Paper 043-2013

Stop Your "Wine"ing: Use a Stored Process!

Tricia Aanderud, And Data, Raleigh, NC
Angela Hall, SAS Institute, Cary, NC

ABSTRACT

One of the major benefits of using SAS® Stored Processes is extendibility. SAS stored processes are one of the most customizable products; there are several advantages, such as the ability to set up reports that can run in various locations, enhance out-of-the box functionality with custom widgets, and all the while leveraging server options and power. In this discussion, you will learn tips and tricks for using stored processes within SAS BI clients.

INTRODUCTION

When new or even experienced users hit a roadblock with the SAS Business Intelligence toolset, they might resort to *whining* or complaining about the tool. Their frustration is the result of having a requirement that should be simple to complete but not knowing an easy way to accomplish it.

This paper suggests some common and advanced ways to use stored processes to solve issues. With the SAS Global Forum only moments away from one of the most famous wine appellations in the world, Napa Valley, we are using data from WINE.COM in the examples.

COMMON USES FOR STORED PROCESSES

SAS stored processes offer a lot of flexibility to BI content developers. Using a stored process, you can perform a variety of tasks, such as generating data sets, creating complex reports from other SAS procedures, and even building web applications. Some examples for stored processes include the following:

- Deliver lists or charts to Microsoft Office applications through the SAS Add-In for Microsoft Office. The user can rerun the stored process later to have updated data.
- Add a report to the Information Delivery Portal using the Stored Process or URL Display portlets.
- Deliver data to the BI Dashboard by using the stored process as the source for the data indicator.
- Enable user drill-down from Web Report Studio reports or BI Dashboard indicator into additional information that is more detailed or output the source data.
- Develop custom applications for user interaction through chained stored processes.

WEB REPORT STUDIO: ADDING A DASHBOARD INDICATOR

You can easily add sliders, dials, and speedometers to your report using a stored process and the SAS GKPI procedure. In Figure 1, the report uses a combination of an information map to display the graph on the left and a stored process to show the average ratings for the vineyard's wine offerings.

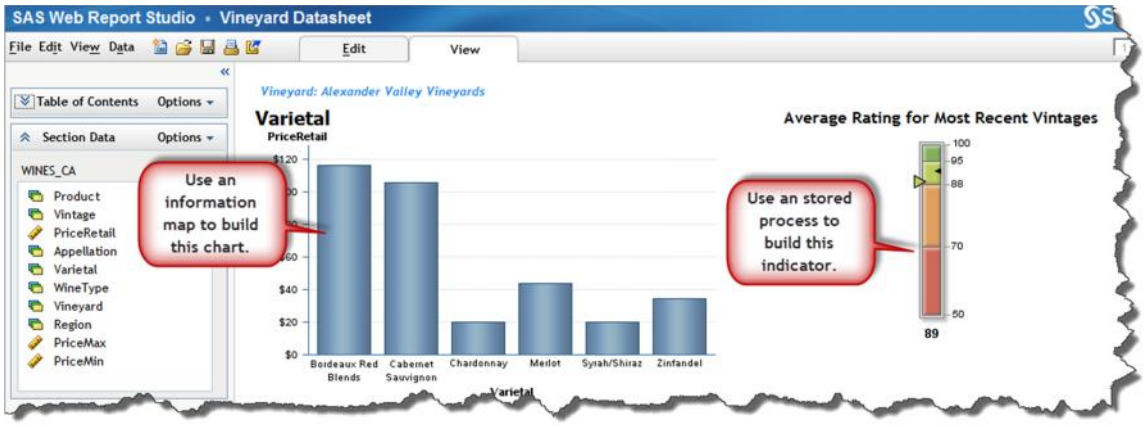

Figure 1 Adding a KPI Dial to Web Report Studio

Write the Code

This example explains how to create the Average Rating for Most Recent Vintages vertical slider shown in Figure 1. Note: Refer to the SAS GKPI Procedure documentation for more examples (reference 1).

