SAS6361-2016

Stored Processes and SAS® Visual Analytics: Giving Users the Power to Load

Renato Luppi and Varsha Chawla, SAS Institute Inc.

ABSTRACT

A stored process is a SAS® program that can be executed as required by different applications. Stored processes have been making SAS users' lives easier for decades. In SAS® Visual Analytics, stored processes can be used to enhance the user experience, create custom functionality and output, and expand the usefulness of reports.

This paper will discuss a technique on how data can be loaded on demand into memory for SAS Visual Analytics and accessed by reports as needed using stored processes. Loading tables into memory so that they can be used to create explorations or reports is a common task in SAS Visual Analytics. This task is usually done by an administrator, enabling the SAS Visual Analytics user to have a seamless transition from data to report. At times, however, users will have a need for tables to be partially loaded or modified in memory on demand. The step-by-step instructions in this paper make it easy enough for any SAS Visual Analytics report builder to include these stored processes in their work. By using this technique, SAS Visual Analytics users gain the freedom to access their data without having to work with an administrator for every little task, helping everyone be more effective and productive.

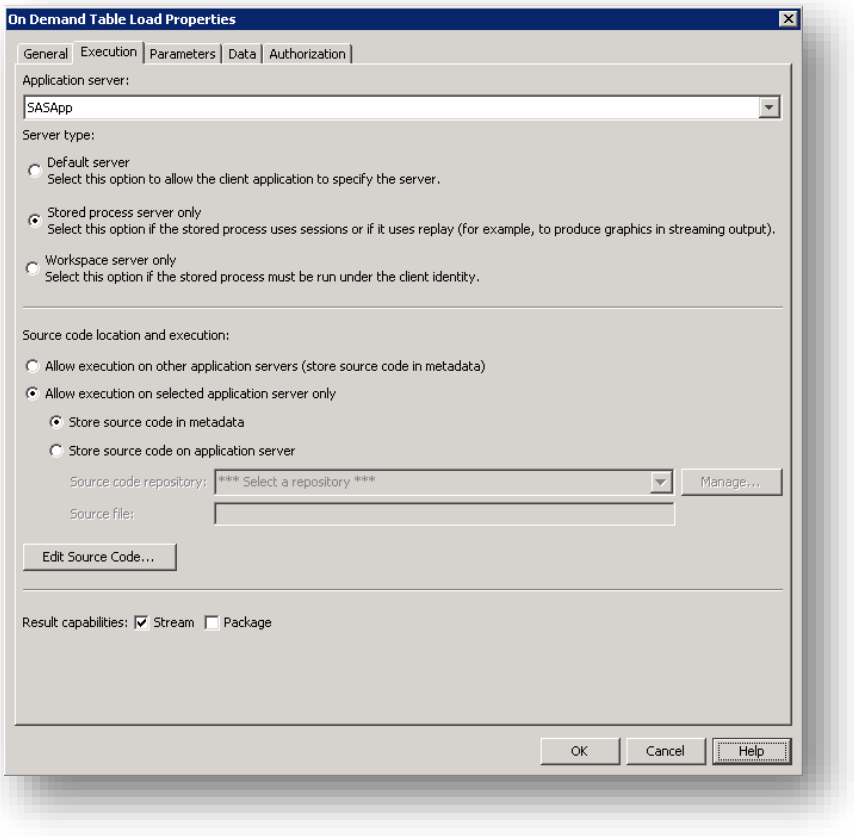
INTRODUCTION

Stored processes are generally used by administrators and end users alike. Those with a little bit of exposure to stored processes as well as SAS Visual Analytics will benefit the most from this paper. We will start by reviewing what stored processes are and showing you how to add them to a SAS Visual Analytics report. Then, we walk through two specific examples, both of which are embedded stored processes in SAS Visual Analytics that produce a form of output. Finally, we move to four even more powerful examples. In these examples, the user does not just view output, but interacts and modifies data using stored processes. The four core examples are as follows:

1. A stored process that loads a table on demand, if the table has not already been loaded.
2. A stored process that loads specific records on demand. These records are specific to a user and the user's permissions in SAS Visual Analytics. The user will only be able to view his or her own records.
3. A stored process that adds, deletes, and updates records that are loaded in memory and used in a SAS Visual Analytics report.
4. A stored process that calculates moving averages and Bollinger bands. (See http://www.bollingerbands.com/.) Each user can view the calculations and Bollinger bands according to their own parameters.

STORED PROCESSES – WHAT ARE THEY AGAIN?

