Paper BB-02

SAS© Stored Processes: Swiss Army Knife of the Business Intelligence Toolset

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 you first learn to use the SAS Business Intelligence toolset, you might realize some obvious ways to use the stored processes. As you advance your skills, you will find situations where you need one of the tools to do something just a little different or a little more. In this paper, you will learn some common ways to use the stored processes and some advanced ways.

COMMON USES

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 charts or tables to the Information Delivery Portal using the Stored Process Portlet or the URL Display Portlet.
- 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.
- Create a custom display for BI Dashboard to provide a non-supported indicator. For instance, you can create a geographical map.
- Leverage a stored process from an information map to reduce rework of existing SQL code into the 'Relationship' tab.
- Add a prompt group to an information map by using the stored process to hold the prompts.
- Use a stored process to create a highly customized report that you can link to from BI Dashboard or Web Report Studio.
- Enable additional functionality from OLAP based Web Report Studio reports, such as passing values into a subsequent prompted report.
- Develop custom applications for user interaction via chained stored processes.
- Reduce the impact of multiple queries/drill-downs on the same data source by utilizing session control in a stored process.
- Generate a custom graphic element for display in a Web Report or Add-in to MS Office when the available tasks are not licensed or unavailable.
- Submit queries to run in the background and email results at a later time.
- Output packaged results for permanent or temporary storage and retrieval.
- Allow users to choose what output device to print the results out to, whether an MS Excel file, PDF, RTF, etc.

WEB REPORT STUDIO: ADDING A DASHBOARD DIAL

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 has a sample of the different gadgets you can add. In addition, notice – there is not any data (cube, info map) being used – it's all stored processes.

Figure 1. Adding a KPI Dial to Web Report Studio

WRITE THE CODE

This example explains how to create the Avg Fix Response Time speedometer shown in Figure 1. The code was based on the examples from the GKPI Procedure, see reference 1.

Code	Comments
`/*Replace LET statement with code to create the needed values*/` `%let YOUR_PCT=.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;`	For this output to work properly, the graphic device must be set to Java Image. You can use a macro variable to set this value.
`%STPBEGIN;` `goptions reset=all` `xpixels=210 ypixels=200;` `proc gkpi mode=raised;` `speedometer actual= &YOUR_PCT.` `bounds=(0 .40 .60 1)` `/target=.85` `lfont=(f="Albany AMT" height=.5cm)` `label="Avg Fix Response Time"` `format="percent8.0";` `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 (%STPBEGIN/ %STPEND) in the SAS Code pane. Your stored process will not work as expected if the macro variables are not assigned outside of the macros. Notice the &YOUR_PCT macro variable is used for the actual calculation.

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

When an OLAP Cube is used as the data source, some BI client tools disable functionality, such as cascading prompts and linking to other prompted reports. You can quickly generate the PROC SQL code and parameterize for use in a prompted stored process report by leveraging SAS Enterprise Guide.

CREATE THE CODE

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

1. Open and slice 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: The slice should match the dimension/hierarchy that you will prompt on in the report.
2. Select the **Edit View -> Edit in MDX Viewer** from the menu to see the MDX code.

