Folder	Aug 20, 2007 4:00:17 PM
UG Admissions Shared	Stored Processes for UG Admissions & external users	Folder	Aug 30, 2007 11:44:56 PM
Dept Budget Reports		Folder	Oct 3, 2007 2:20:54 PM
HR Masters	Processes for Jobs & Emp master tables FDK 10/3/07	Folder	Oct 3, 2007 2:14:32 PM
Budget Office	Stored Processes for Budget Office reports 10/22/07 FDK	Folder	Oct 22, 2007 1:46:26 PM
IRT	Stored Process for IRT use. 11/5/07 FDK	Folder	Nov 5, 2007 2:50:48 PM
Payroll	Stored Process for Payroll use---JUM 11/8/07	Folder	Nov 8, 2007 3:56:16 PM
Graduate	Stored processes for graduate reporting 11/15/07 FDK	Folder	Nov 15, 2007 1:48:54 PM
Internal Audit	Internal Audit Reporting System	Folder	Feb 20, 2008 1:18:00 PM
Records		Folder	Feb 29, 2008 3:44:40 PM
Registration		Folder	Jul 26, 2008 4:04:30 PM
Sponsored Programs	Stored Processes for Sponsored Programs---11/20/08---JUM	Folder	Nov 20, 2008 8:41:27 AM
Cashiers		Folder	Dec 15, 2008 9:16:22 AM
Law		Folder	Jan 14, 2009 11:29:03 AM
Success Center		Folder	Jan 21, 2009 4:18:57 PM
Graduation	Stored processes for graduation reporting	Folder	Feb 2, 2009 2:20:56 PM
FinancialAid	Programs for Financial Aid office	Folder	Apr 30, 2009 2:38:01 PM

Figure 1: Stored Process Folders

Figure 2 and Figure 3 show the permissions that have been applied to the Undergraduate folder for the Undergraduate and Public groups respectively using an access control template. The specific permissions applied by the template are indicated by the green background around each permission.

Permissions	Grant	Deny
ReadMetadata	<input checked="" type="checkbox"/>	<input type="checkbox"/>
CheckInMetadata	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Create	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Administer	<input type="checkbox"/>	<input checked="" type="checkbox"/>
WriteMetadata	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Read	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Write	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Delete	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 2: Undergraduate Group Permissions

Permissions	Grant	Deny
ReadMetadata	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CheckInMetadata	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Create	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Administer	<input type="checkbox"/>	<input checked="" type="checkbox"/>
WriteMetadata	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Read	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Write	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Delete	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 3: Public Group Permissions

The next step in registration is to specify the name and location of the SAS® program code that is executed by the stored process. The location is specified by choosing a predefined path called a repository. If this is the first time a program is being registered from this path, click on the Manage button and add the path as a new repository. Enter Source file as name of program with a suffix of .sas. The stored process will not work if the suffix is missing from the file name.

After identifying the source program, you must select the output type. If you are using the stored process to pre-filter an information map select None. If you are using the stored process to send information back to a web browser through the portal, you must choose Streaming to be able to send output to the _webout destination.

The final step in registering a stored process is to specify parameters. Parameters are not utilized by the types of stored processes being discussed in this paper.

PRE-FILTERING AN INFORMATION MAP

Information maps were designed so that a stored process can be used to subset or pre-filter the data on which the information map is based. A stored process that is associated with an information map is executed before any of the queries that are generated from that information map. This processing order enables you to use SAS tools such as the DATA step or the macro language to process the data that will be used as input for the information map. One reason for pre-filtering an information map would be to enforce user based security to restrict the rows of data to only those which each user is authorized to see. This technique will be used as an example for this paper.

DATA STRUCTURE

In order to facilitate user based security, the data must contain some sort of hierarchy information. Virtually every organization has some hierarchy and this information may already exist in the data. For this example, we will use three levels that exist in a university; campus, school or administrative division, and department. **Table 1** shows a simple dataset containing employee position and salary information along with the hierarchy information. It may appear redundant to have a field for campus when all rows have the same value. However, this provides a simple way to give access to someone like the president who has access to all rows.