Code	Comments

`/*Replace LET statement with code to create the needed values*/`
 	
`%let AvgRating=72;`
	Place your code to set the value in this area. For this example, the value is hard-coded using a macro variable to demonstrate how the PROC GKPI works with the stored process.

`%let _gopt_device=javaimg;`
 	
`goptions reset=all`
`xpixels=210 ypixels=200;`
 	
`title1 ="Average Rating for Most Recent Vintage";`
	For this output to work properly, the graphic device must be set to Java Image.
	
	You can use the GOPTIONS to control the size of the indicator.

`%STPBEGIN;`
 	
`proc gkpi mode=raised;`
`vslider`
`actual= &AvgRating.`
`bounds=(50 70 88 95 100)`
`/target=92`
`lfont=(f="Albany AMT" height=.5cm)`
`label`
`;`
`run;`
 	
`%STPEND;`
	Ensure your LET statements are outside of the %STPBEGIN/%STPEND macros. When you register the stored process, ensure you turn off the stored process macros in the SAS Code pane.
	
	Your stored process might not work as expected if the macro variables are not assigned outside of the macros.
	
	Notice the &Avg_Rating. macro variable is used for the actual calculation. Unlike other procedures, you cannot assign a dataset and use a variable from it. You variable must be assigned using a macro variable or a hard-coded value.

Register and Test the Stored Process

When you register the stored process, make sure you check that the SAS Result Type is *Package*. Add the stored process to the Web Report Studio report using the stored process icon on the Edit menu.

USE AN OLAP CUBE AS A DATA SOURCE

What do you do when all of your data is stored in an OLAP Cube and you need to create a simple query? You can generate the PROC SQL code and parameters for use in a stored process using SAS Enterprise Guide. SAS Enterprise Guide can guide you through the MDX code so you do not have to be much of an expert.

For this example, we want to rank the wines based on average ratings during 2007 and 2009. As shown in the following figure, the wine type is red and the grape varietal is cabernet sauvignon.

Top 10 Regions for Red Wines - Cabernet Sauvignon

Based on Avg Rating for 2007-2009

Appellation	Vintage_2	Ratings (Avg)	Retail Price (Avg)	Vineyard Count
Santa Cruz Mountains	2009	94	$59	1
Napa Valley	2007	94	$109	119
Napa Valley	2008	93	$133	87
Napa Valley	2005	93	$114	90
Napa Valley	2006	93	$108	85
Napa Valley	2009	93	$130	72
Sonoma County	2005	92	$55	18
North Coast	2008	92	$16	1
Central Coast	2006	92	$32	1
Other California	2008	91	$57	4

Figure 2 Results of a OLAP Cube Query

Create the Code

You can quickly create the SQL code for extracting OLAP cube data by copying a sample of the code.

1. Open the OLAP Cube in SAS Enterprise Guide. This allows you to navigate through the cube, generate the tedious MDX code for that specific view, and limit all of your typing.

   Note: Arrange the cube view to match the dimension/hierarchy that you want for the finished report.