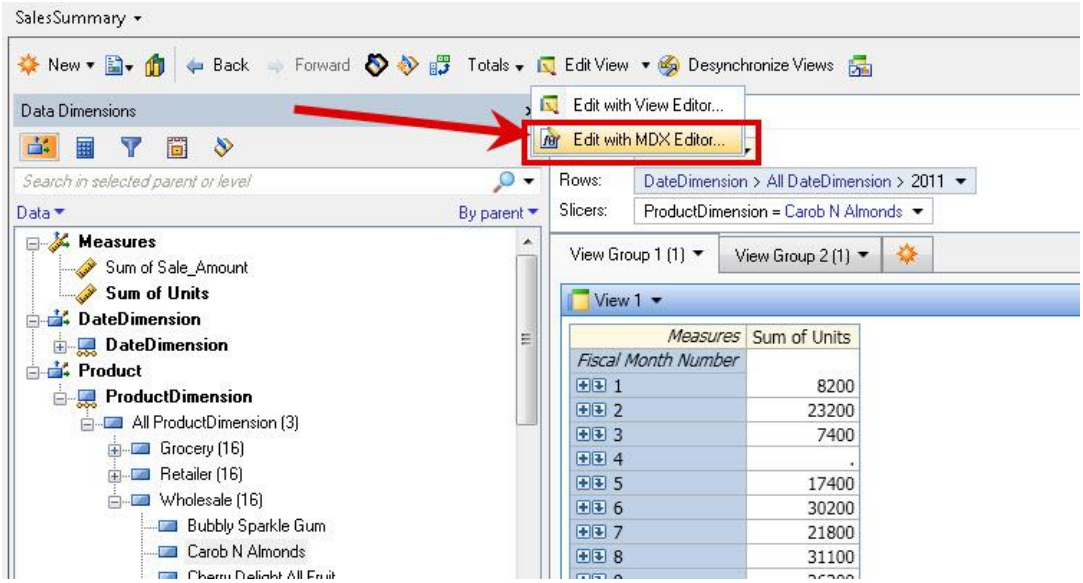


Figure 2. OLAP Cube Slice

3. Now you can view all of the MDX code associated with this view. Copy this entire code for the next step.

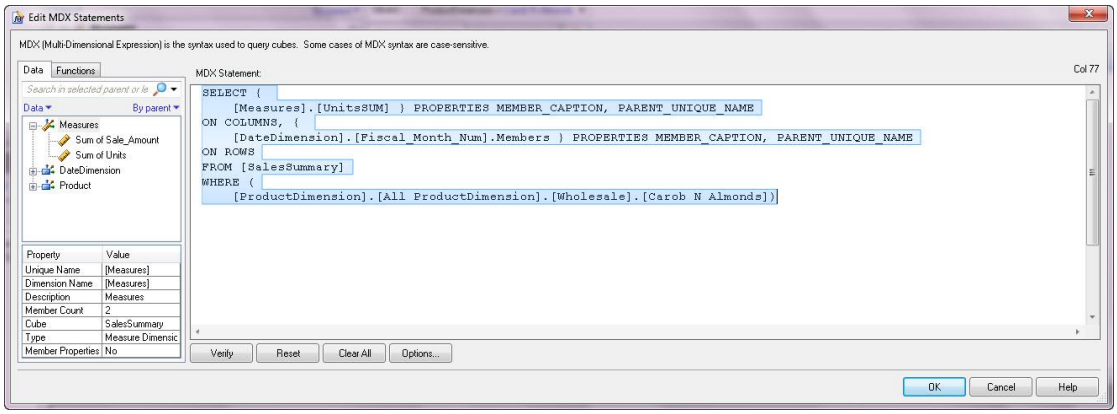


Figure 3. MDX Editor

4. Place the MDX code in a PROC SQL statement as shown.

Code	Notes
`proc sql;` `connect to olap (host="machine" port=5451` `  user="username"  pass="pwd");` `CREATE TABLE stp_datasource as` `  select * from connection to olap`	Connect to the OLAP cube.
`(` `  SELECT {[Measures].[UnitsSUM] } PROPERTIES` `  MEMBER_CAPTION, PARENT_UNIQUE_NAME` `  ON COLUMNS, {` `  [DateDimension].[Fiscal_Month_Num].Members` `  }` `  PROPERTIES MEMBER_CAPTION,` `  PARENT_UNIQUE_NAME` `  ON ROWS FROM [SalesSummary]` `  WHERE ([ProductDimension].[All` `  ProductDimension].[Wholesale].[Carob N` `  Almonds])`	Paste the MDX code here
`  WHERE ([ProductDimension].[All` `  ProductDimension].[&type].[&product])` `);`	*<optional>* Use a WHERE clause to build cascading filters. Add the macro variables in the appropriate locations in the code, as shown.
`quit;`	

REGISTER AND TEST THE STORED PROCESS

Register and test your process. In Figure 4 you can see an example of the cascading prompt.