ID	Position	Salary	Campus	Division	Department
887555001	Lab Manager	\$24,000	Waco	Arts&Sci	Biology
887555002	Administrative Assistant	\$25,000	Waco	Arts&Sci	Biology
887555004	Lecturer	\$48,000	Waco	ECS	Engineering
887555007	Professor	\$96,000	Waco	Arts&Sci	Criminal Justice
887555008	Assistant Director	\$80,000	Waco	FinAdmin	Budget Office
887555013	Programmer Analyst	\$33,000	Waco	FinAdmin	Institutional Research

Table 1: Employee Positions Dataset (Emp_Position)

Table 2 shows the BI Security table which contains the logins that are registered in metadata and the access information associated with each user. One of the first steps in each stored process reads the BI Security table and initializes a macro variable that contains the access information. The macro variable is used to construct a where clause that will subset all of the data that is read by the stored process.

Login	Access
Papa_Bear	Campus='Waco'
Goldie_Locke	Department in ('Biology', 'Chemistry')
BB_Wolfe	Division='FinAdmin'
RR_Hood	Division='ECS' or Department='Institutional Research'

Table 2: BI Security Dataset (BI_Users)

There is essentially no limit to the access combinations that can be granted to any user to match responsibilities that cross organizational boundaries. The one limit we discovered was the inability to pass the macro value from one stored process to another due to the quotation marks in the data. The workaround for this was to have each stored process go back to the BI security table and create the macro value over again.

IMPLEMENTATION PROCESS

There are 5 steps involved in the implementation of a stored process as a filter for an information map:

1. Define a source table and library in metadata.
2. Write a SAS program that filters or subsets data and writes the output to the source table.
3. Register the program as a stored process.
4. Create the information map.
5. Associate the stored process with the information map.

SOURCE TABLE AND LIBRARY

For the purposes of this procedure, the table containing the original, unfiltered data will be referred to as the primary table. A subset of the primary table as a result of the filtering in our stored process will be called the source table since it is the source for our information map. In reality, the source table does not actually exist until the stored process is executed. However, a “dummy” source table must physically exist in order to start the metadata configuration process. The source table can be in the same physical location and can even be the same table as the primary table as long as the library for the source table has a separate definition in metadata. For easy recognition, it is recommended that the name of the library contain STPOUT (e.g.: HRSTPOUT). We chose to create a separate table (e.g.: EMPFLTR) for the source table to avoid confusion and reduce the risk of overwriting our primary table. Unless the SAS stored process program contains an SQL statement or other detailed code to control the structure of the source table, the source table needs to be a mirror image of the primary table. Otherwise, the “dummy” source table will need to be identical to the table created by the stored process. The physical source table does not need to contain any rows of data. It exists only as a reference from which the Management Console can read the table structure. Once the source table has been created and the library defined in the Management Console, use the Import Table function in Management Console to import the source table structure into metadata. Any changes that are subsequently made to the layout of the primary table or to the stored process code must also be reflected in the “dummy” source table. After the “dummy” source table has been updated, the metadata for the source table can be updated from within Data Integration Studio. If the metadata does not match the physical structure of the table, the information map will have difficulty reading the data and is likely to cause an error in the application such as Web Report Studio.

SPECIAL ELEMENTS OF THE PROGRAM

The stored process used in this technique is simply a SAS program that subsets data based on any type of query, matching, or logic that the user desires. It is recommended that the primary program code that produces the output be written and tested as much as possible before the program is modified to function as a stored process.