2. Select the **Edit View -> Edit in MDX Viewer** from the menu to see the MDX code.

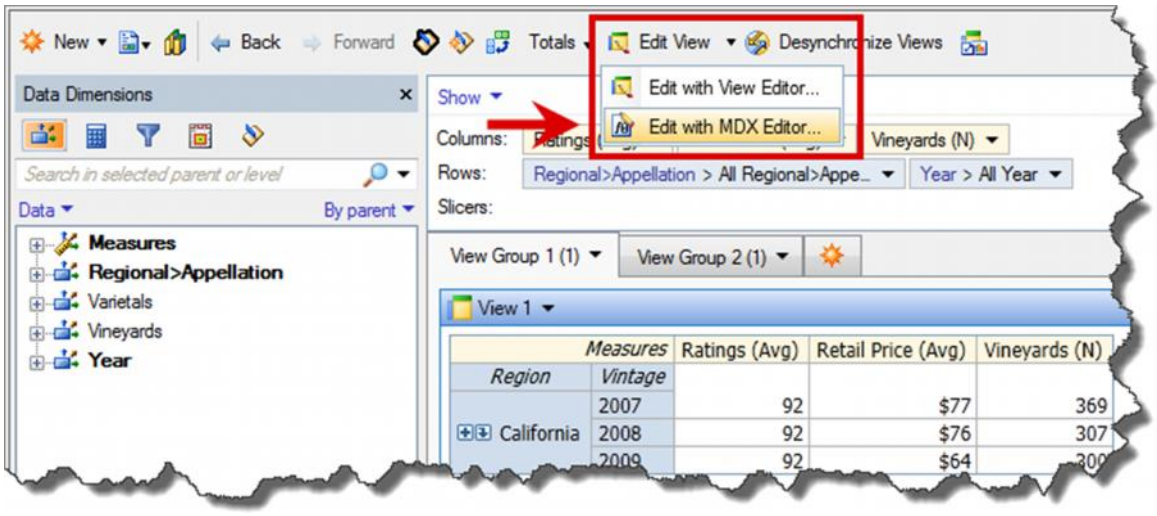

Figure 3 OLAP Cube Desired View

3. You can view all of the MDX code associated with this cube view. To have a prompt with the data, add a WHERE statement, highlight the dimension and dragging it into the WHERE statement, which would be similar to the following figure.

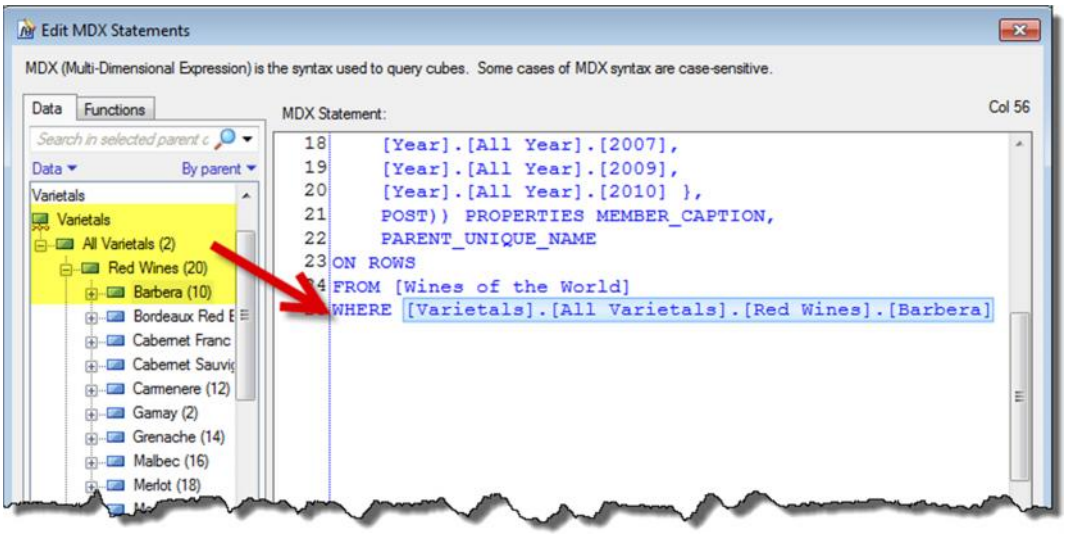

Figure 4. MDX Editor

4. Verify your code works and then copy this code to use in your next step.
5. Start a new program to use with you stored process. Your program will be similar to this code:

Code	Notes

`proc sql;`
`connect to olap (host="server-name" port=5451`
`  user="username"  pass="pwd");`
 	
`CREATE TABLE olap2stp as`
`  select * from connection to olap`
`(`
	Connect to the OLAP cube using the SAS Access engine.

`  SELECT {`
`  [Measures].[Ratings (Avg)],`
`  [Measures].[Retail Price (Avg)],`
`  [Measures].[Vineyard Count] } PROPERTIES`
`  MEMBER_CAPTION, PARENT_UNIQUE_NAME`
`  ON COLUMNS, CrossJoin({`
`  [Regional>Appellation].[All`
`  Regional>Appellation].[California] },`
`  Hierarchize({`
`  [Year].[All Year].[2007],`
`  [Year].[All Year].[2008],`
`  [Year].[All Year].[2009] }, POST))`
`  PROPERTIES MEMBER_CAPTION, PARENT_UNIQUE_NAME`
`  ON ROWS`
`  FROM [Wines of the World]`
	Paste the MDX code from the SAS Enterprise Guide into the stored process.