At its core, stored processes are simple, re-usable SAS programs that are registered in metadata via SAS Management Console® or by using SAS code. You can create a stored process in SAS® Enterprise Guide® as well. You simply type your code and then use a wizard to create a stored process out of it. The properties of a stored process can be explored in SAS Management Console as well, as shown in Display 1.



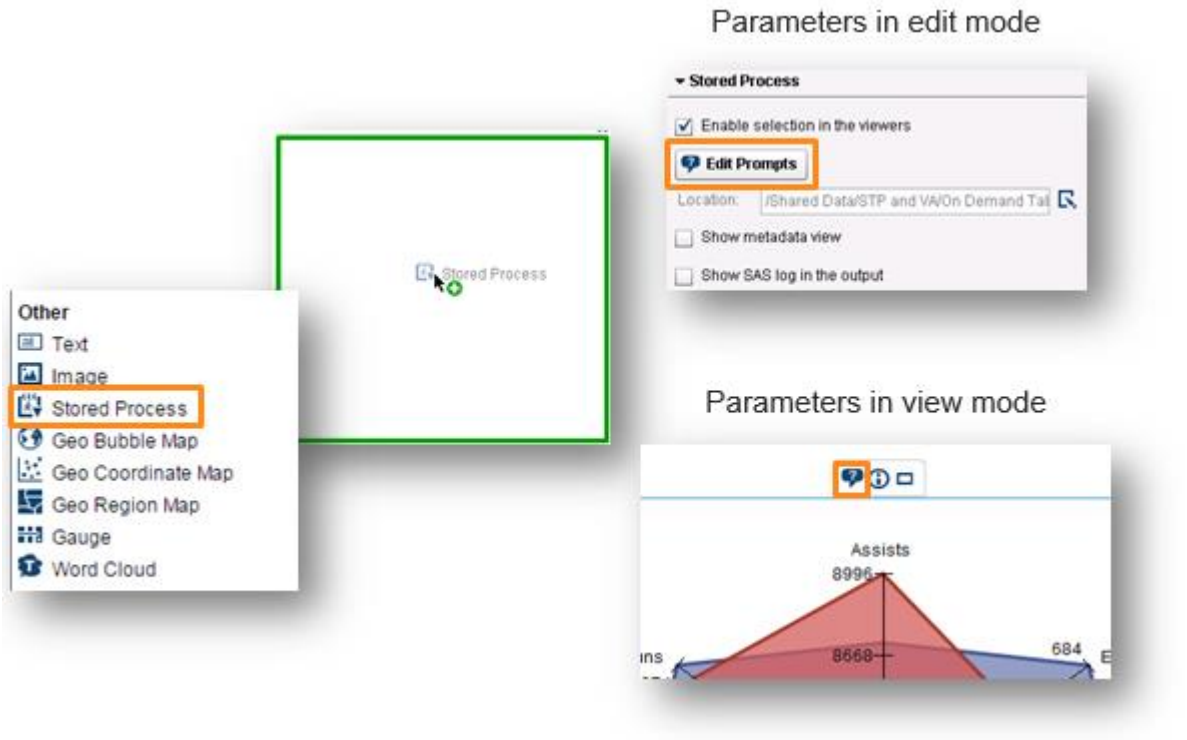
Display 1. Stored Process Properties in SAS Management Console

Stored processes can be executed on the workspace server, which runs under the client's identity, or on the stored process server, which runs under a shared account. Although all stored processes run in the background, users can interact with stored processes as well in several ways. Stored processes can accept parameters, which can be entered by the user or passed in from somewhere else. What do you get after you run a stored process? One of two possibilities: As mentioned, a stored process can run in the background and manipulate data. In this case, there is no visible output to the user. Results are not sent back to the client, and the program simply executes without the user necessarily knowing it runs. In the second case, you do get viewable results, either in streamed or package format.