There are four significant elements that the program must contain in order for the pre-filtering technique to be successful. First, there must be a libname statement redirecting your source table library to the work library. This enables the information map to read the temporary source table created by your stored process code instead of the “dummy” source table that was used to define the metadata. An example statement is shown below:

```
libname hrstpout (work);
```

Second, there must be a ProcessBody comment near the top of the program. This identifies the program code as a stored process. An example is shown below. (The %global statement is not required for pre-filtering but is used in our security filtering as will be explained later.)

```
%global bi_where;  
*ProcessBody;
```

Third, there must be a step that retrieves the user's access level from the security table and initializes the access level macro variable (bi_where). This step is not actually required as part of pre-filtering for information maps in general, but is required for our example since we are filtering based on user level access permissions.

```
data _null_;  
set bisec.bi_users;  
where upcase(login)=upcase("&_metauser") ;  
call symput('bi_where',trim(access));  
run;
```

Fourth, the program must have at least one DATA step or SQL procedure that produces the source table with the library, name and structure that is defined in metadata from the steps above. This may be as complex as dictated by your needs. A simple example is shown below:

```
data hrstpout.empfltr;  
set hrdata.Emp_Position;  
where &bi_where;  
run;
```

The entire text of the stored process program is shown below. By declaring `bi_where` as a global macro variable, its value will be empty if the logged in user has not been set up in the security table. The entire filtering code has been encapsulated within the `imfilter` macro to enable us to conditionally execute certain data steps. If we have implemented our security policy correctly, users should not have access to the information map unless they are set up in the security table and added to the appropriate security groups. However, as a failsafe mechanism, the source dataset will be created with zero observations when the user is not found in the security table. This prevents the stored process from aborting with an error that would have otherwise been caused by an incomplete `where` clause.

```
libname bisec meta library='BI_Security' access=readonly; * Location of User
Security Table;
libname hrdata meta library='HR_Data' access=readonly; *Location of primary
data;
libname hrstpout (work); *Redirects filtered data to the work library;

%global bi_where;

*ProcessBody;

*Initilize the bi_where macro variable with the subset clause read from the
bi_users table;
data _null_;
set bisec.bi_users;
where upcase(login)=upcase("&_metauser") ;
call symput('bi_where',trim(access));
run;

%macro IMFILTER;

%*As a safety precaution, create an empty dataset if the user does not have
assigned access;
  %if %quote(&bi_where) = %then %do;
    data hrstpout.empfltr;
    set hrdata.Emp_Position (obs=0);
    run;
  %end;
  %else %do; %*If the user has access, create a subset of emp_position;
    data hrstpout.empfltr ;
    set hrdata.Emp_Position;
    where &bi_where;
    run;
  %end;
%mend;

%imfilter;
```

REGISTER THE STORED PROCESS IN MANAGEMENT CONSOLE

The stored process must be registered in Management Console as described earlier in this paper. As a reminder, it must be assigned to the SAS® Workspace Server and will be launched using the end-user's credentials. Select an output type of None.

CREATE THE INFORMATION MAP

Open Information Map Studio and login as `sasadm` or with an account that has sufficient privileges to create the information map.

Click `Insert>Table`.

Expand the source table library and select the source table that was created and registered earlier in this procedure. Click `OK`.

The table should now be visible in the Physical Data pane of the Information Map Studio window. Click the plus sign beside the table name. Use the blue arrows in the center to move selected fields into the Information Map. Only those fields in the Information Map pane will be available for users of Web Report Studio.

Save the Information Map. When the Save As window appears, double click on the following folders in sequence to reach the desired location: BIP Tree, ReportStudio, Maps, *appropriate folder*. (Choose or create a folder based upon your predetermined organizational and security structures.) Enter a descriptive name for the information map in the Name field and click Save.