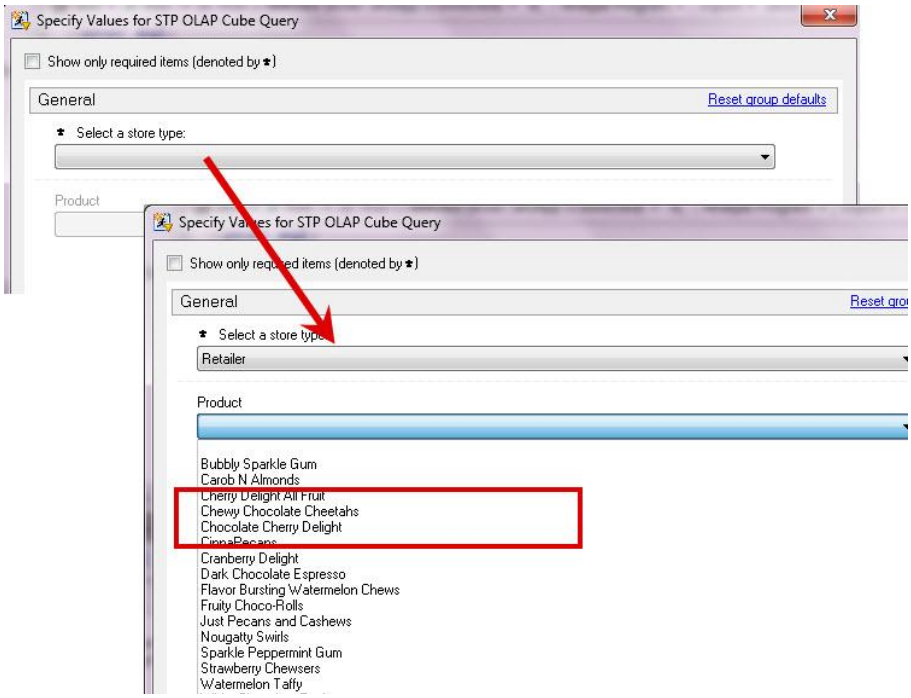


Figure 4. Create a cascading prompt from an OLAP Cube

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 SASHELP.DEMOGRAPHICS 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.