STORED PROCESSES IN SAS VISUAL ANALYTICS

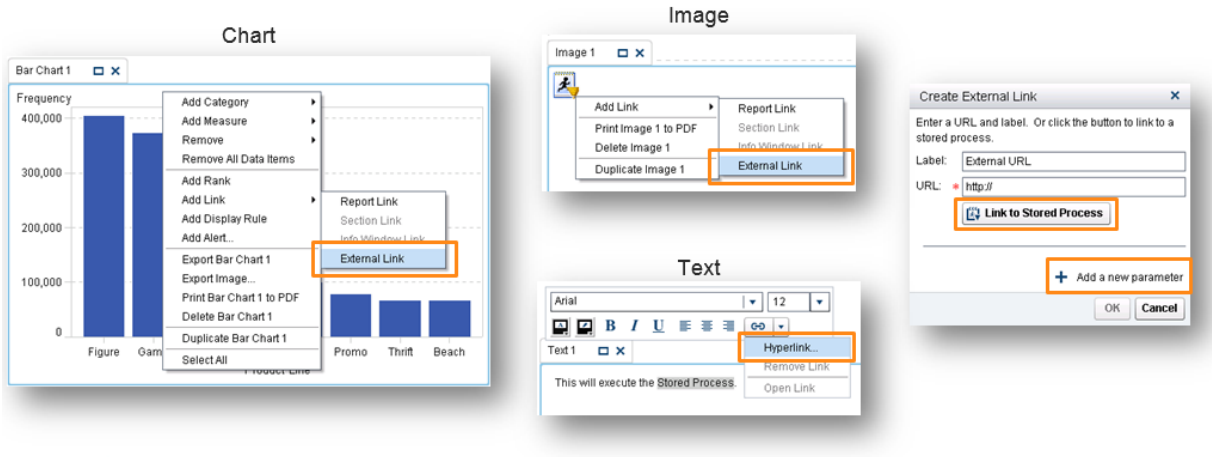
You can use stored processes to extend SAS Visual Analytics capabilities. For example, in the context of reports, you can use stored processes to produce charts or perform specific calculations that you cannot find in SAS Visual Analytics Designer. There are two ways you can add a stored process to a report in the

SAS Visual Analytics Designer. The first way is by dragging the Stored Process report object and embedding it into the report. Users can enter and pass parameters to stored processes via prompt interface, which is available in edit and view modes. This is shown in Display 2 below. Stored processes added to SAS Visual Analytics reports via this method are executed whenever the reports are opened, refreshed, or user changes the stored process parameters.



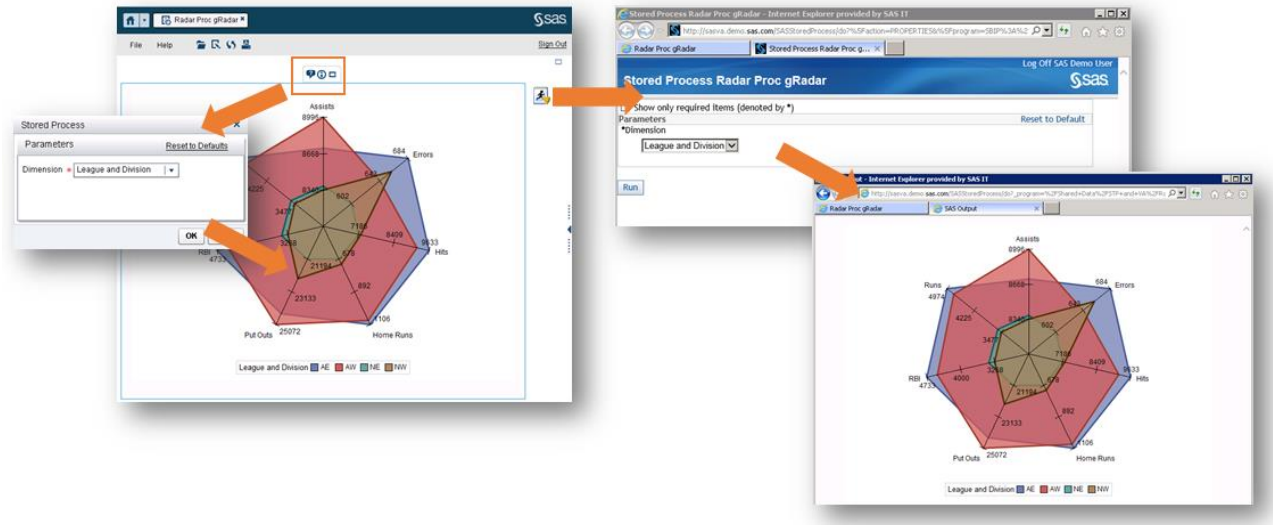
Display 2. Stored Process as Embedded Object in SAS Visual Analytics Report

The second way to use a stored process in SAS Visual Analytics is to add it as an external link. Parameters can be passed from other report objects to the stored process as part of the external link. Note that external links can be created from report objects such as charts, tables, images, and texts, as shown in Display 3 below.