ASSOCIATE THE STORED PROCESS WITH THE INFORMATION MAP

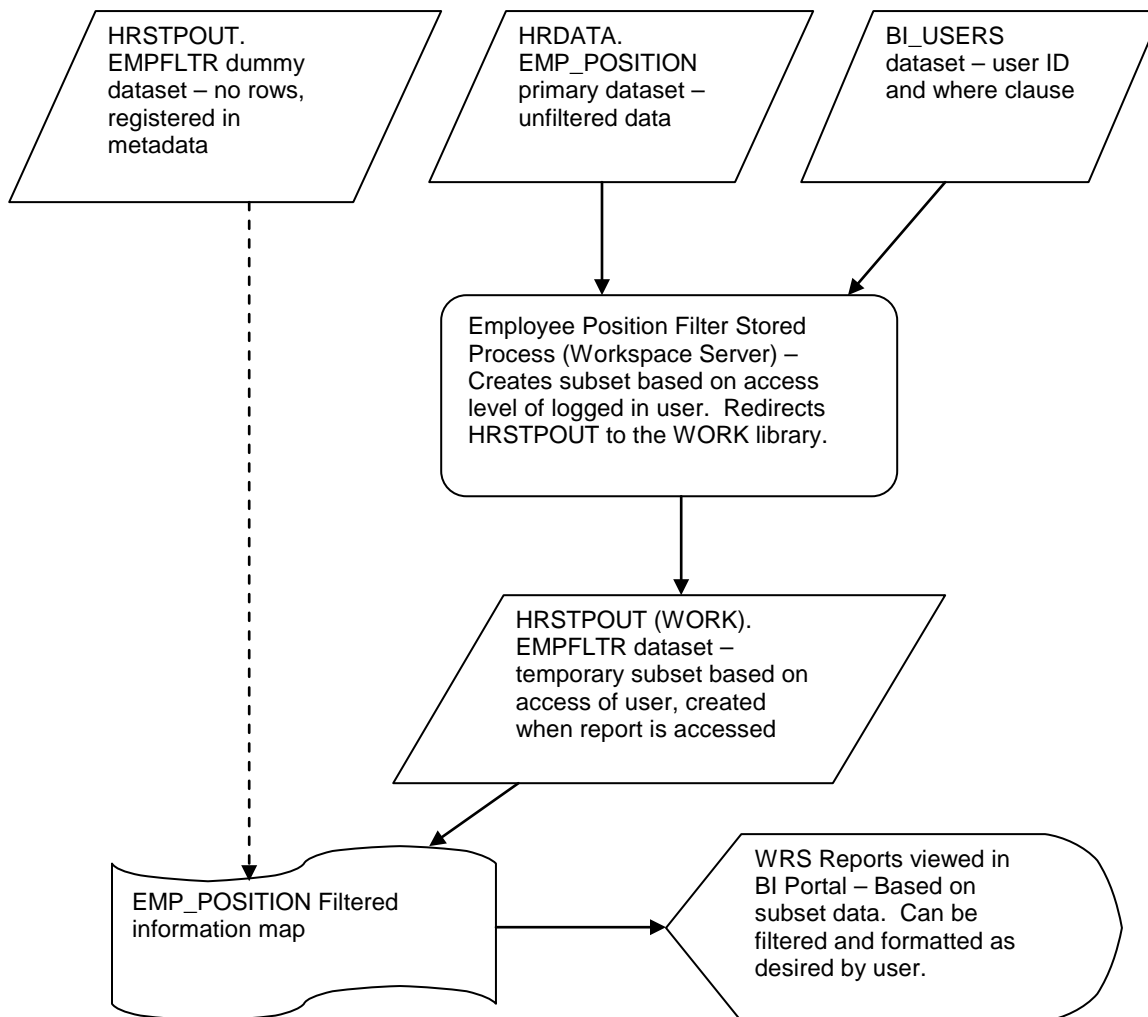
From the SAS Information Map Studio menu bar, select Tools > Stored Processes. Select “Use the selected stored process” and then select the desired stored process from the list of available stored processes. If your stored process is not available it most likely was not properly registered in metadata and assigned to the Workspace server. Click OK.

From the SAS Information Map Studio menu bar, select Tools > Test. Select a couple of fields from “Available items” and use the blue arrow to move them over to “Selected Items”. Click the “Run Test” button.