`  WHERE [Varietals].[All Varietals].` `  [&WineTypePrompt.].[&VarietalPrompt.])`	<*Optional*> Use a WHERE clause to build cascading filters. Add the macro variables in the appropriate locations in the code, as shown. Then set up your prompts in the stored process.
`);` `quit;`	

Register and Test the Stored Process

Register, add prompts, and test your process. In Figure 5, you can see an example of the cascading prompt. The Varietal prompt is a dependent prompt based on what values the user chooses for wine type. For instance, chardonnay is a white wine grape type, so if the user selects White Wines then the second prompt shows only white wine types.

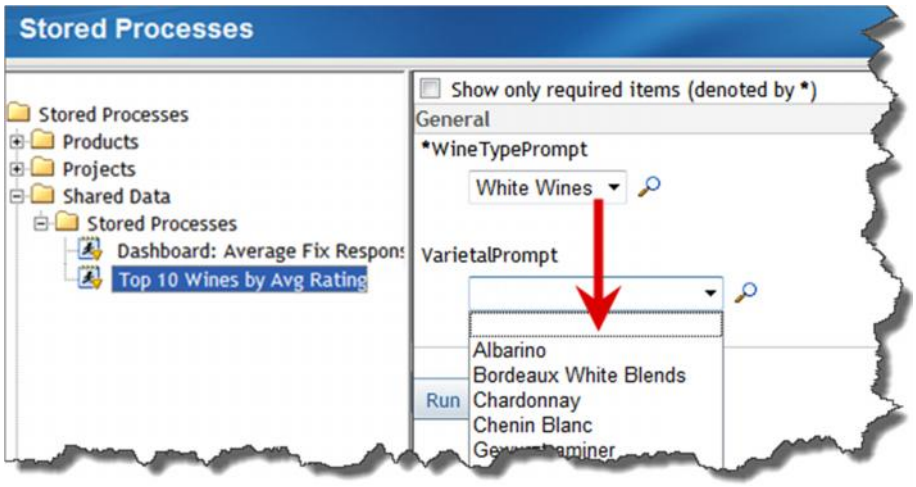

Figure 5. Create a cascading prompt from an OLAP Cube

USE HTML AND JAVASCRIPT TO BUILD YOUR OWN PROMPTS

If you want to add an interactive map to your output, you can do it easily with HTML and JavaScript. This stored process uses PROC GMAP to create a heat map based on data from the Wine API data set. To use the stored process, the user selects a variable from a drop-down box, which immediately updates the chart. Using JavaScript, you can implement this type code quickly.