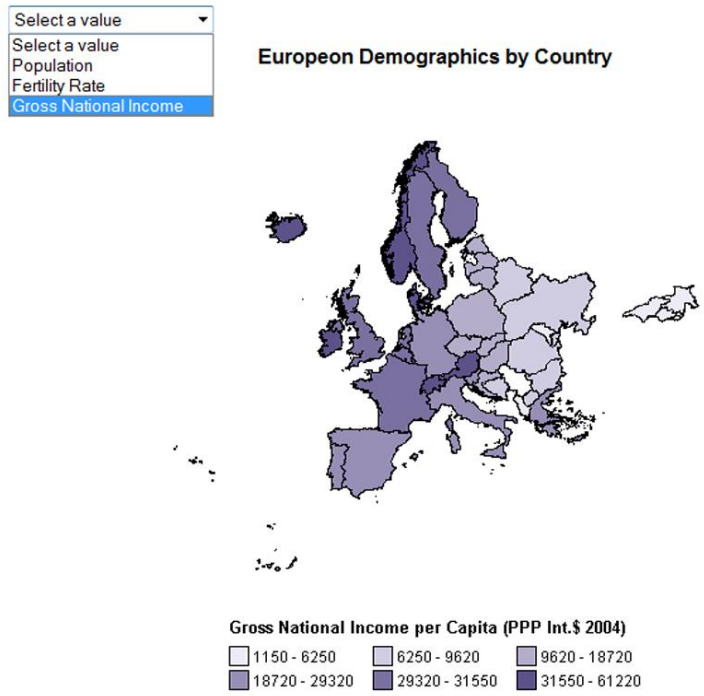


Figure 5. Using a drop-down box with a SAS graph

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 5. 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` `  path=&_tmpcat (url=&_replay);` `ods html close;`	Open the _WEBOUT location using an ODS HTML statement. Use the no_bottom_matter option to write the file to the stream without writing the closing HTML tags.

Code	Notes
`%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 so the JavaScript document submit function above works as expected.
`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="pop"';` `  %if "&VarPrompt"="pop"` `  %then put " SELECTED";` `put '>Population</OPTION>';` `put '<OPTION VALUE="totalfr"';` `  %if "&VarPrompt"="totalfr"` `  %then put " SELECTED";` `put '>Fertility Rate</OPTION>';` `put '<OPTION VALUE="gni"';` `  %if "&VarPrompt"="gni"` `  %then put " SELECTED";` `put '>Gross National Income</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=pop;` `  %end;` `run;`	If this is the initial run or if &VarPrompt is empty, use pop 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. Also note that ods html close is NOT included here, this keeps the html output area open for other PROCS to send output into.
`Title "Europeon Demographics by Country";` `PROC GMAP GOUT=MAPCHART` `  DATA=SASHELP.DEMOGRAPHICS` `  MAP=MAPS.EUROPE;` `where region = 'EUR';` `ID CONT ID;` `CHORO &VarPrompt / WOUTLINE=1  ;`	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.

Code	Notes
RUN; QUIT;	
ods html close;	Close the ODS output statement.
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. Some examples of possible stored process use within Information Maps:

- Leveraging explicit pass-thru SQL when connecting to RDBMS
- Querying an OLAP Cube to allow for prompting (pre 9.2)
- Utilizing existing SQL queries rather than going through the Information Map Relationship tab builder
- Taking prompted values into account to refine the resulting database query

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);

That's right – what can fix your headache from thinking through this is just one line of code. Let's walk through the four steps. First, set up the metadata for this stored process created data table then develop and register the stored process and finally define the information map.

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. What I do is create a data table with 0 records that represents the output of the stored process as well as the input into the information map.

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.
`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 structure.
`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

Register this stored process by following these steps.

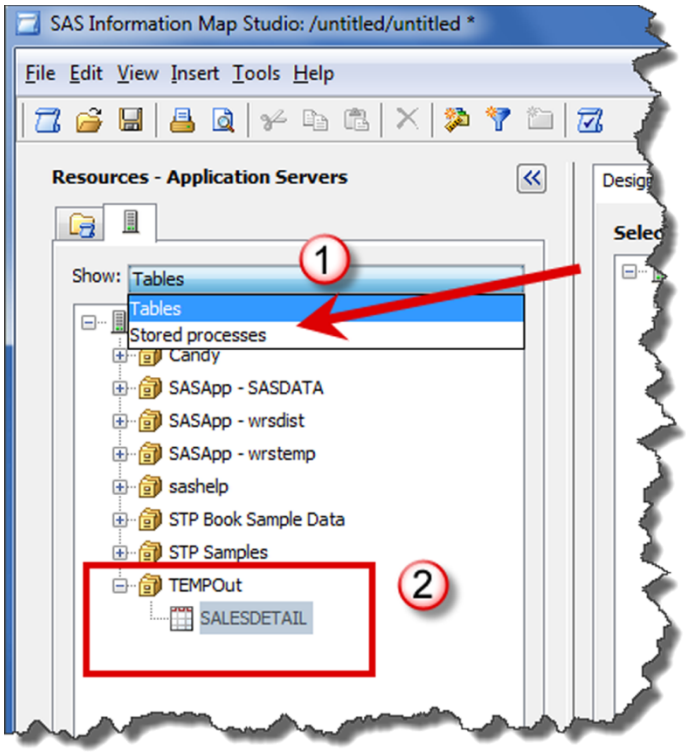
1) In the Execution tab, ensure that the **Result capabilities**: checkboxes are not checked.
2) Add a date range prompt for the DATE_RANGE macro referenced in the stored process code.