Depending on the results of the test, you may need to click on the “Show Exception Text” button or “Show Server Log” to examine the log and verify that the stored process is running properly. (Note: *You may need to use File > “Switch Metadata Profile” to log in as a different user for the test if sasadm is not configured to access all data.*)

Save the information map and exit Information Map Studio. You may now use Web Report Studio to build reports based on your pre-filtered information map.

The diagram below provides a graphic illustration of the various elements of the pre-filtering process and how they relate to one another.

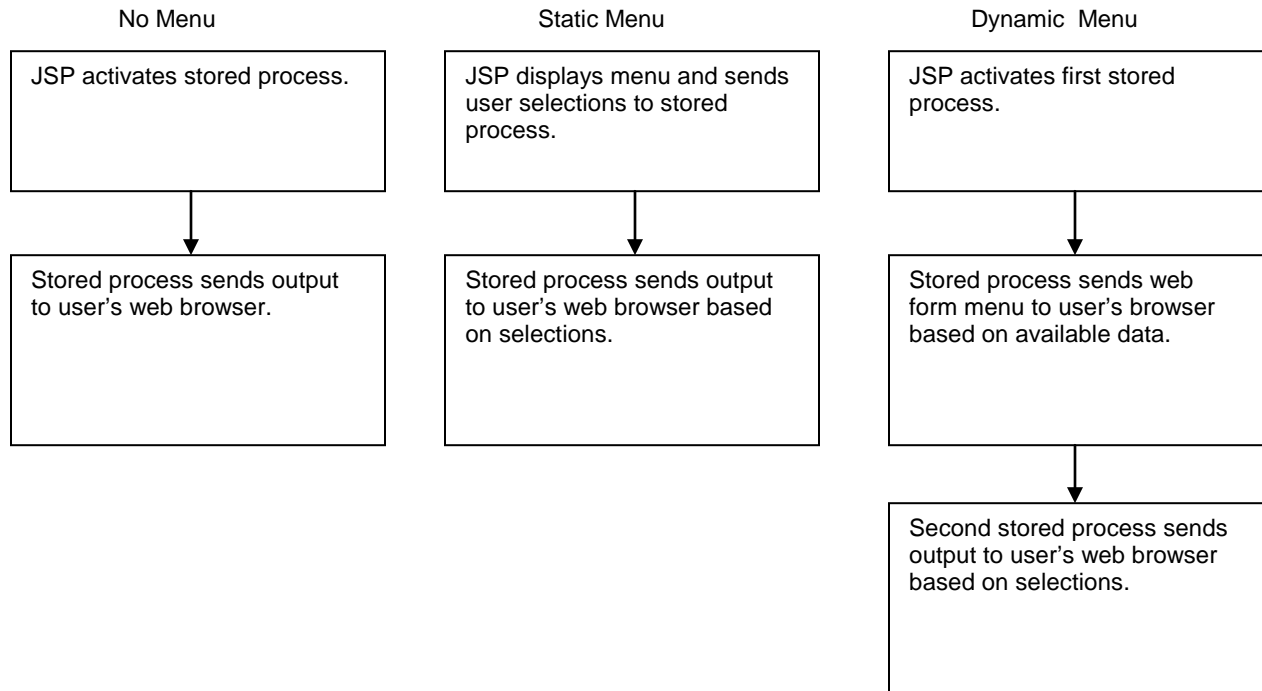


RUNNING STORED PROCESSES FROM THE PORTAL

This section will describe how to run a stored process from the portal without getting the default Stored Process Dialog page. It will also provide instructions for changing the format of the output delivered by the stored process.

Depending on the complexity of the task being performed by the stored process, there may be multiple SAS programs required. It is common in our installation to have a stored process that creates a menu based on available data and then allows the user to launch a second process from that menu to create the actual report output. As a minimum, there must be one SAS program and a corresponding Java Server Page (JSP). Even when no user input is required, there must be a JSP to trigger the execution of the stored process program. If the portal does not locate a JSP, it will display the Stored Process Dialog which is not intuitive for our users and causes unnecessary delay.