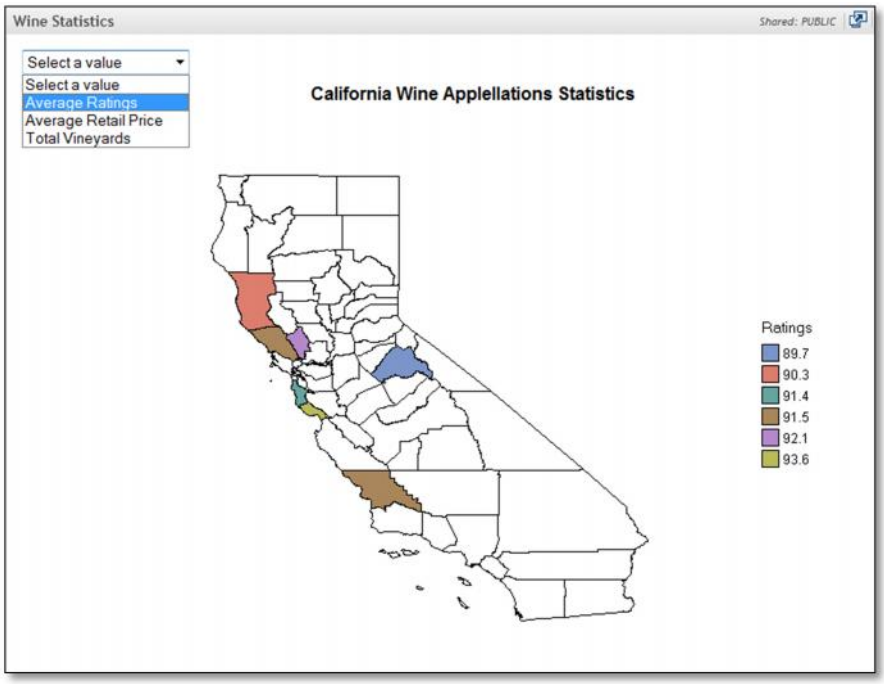

Figure 6 Using a drop-down box with a SAS graph

When a SAS procedure is in a stored process, the %STPBEGIN/%STPEND macros ensure that the output displays properly. Using HTML code in a stored process ensures that the output goes to the _WEBOUT file location.

HTML code requires a set of tags to start a Web page (<HTML>, <BODY>) and another set of tags to end the page (</BODY>, </HTML). Both %STPBEGIN/%STPEND macros and ODS HTML statements write this HTML code for you when creating output. To create the example above, you must control the creation of the HTML starting and ending tags so that you can use the JavaScript and SAS procedure together.

Adding JavaScript to a Stored Process

The following code creates the stored process shown in Figure 6. The first time the stored process runs, it uses a default value to display the chart. In subsequent applications, the user can select a value from the drop-down menu and the stored process is updated immediately. Add the JavaScript code to submit the stored process when the user makes a selection from the drop-down menu.

Code	Notes
`%global VarPrompt;`	Create a global macro variable for the default value and user selection. The drop-down menu on the top left of the page uses the prodprompt macro variable.
`goptions device=actximg CBACK=WHITE;`	Define the graphics device.

`ods html`
`  body=_webout(no_bottom_matter)`
`  style=sasweb`
	Open the _WEBOUT location using an ODS HTML statement. Use the no_bottom_matter option to write the file to the stream without writing

Code	Notes
`path=&_tmpcat (url=&_replay);` `ods html close;`	the closing HTML tags.

`%macro view;`
`data _null_;`
`file _webout;`
	Create the VIEW macro to control what displays on the page. Use FILE to output to the _webout location.

`put '<script type="text/javascript"`
`language="JavaScript">';`
`put ' function UpdateChart() {';`
`put ' document.DoubleOut.submit();';`
`put ' }';`
`put ' </script>';`
	The JavaScript UpdateChart() function automatically submits the DoubleOut form therefore no Run or Submit button is needed.

`put " <FORM NAME='DoubleOut'`
`ACTION='&_URL' method='post'`
`enctype='multipart/form-data'>";`
	Add the code to create the drop-down menu. The form is named DoubleOut, which is used by JavaScript code when the form is re-submitted.

`put "<INPUT TYPE='HIDDEN' NAME='_program'`
`VALUE='&_PROGRAM'>";`
`put '<table border="0" cellpadding="5">';`	

`put '<TR> <TD valign=TOP>`
`<SELECT Name="VarPrompt"`
`onChange="UpdateChart();">';`
	Create the prodprompt drop-down menu and call the UpdateChart() JavaScript function when the user makes a different selection.

`put '<option value="">Select a value`
`</option>';`
`put '<OPTION VALUE="RATE"';`
`%if "&VarPrompt"="RATE"`
`%then put " SELECTED";`
`put '>Average Ratings</OPTION>';`
`put '<OPTION VALUE="PRICE"';`
`%if "&VarPrompt"="PRICE"`
`%then put " SELECTED";`
`put '>Average Retail Price</OPTION>';`
`put '<OPTION VALUE="VINEYARD"';`
`%if "&VarPrompt"="VINEYARD"`
`%then put " SELECTED";`
`put '>Total Vineyards</OPTION>';`
`put '</SELECT></TD>';`
	Add an option for each product value. This code replaces a prompt.
	If the &VarPrompt macro value was previously selected, the drop-down menu maintains that selection by adding SELECTED to the OPTION tag.