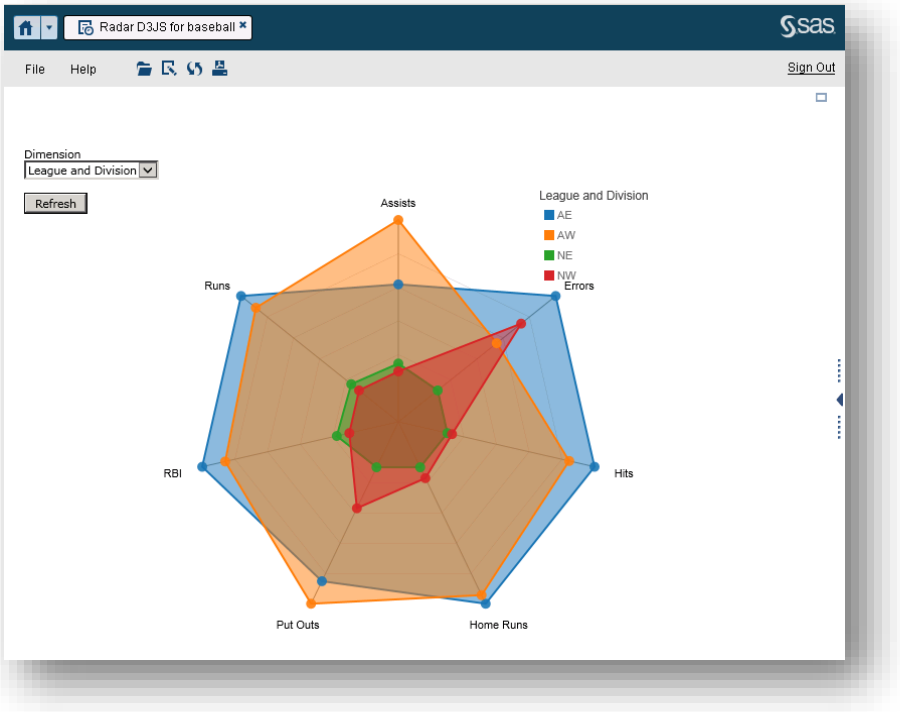
Display 3. Stored Processes as External Links from Report Objects

In our first example, we use a stored process to create HTML output via ODS (Output Delivery System). We use simple SAS code and PROC GRADAR to create a radar chart. Radar charts, also known as spider or star charts, are useful for displaying multivariate data from a common starting point. By visually being able to analyze the data points and lines that connect the data points, we can quickly compare numbers across different categories. Since radar charts are not in SAS Visual Analytics, a user might choose to create one using PROC GRADAR. (See http://support.sas.com/documentation/cdl/en/graphref/67881/HTML/default/viewer.htm#n0hhznjklmhf66n14ry392qrxs68.htm.) The chart could be displayed as either embedded output in SAS Visual Analytics or as an external link (in its own browser window). The two types of output, as well as their respective prompt interface, are displayed below in Display 4.



Display 4. Embedded and Externally Linked Radar Charts Created Using PROC GRADAR

Our second example is similar to the first one. But in this case, we are taking advantage of D3 JavaScript (https://d3js.org/) code to create a radar chart for us. We then embed the output into SAS Visual Analytics using a stored process. Customizing the radar chart is done by tweaking the JavaScript and HTML code that is provided to us from the D3JS site. The output looks as shown in Display 5 below.



Display 5. Radar Chart Embedded in SAS Visual Analytics, Created Using D3

Being able to add external output to SAS Visual Analytics reports can be extremely useful for organizations that need certain visualizations in particular, or have a need to pull in existing content into new SAS Visual Analytics reports.

ADVANCED STORED PROCESSES IN SAS VISUAL ANALYTICS

Up to this point, we have focused on using stored processes to create visual output in SAS Visual Analytics. We are now taking this one step further by sharing some examples of how stored processes can manipulate data and provide powerful capabilities to the user through SAS Visual Analytics. Here are a few guidelines for the upcoming examples:

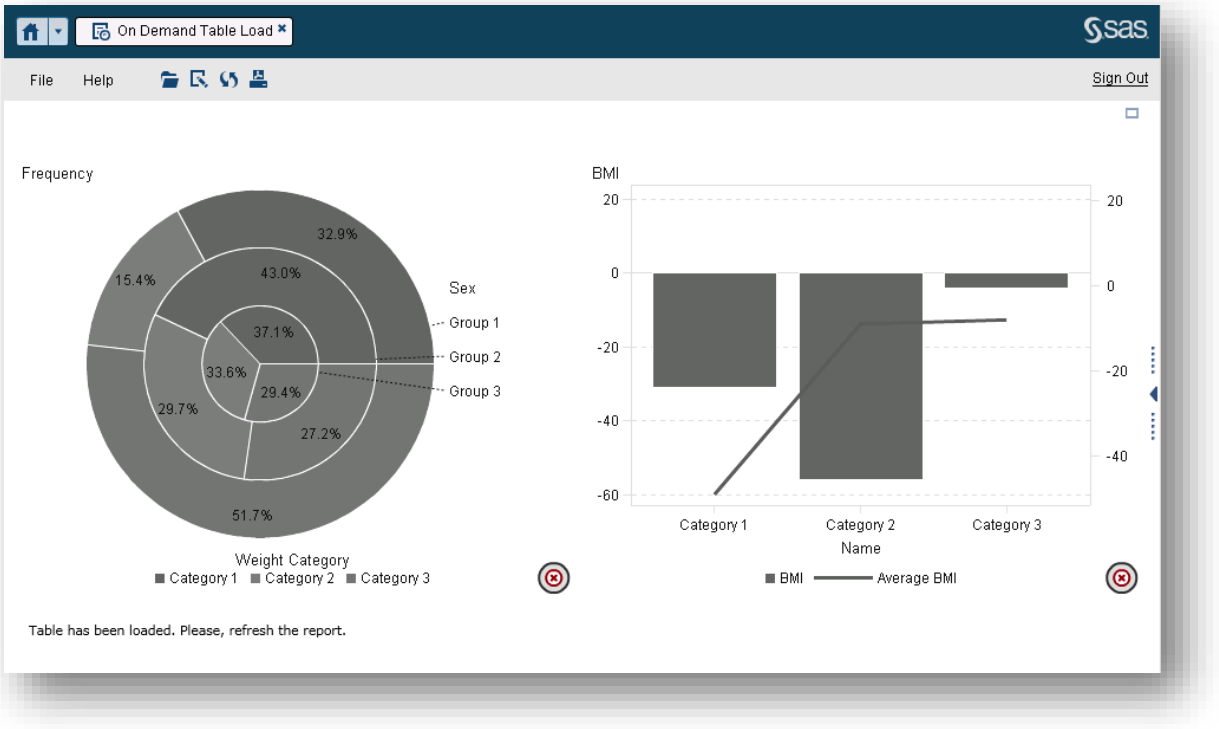
- All stored processes are embedded in SAS Visual Analytics 7.3 reports as report objects.
- All stored processes are configured to be executed in the Stored Process server, but could run on the Workspace server as well.
- Most of the stored processes produce streamed HTML output. This streamed output consists of update messages and/or parameter input forms, not graphical output.

EXAMPLE 1 – ON DEMAND TABLE LOAD

Although you can load tables in SAS Visual Analytics via either the Administrator component or the Import function in the Data Preparation component, this capability is usually limited to the SAS Visual Analytics administrator. What if you are a report consumer and want to view a report, but don't have the data table associated with the report loaded in memory yet? Wouldn't it be great if you could have the data table needed automatically loaded while the corresponding report is open in SAS Visual Analytics?

This would be a huge time saver since you could access the report without having to jump into another SAS Visual Analytics component or ask an administrator to load the table for you. On demand table load through the SAS Visual Analytics Report Viewer interface is exactly the purpose of this first example. Here's how it works in more detail:

1. The report consumer opens a report and notices that the data is "missing." In other words, it has not yet been loaded into the SAS® LASR™ Analytic Server. When data is missing in SAS Visual Analytics, the report objects are shown as gray "skeletons," and we have a red X icon for each report object. Display 6 shows what the report looks like.



Display 6. Report without Data Loaded into SAS LASR Analytics Server

2. At the time the report is open, the stored process that is embedded in the report executes and detects the table is not loaded yet. As a result, the stored process loads the table in memory. The user still sees the report as in Display 6, but most likely the table was already loaded at that time. That happens because the report rendering and the stored process execution run asynchronously.
3. The report consumer now simply refreshes the report to view the data. Notice that the stored process is re-executed every time the report is refreshed or opened. But because the table gets loaded when the report is open for the first time, the stored process doesn't do anything during subsequent accesses to the report, no matter if it's the same user who is opening or refreshing the report or someone else. Of course, the stored process does this without the user knowing what's happening in the background. Although this is not all the code that the stored processes consist of, these are the key components that check for the data table and load it into the LASR Analytics Server:

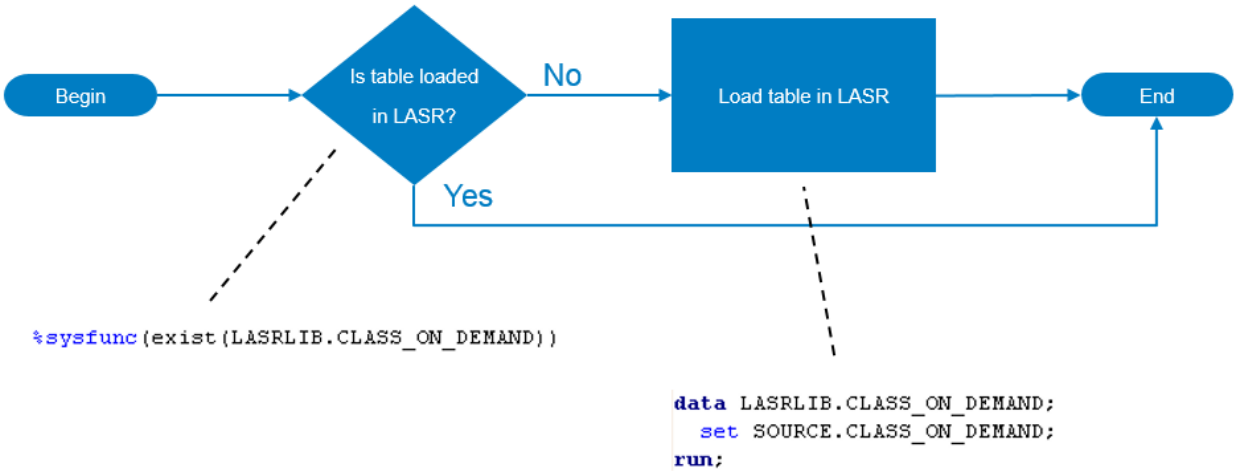
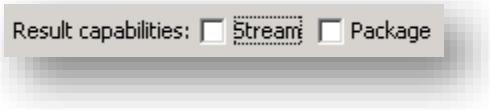


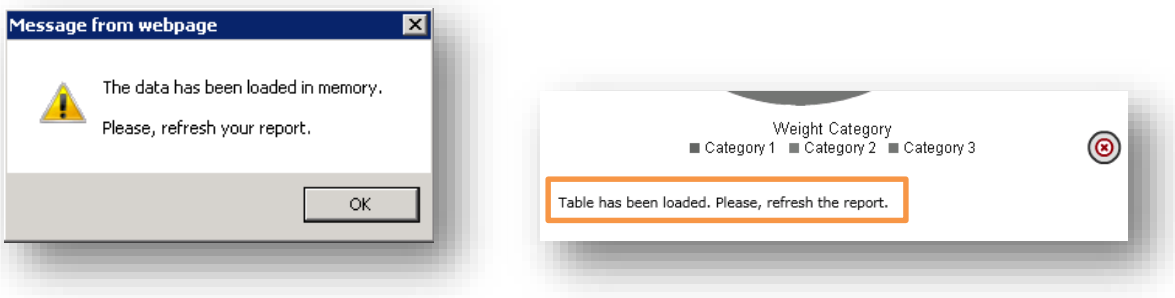
Figure 1. Basic SAS Code Used for the Stored Process in Example 1

Because this stored process does not have visible output in the client, it must be registered in metadata (SAS Management Console) with the result capabilities unchecked, as shown in Display 7 below:



Display 7. Results Capabilities Are Unchecked for the Stored Process

Once the data table has been loaded in memory, it might be useful to let the report consumer know about this since the stored process runs undetected behind the scenes. There are multiple ways that you can use a stored process to inform the report consumer about what is happening. We selected two easy options to work with: a plain text message that is added to the report as a footnote and a pop-up message. Both are shown in Display 8 below.



Display 8. Pop-up Message and Plain Text Message

Here is the SAS code needed to send back a pop-up message like the one in Display 8. Note that we created a macro variable called “&popup” that controls whether the pop-up window is displayed. The pop-up message is only displayed the first time the report runs, when the table has not been loaded in

memory yet. If it has already been loaded in memory, the stored process returns an empty HTML document since the stored process is still expected to stream something back.

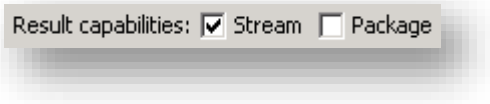
```
data _null_;
  file _webout;
  put "<HTML>";
  put "<BODY>";
  if "&POPUP" eq "true" then do;
    put "<SCRIPT>";
    put "alert('&MSG')";
    put "</SCRIPT>";
  end;
  put "</BODY>";
  put "</HTML>";
run;
```

Because the pop-up message is not embedded in the report, the stored process object can be added anywhere in report and the object can have any size.