The diagrams below illustrate the three main scenarios in which a stored process will be configured for use in the portal:



IMPLEMENTATION PROCESS

There are 4 steps involved in the implementation of a stored process for use in the BI portal:

1. Write or convert the SAS program(s).
2. Register the program(s) as a stored process in the Management Console.
3. Create the JSP that invokes the stored process program.
4. Add the stored process to a portlet in the BI portal.

WRITE THE PROGRAM

It is recommended that the primary program code that produces the output be written and tested before the program is modified to function as a stored process.

DIFFERENCES FROM A TRADITIONAL PROGRAM

The stored process is very similar to an Application Dispatcher program that was used by SAS/IntrNet®. Many of our stored processes are converted Application Dispatcher programs. Only minor changes are required for a conversion. There are two major differences between a program run as a stored process and a traditional SAS program.

First, the DATA step and procedure portion of the program is encapsulated within a macro in most of our stored process programs. This allows us to use macro logic to trap errors and incorrect user input. It also allows us to programmatically "write code" based on the user's selections. The sample code shown below is the beginning of a macro from within a stored process. This stored process is activated by a static menu in which the user selects the

day, month, and year of a start date. The form is designed so that if the user fails to select a day or year, the default value of 0 is passed to the stored process. The default value of NA is passed through if no month is selected. If one of these default values is detected, the DATA _null_ step is executed. This step displays, in large red font, a message instructing the user to enter a valid date. Otherwise, the macro will execute the reporting portion of our stored process program which will be placed at the end of our sample code below.

```

%macro runrept;
  /* Check for valid start date */
  %if ((&startYear = 0) or (&startDay = 0) or (&startMonth = NA)) %then
    %do;
      data _null_;
      file _webout;
      put "<html>";
      put "<body>";
      put "<font color='#FF0000'><center><h1>ERROR: Please
          select a valid month, day, and year for the
          start date.</h1></center>";
      put "</font>";
      put "</body></html>";
    run;
  %end;
%else /* Produce report based on user selections */

```

The second difference between a stored process and a traditional program is some extra code that is required to format the output of the stored process. Most of the SAS documentation about stored processes directs you to sandwich your program between the %stpbegin and %stpend macros. These macros put a lock on the _WEBOUT destination. This caused some problems for us when we were converting our old Application Dispatcher jobs to stored processes so we have opted not to use this method. Instead we use the _WEBOUT file reference and a DATA _null_ step to direct the output to the web browser in the correct format. The code shown below is an example for creating an Excel spreadsheet using this method.

```

***
*** Download to a spreadsheet
***;
ods listing close;

  *** Define output type as excel;
  data _null_;
    rc= stpsrv_header('Content-type', 'application/vnd.ms-excel');
    rc= stpsrv_header('Content-disposition', 'attachment;
        filename=DescriptiveName.xls');
  run;

ods tagsets.ExcelXP body=_webout style=minimal ;

  *** Your original procedures that produce output go here;
  proc print ...;
  run;

ods tagsets.ExcelXP close;

```

Most of the output produced by our stored processes is either Excel or PDF. HTML is typically not used for data presentation because it does not control page breaks when the output is printed. Therefore, the use of HTML is restricted to output for dynamic menus or warning messages. Since HTML is the default output of a stored process run from the portal, a DATA _null_ step to override the output type is not required for menus or warnings. When configuring output other than HTML, it is imperative that the application type, filename extension, and ODS type are all compatible with each other. Either ODS html or ODS tagsets.excel may be used to produce Excel output. If the output file is large and performance is an issue, try using ODS chtml for Excel output. ODS chtml produces compact, minimal HTML output with no style information. However, since there is no style information, chtml output may be hard to read for some output procedures. The code below configures output for PDF.

```

*** Define output type as PDF;
data _null_;
    rc= stpsrv_header('Content-type', 'application/pdf');
    rc= stpsrv_header('Content-disposition', 'attachment;
        filename=DescriptiveName.pdf');

run;

ods pdf body=_webout style=minimal ;

*** Your original procedures that produce output go here;
proc print ...;
run;

ods pdf close;

```

The stored process can only send output to one file per execution. For example, if your program contains three separate report procedures, you cannot send the output of each procedure to a separate file. However, you can send output from essentially any number of procedures to a single output file as long as the procedures are located within the ODS statement. ODS tagsets.excel gives you the option of creating a separate tab in your spreadsheet for each output procedure. This is the default behavior for tagsets.excel. Be aware that some styles such as minimal do not support the ODS tagsets.excel output type.