`%if %length(&VarPrompt) = 0 %then`
`%do;`
`%let VarPrompt=RATE;`
`%end;`
`run;`
	If this is the initial run or if &VarPrompt is empty, use RATE as the default value.

`ods html`
`body=_webout(no_top_matter`
`no_bottom_matter)`
`path=&_tmpcat (url=&_replay);`
	Use the ODS HTML statement to open the _WEBOUT stream for the PROC statements.
	Use the no_top_matter and no_bottom_matter options to prevent the system from writing HTML starting or ending tags to the_WEBOUT stream.

`title "California Wine Appellations`
`Statistics";`
`PROC GMAP GOUT=MAPCHART`
`DATA=Wine.WineMap`
`MAP=MAPSGFK.US_COUNTIES;`
`where State = 6;`
`ID CONT ID;`
`CHORO &VarPrompt / WOUTLINE=1 ;`
`RUN; QUIT;`
	Add a TITLE statement and use the prodprompt macro variable so that the user knows which product is showing
	The macro variable determines what variable is used for the map.

`ods html close;`	Close the ODS output statement.

Code	Notes

`data _null_;`
`  file _webout;`
`  put '</TR>';`
`  put '  </table>';`
`  put '</FORM>';`
`  put '</BODY>';`
`  put '</HTML>';`
`  run;`
`%mend view;`
`%view;`
	Close the _WEBOUT stream with the HTML closing tags.
	
	Run the %view macro.

Note: When you register the stored process, make sure the %STPBEGIN/%STPEND marcos are turned off.

USING STORED PROCESSES FROM INFORMATION MAPS

There are so many uses for stored processes within information maps. The problem for many developers however is that the setup resembles a typical information map layout because the metadata table definitions are permanently stored. Questions developers have include: What keeps the stored process flexible enough to handle multiple and concurrent users? How is the data removed/replaced on subsequent requests? The trick is redirecting the data output to a WORK location.

libname *libref* (work);

The following example is excerpted from *The 50 Keys to Learning SAS Stored Processes* book shows how to create a stored process that allows you to pull data based on the year.

Step 1 Setup the Metadata

Of course, you can manually define a data table in SAS Management Console but importing an existing table is much easier. Create a data table with 0 records to represents the stored process output and the information map input.

1. Create a temporary folder c:\sas\data\tempout. This entire folder structure will be deleted later.
2. From SAS Enterprise Guide or BASE SAS, run the following code to create a sample dataset. The WHERE statement must generate an empty table. This is useful for systems with limited data space because only the metadata is needed for subsequent steps.

```
libname tempout "c:\sas\data\tempout";
data tempout.salesdetail;
 set booksamp.salesdetail2011(where=(date="01oct74"d));
run;
```

3. From SAS Management Console (or SAS Enterprise Guide's Update Library Tool):
   a) Register a BASE SAS library called TempOut that points to the c:\sas\data\tempout folder.
   b) Register the SalesDetail dataset.
4. Delete SalesDetail.sas7bdat from the c:\sas\data\tempout folder.

Step 2 Create the stored process

Develop a stored process that performs the task at hand. The important aspect of this stored process, independent of what task you wish it to undertake is that it generates a data table named the same as the registered data table SalesDetail. In the stored process below, we are choosing which detail table to retrieve based on a date range prompt and then returning the desired data records.

IMAP Dynamic Table	Notes
`%stpbegin;`	
`libname tempout (work);`	Create a library reference called tempout that is redirecting to the temporary WORK file folder. You must use the exact LIBREF name as the metadata library that you created in the previous step 1.