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.

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



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.)

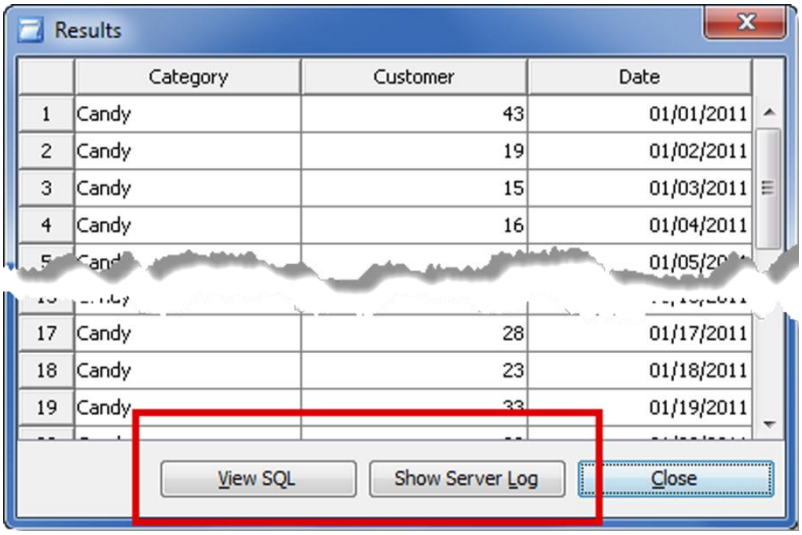


Figure 6. View the Server Log

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.

Server Log	Notes
4 /* v2 (9.3) stored process support */ 5 PROC STP program='/User Folders/Angela Hall/My Folder/07_a_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/My Folder/07_a_IMAPDynamicTableTEST(StoredProcess) ======	In SAS 9.3, all stored processes are called using PROC STP.
NOTE: %INCLUDE (level 1) file c:\SAS\Stps93\TEST_Example_7a_IMAPDynamicTable.sas is file c:\SAS\Stps93\TEST_Example_7a_IMAPDynamicTable.sas. 2 +%stpbegin; 3 +	The Stored Process raw code is then submitted.
4 +libname tempout (work); NOTE: Libref TEMPOUT was successfully assigned as follows: Levels: 1 Engine(1): V9 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  +
14  +   year3=year(today());
15  +   call symput('current', strip(year3));
16  +run;
NOTE: Numeric values have been converted to character values at the places given by:
(Line):(Column).
      15:31
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

17  +%macro join;
18  +data tempout.salesdetail;
19  +
20  +set
21  + %do i=2008 %to &current;
22  +   %if &i >= &start and &i <= &end
23  +   %then booksamp.salesdetail&i ;
24  + %end;
25  +;
                                                                          The
SAS System
```

Server Log	Notes

```
26  +
27  +run;
28  +%mend;
29  +
30  +%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.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
31  +%stpend;
```

If you still do not believe me, open a BASE SAS session to open the tempout.salesdetail data table in the initial folder location where you created the 0 record file. When I tested it myself, I received the following note, which in this instance made me quite pleased.

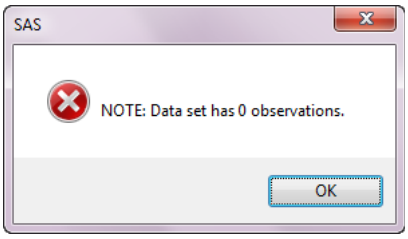


Figure 7. Result

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