CREATING A DYNAMIC MENU

The dynamic menu scenario is a chain of at least two stored processes. The first stored process creates the menu based upon changing parameters such as currently available data. The second stored process creates output information based upon user selections from the dynamic menu. One way to get started with a menu program is to use a web authoring tool to design your menu or web form. Then copy the HTML code into your SAS program editor. Use a series of DATA _null_ steps with PUT statements inside a SAS macro to supply the dynamic information and send the HTML code out to a web browser. The HTML form shown below was created by a stored process that checks to see which semesters are available and then populates the selection box accordingly.

Most of the code that produced this form is shown below along with some explanation of the various components of the process. As mentioned earlier, the code is typically sandwiched inside a macro. In this simple example there is no macro logic so the macro is not actually required. However, if there was a possibility that there would be no datasets matching our criteria, we could check for that and use macro logic to display a message instead of trying to create a menu with no selections. The DATA step shown below creates a list of semesters for which data is available in the STUDENT library. The sashelp.vmember view provides a listing of all datasets in all currently assigned libraries. Our datasets have a naming convention of ENROLLMENT_200910 where 200910 represents the Spring 2009 semester. We use substrings to make sure we don't go back further than the year 2000 and to parse the semester value out of the dataset name. We have a format named \$term that we use to supply the "friendly" name of the semester.

```

%macro PUTMENU ;
  data menudata (keep=term termname);
  set sashelp.vmember;
  where libname='STUDENT' and substr(memname,1,10)='ENROLLMENT_'
  and substr(memname,11,4) ge '2000';
  length term $6;
  term=substr(memname,11,6);
  termname = put(term,$term.);
run;

```

We want to display the most recent term first in the menu so we sort the data in descending order.

```

proc sort data=menudata;
  by descending term;
run;

```

This DATA _null_ step begins the definition of the HTML form. The first PUT statement defines the body. The second and third PUT statements instruct the portal to call the second stored process when the submit button is clicked. The program that we are looking at is the first stored process and we are creating a menu that will call the second stored process to produce our output. The _program value must contain the path and name of the stored process exactly as they are registered in the Management Console.

```

data _null_;
  file _webout;
  put "<body>";
  put "<form ACTION='&_URL'>";
  put "<input TYPE='hidden' NAME='_program' VALUE='/Stored Process/Student
Data/EnrollByLevelOutput'>";

```

The remaining PUT statements in this DATA step define the HTML table in which we will put our selection menu.

```

  put "<table BORDER=1 CELLSPACING=0 CELLPADDING=4 WIDTH='80%'>";
  put "<tr ALIGN=LEFT VALIGN=TOP BGCOLOR='#003300' >";
  put "<td ALIGN=Center VALIGN=TOP ><font color='#FFFFFF'><B>Enrollment by
Level of Student (12th Class Day)<B></td>";
  put "</font></tr>";
  put "<tr BGCOLOR='#FCFBED'>";
run;

```

The next DATA _null_ step reads the dataset that contains our list of semesters and creates the select menu. On the first observation of the dataset, we define our select input type. By naming it menuv this selection will initialize the menuv macro variable and pass it on to the next stored process. As the DATA step continues, each row of our data will become one of the choices in our menu.

```

data _null_;
  file _webout;
  set menudata;
  if _N_ = 1 then
  do;
    put "<td align=center valign=top WIDTH='40%'>";
    put "<BR><font size=3>Choose a semester:</font><br>";
    put "<select size=3 name='menuv'>";
    put "<option value='' term '' SELECTED>" termname "</option>";
  end;
  else do;
    put "<option value='' term ''>" termname "</option>";
  end;
run;