IMAP Dynamic Table	Notes

`libname booksamp meta library="STP Book Sample Data" metaout=data;`
`data _null_;`
`year1=substr("&date_range_min", 6, 4);`
`year2=substr("&date_range_max", 6, 4);`
`call symput('start', year1);`
`call symput('end', year2);`
`year3=year(today());`
`call symput('current', strip(year3));`
`run;`
`%macro join;`
	Assign the metadata library where the detail source data is located.
	Create two temporary macros (Start and End) that only contain the 4-digit year (for example, 2012). When the stored process runs, these macro variables come from a date_range prompt.
	Create a macro for the current year to reduce the maintenance of the do loop in the next step.

`data tempout.SalesDetail;`	Create the SalesDetail table in the WORK folder.

`Set`
`%do i=2008 %to &current;`
`%if &i >= &start and &i <= &end`
`%then booksamp.salesdetail&i ;`
`%end;`
`; run;`
	Loop through all years available (in this case 2008 to the current year) and set the necessary data sets.

`%mend;`
`%join;`
`%stpend;`
	Stop the data step, end the join macro, run the JOIN macro and close the stored process.

Step 3 Register the stored process

When you register the stored process, ensure that the **Result capabilities**: checkboxes are empty and add a date range prompt for the DATE_RANGE macro referenced in the stored process code.

Step 4 Create the information map

In SAS Information Map Studio, create a new information map using the metadata table that you defined in step 1.

After a table is created in the information map, you can add the stored process.

5. Select **Tables** ❶ from the Show menu.
6. Navigate to the table location in TEMPout library and add the SALESDETAIL table❷.
7. Now change the Show: selection box to Stored processes ❶
8. Select the stored process name registered in Step 3 above.

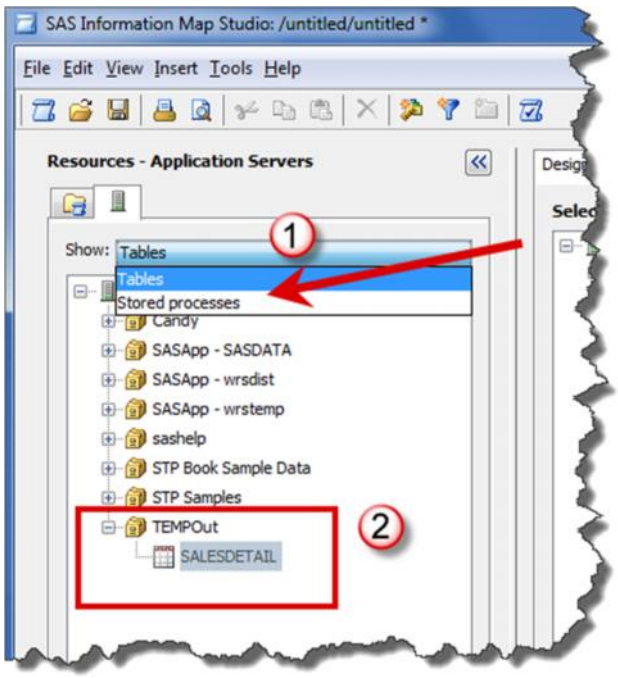

Figure 7 Add the stored process

Step 5 Review the Log

When testing the information map, use the **Show Server Log** to review the code. (Note that the View SQL button is misleading, as it shows the tempout permanent location and then the SQL code – when in fact the Server Log shows that the tempout location is pointing to WORK.)