If you choose to communicate back via a plain text message to be embedded in the report, you could use the SAS code below, where the message is set in the “&msg” macro variable.

```
data _null_;
  format infile $char256.;
  input;
  infile = resolve(_infile_);
  file _webout;
  put infile;
cards4;
<HTML>
<BODY style="font-family:verdana;font-size:8pt">

&msg

</BODY>
</HTML>
;;;;
run;
```

Be sure to check the “Stream” box for result capabilities when you create the stored process. This will ensure that the message can be displayed to the user. Note that the full code for the stored process is available in the appendix of this paper.



Display 9: Stream Box Is Checked for the Stored Process that Sends Back a Message

EXAMPLE 2 – ON DEMAND RECORD(S) LOAD

The on demand record(s) load example is similar to the previous example. However, instead of the entire table being loaded, only those records that a report consumer is supposed to be able to see will be loaded in memory. This is very useful when the report consumer only needs to see his or her data (a subset of a data table), especially if the data is only queried from the table, without aggregations taking place. Some examples for this use case include salaries, student scores, profile information, and other personal data. Loading records independently only when they are necessary can also help optimize memory utilization.

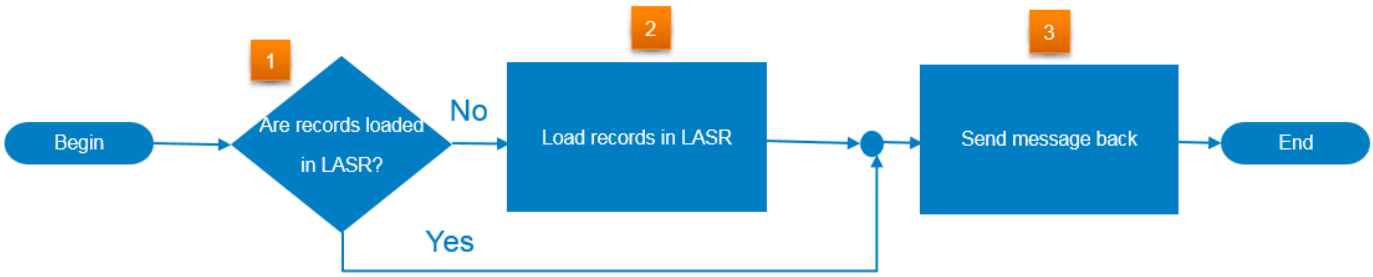


Figure 2 –Process Flow of the Stored Process Used in Example 2

At a high level, when the report is open, the stored process checks if the user's records have been loaded or not. If the records were already loaded, the stored process sends back a message informing the user when the records were loaded and the user can view the report. If the records were not yet in memory, the stored process loads them and sends a message, telling the user to refresh the report. Once the report is refreshed, the user can view it.

This is the code that checks if the records were already loaded in memory:

1

```
* Assume table/record has NOT been loaded yet: set the flag for loading it;
%let execute=1;

%if %sysfunc(exist(LASRLIB.ON_DEMAND_RECORD_LOAD)) %then %do;
  * The table has already been loaded.;

  * Check if the record is already loaded.;
  data _null_;
    set LASRLIB.ON_DEMAND_RECORD_LOAD;
    * this portion of the data step only executes if there is data already
      loaded for the current user (rules given by row-level security);

    * Record is already loaded: set the flag for doing nothing;
    call symput('execute','0');
    stop;
  run;
%end;
```

Here is the block of code that loads the records into the LASR Analytics Server. This code should only be executed if the macro variable “&execute” is ‘1’. Note that “&_metaperson” is an automatic macro variable that refers to the person running the stored process - which in this case, is the report consumer, not the person who created the stored process. For more information about stored processes reserved macro variables, please see https://support.sas.com/documentation/cdl/en/stpug/68399/HTML/default/viewer.htm#p184mqqbi9w6qjn1q0619x19eg02.htm.

2

```
data work.ON_DEMAND_RECORD_LOAD;
  set source.ON_DEMAND_RECORD_LOAD
 (where=(upcase(Employee_Name)=upcase("&_metaperson")));
run;
```