```

The final DATA _null_ step closes out our selection menu along with the table and the web form. It also creates a submit button and reset button at the bottom of the form.

```
data _null_;
  file _webout;
  put "</select>";
  put "</tr></table><P>";
  put "<input type='SUBMIT' value='Generate Report'>";
  put "<input type='RESET' value='Reset Form'>";
  put "</body></html>";
run;
```

Finally, we close out the macro and call it to complete the stored process.

```
%mend PUTMENU;
```

```
%PUTMENU;
```

REGISTERING THE STORED PROCESS IN MANAGEMENT CONSOLE

The stored process must be registered in Management Console as described earlier in this paper. As a reminder, it must be assigned to the SAS® Stored Process Server and will be launched using the login credentials of the SAS General Servers group. Therefore, the stored process must be registered in a folder that is restricted to only those users who should have access to the data read by the stored process. Select an output type of Streaming.

CREATING THE JSP

The key to eliminating the Stored Process Dialog is the existence of a JSP in the appropriate location on the Web Application Server. When you invoke a stored process from the portal, it looks for a JSP with the same name as the stored process. If it finds one, it will execute the JSP instead of displaying the Stored Process Dialog. In a typical installation, the JSP folders will be located under the Web Application Server in the \SASStoredProcess\input folder. When the BI platform is installed a **Samples** folder is typically installed under the **input** folder. The **Samples** folder contains a number of JSPs from which code can be copied to create a JSP to meet your needs. The folder structure under the **input** folder must exactly match the folder structure under BI Manager in the Management Console. The name of the JSP must also match the name of the stored process in Management Console exactly. If the names of folders or stored processes in the Management Console contain spaces, these spaces must be replaced with underscores on the Web Application Server. It is imperative that you keep a current backup of the **input** folder as this folder will get overwritten should you have to redeploy the portal for any reason such as the installation of a hotfix.

In the stored process scenarios with no menu or with a dynamic menu, you want a stored process to run immediately without any additional user interaction. In this case the JSP needs only the few lines of code shown below. The onLoad command in the BODY tag automatically submits the form when the page is loaded so that no button has to be clicked. The INPUT line instructs the server which stored process to run. If the stored process takes several seconds to produce output, you might want to insert a message line somewhere between the BODY tags instructing the user to wait while output is being prepared.

```
<BODY onLoad="document.sub.submit()">
<FORM name="sub" ACTION="<%= request.getContextPath() %>/do">
<INPUT TYPE="HIDDEN" NAME="_PROGRAM" VALUE="/Stored Process/appropriate folder/process name">
</FORM>
</BODY>
```

If you want to run a stored process from a static menu whose design never changes use web authoring software or copy code from the samples into a text editor to create the desired menu. The names of input fields in the JSP form must match the name of the corresponding macro variables in the stored process program. The FORM definition must contain the ACTION string as shown below. There must also be INPUT lines to define the program and the submit button.

```
<FORM name="sub" ACTION="<%= request.getContextPath() %>/do">
<INPUT TYPE="HIDDEN" NAME="_PROGRAM" VALUE="/Stored Process/appropriate folder/process name">

<input type='SUBMIT' value='Generate Report'>
```

PORTAL

Once you have completed the program and the JSP, add the stored process to a collection portlet on the appropriate page of the portal and TEST!!

CONCLUSION

The stored process is truly a workhorse of the BI platform. By using the tips and tricks described above, you can use the stored process to provide your users with the desired data in the desired format.

REFERENCES

SAS Institute Inc. 2007. *Creating, Distributing, and Using SAS® Stored Processes Course Notes*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2006. *SAS® Information Map Studio 3.1: Tips and Techniques*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Faron Kincheloe
Baylor University
One Bear Place #97032
Waco, TX 76798
Phone: (254) 710-8835
Email: Faron_Kincheloe@baylor.edu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.