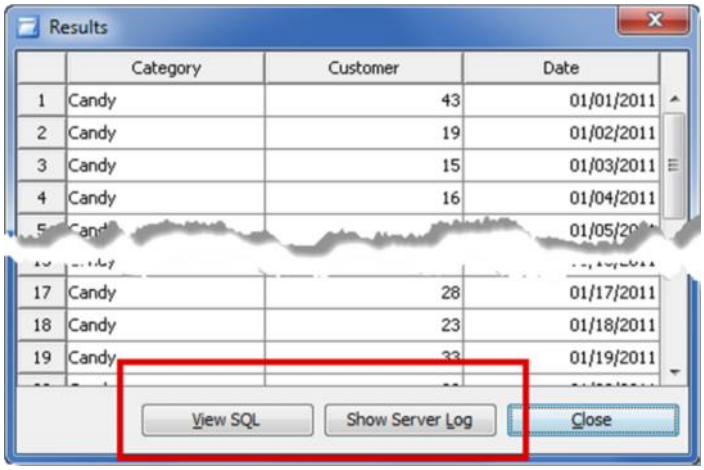

Figure 8. View the Server Log

Server Log	Notes
2 LIBNAME tempout BASE C:\SAS\Data\TempOut"; NOTE: Libref TEMPOUT was successfully assigned as follows: Engine: BASE Physical Name: C:\SAS\Data\TempOut	Notice that first the information map defines the TEMPOUT library using the metadata definition.
4 /* v2 (9.3) stored process support */ 5 PROC STP program='/User Folders/Angela Hall/IMAPDynamicTableTEST (StoredProcess)'; 6 inputParam date_range='March 01, 2007 -- November 25, 2009'; 7 run; NOTE: PROC_STP: Proc STP Execution Starting ===== NOTE: PROC_STP: = Stored Process:/User Folders/Angela Hall/ IMAPDynamicTableTEST (StoredProcess) ======	In SAS 9.3, all stored processes are called using PROC STP.
NOTE: %INCLUDE (level 1) file c:\SAS\Stps93\IMAPDynamicTable.sas is file c:\SAS\Stps93\IMAPDynamicTable.sas. 2 +%stpbegin;	The Stored Process raw code is then submitted.
3 +libname tempout (work); NOTE: Libref TEMPOUT was successfully assigned as follows: Physical Name(1): **C:\Users\sassrv\AppData\Local\Temp\SAS Temporary Files_TD7992_L73453_\Prc9**	TEMPOUT is reassigned to a work location. The remaining steps produce the tempout.salesdetail table in that work location.

```
5   +libname booksamp meta
6   +  library="STP Book Sample Data"
7   +  metaout=data;
NOTE: Libref BOOKSAMP was successfully assigned as follows:
      Engine:        META
      Physical Name: C:\SAS\Data\STPSamples
8   +data _null_;
9   +   year1=substr("&date_range_min", 6, 4);
10  +   year2=substr("&date_range_max", 6, 4);
11  +   call symput('start', year1);
12  +   call symput('end', year2);
13  +   year3=year(today());
14  +   call symput('current', strip(year3));
```

Server Log	Notes
15 +run; NOTE: DATA statement used (Total process time): 16 +%macro join; 17 +data tempout.salesdetail;	
18 +set 19 + %do i=2008 %to &current; 20 + %if &i >= &start and &i <= &end 21 + %then booksamp.salesdetail&i ; 22 + %end; ; 23 +run;	
24 +%mend; 25 +%join; NOTE: There were **3036** observations read from the data set BOOKSAMP.SALESDETAIL2008. NOTE: There were **2968** observations read from the data set BOOKSAMP.SALESDETAIL2009. NOTE: The data set TEMPOUT.SALESDETAIL has 6004 observations and 17 variables. +%stpend;	

CONCLUSION

SAS stored processes offer a lot of flexibility to BI content developers. You can use the stored process to add the extra functionality that may not be readily available in the other SAS BI tools.

REFERENCES

1. The GKPI Procedure, SAS/Graph 9.3 Reference Manual, Second Edition. SAS Institute.
2. Aanderud & Hall, *The 50 Keys to Learning SAS Stored Processes*, Siamese Publishing, Raleigh, NC, April 2012.

RECOMMENDED READING

- Aanderud & Hall, *Building Business Intelligence with SAS*: *Content Development Examples*, SAS Press, Cary, NC, February 2012.
- SAS 9.3 Stored Processes Developer's Guide, SAS Institute.
- Business Intelligence Notes for SAS BI Users blog, *http://www.bi-notes.com*
- Real BI for Real Users, *http://blogs.sas.com/content/bi*

CONTACT INFORMATION

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

Tricia Aanderud, tricia.aanderud@and-data.com
Angela Hall, angela.hall@sas.com