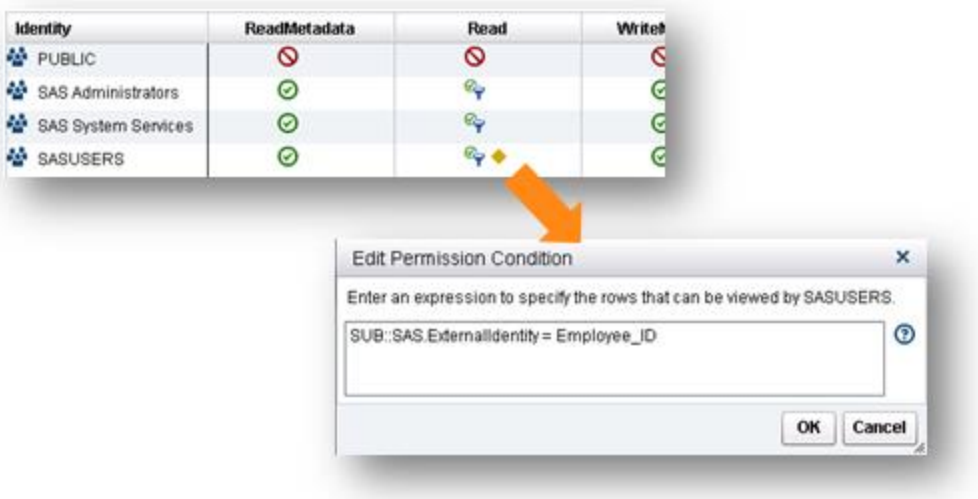
```
data LASRLIB.ON_DEMAND_RECORD_LOAD ( append );
  set WORK.ON_DEMAND_RECORD_LOAD;
run;
```

Finally, here is the code that sends a message back to the user. The macro variable “&msg” should be set appropriately. For example, it could state that the records have been loaded and then prompt the user to refresh the report. Note that this message could also inform the user that records have failed to load (in case of error), or were already in memory. The stored process designer can decide the appropriate message to display.

3

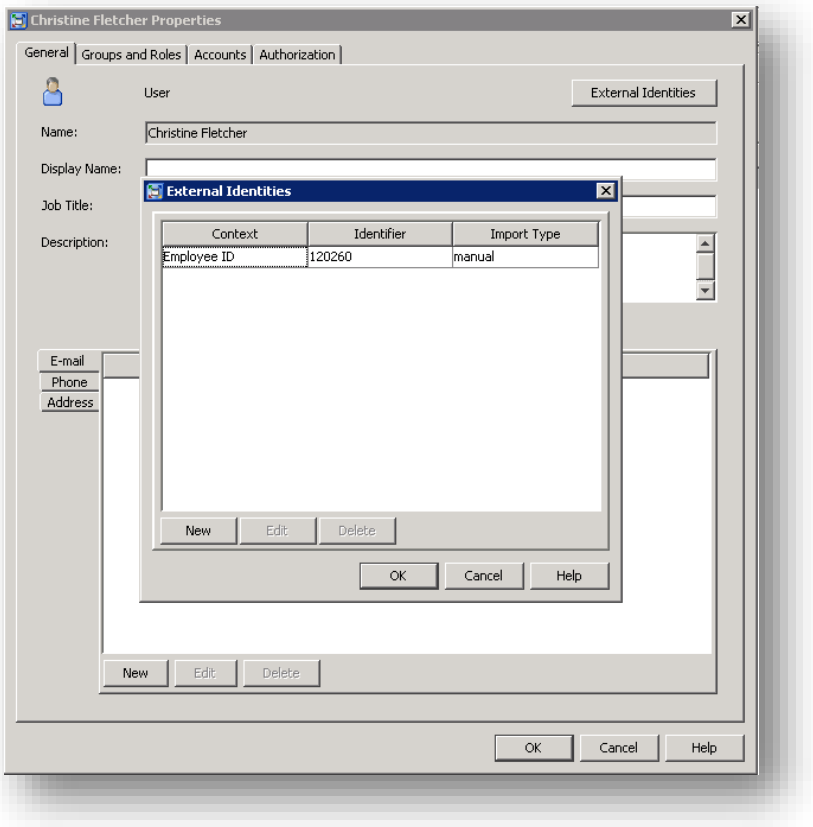
```
data _null_;
  format infile $char256.;
  input;
  infile = resolve(_infile_);
  file _webout;
  put infile;
cards4;
<HTML>
<BODY style="font-family:verdana;font-size:8pt">

&msg

</BODY>
</HTML>
;;;;
run;
```

At this point, you might be wondering how records that are loaded for one user are not visible to another, and vice versa. SAS Visual Analytics has row-level security capability that can be applied to a LASR table. Applying row-level security (http://support.sas.com/documentation/solutions/va/73/en/vaag.pdf) is done in the administrator component of SAS Visual Analytics. You can specify different conditions to set permissions. Here is what we did to set permissions for each user in SAS Visual Analytics:



Display 10. Identity-driven Property in Permission Condition

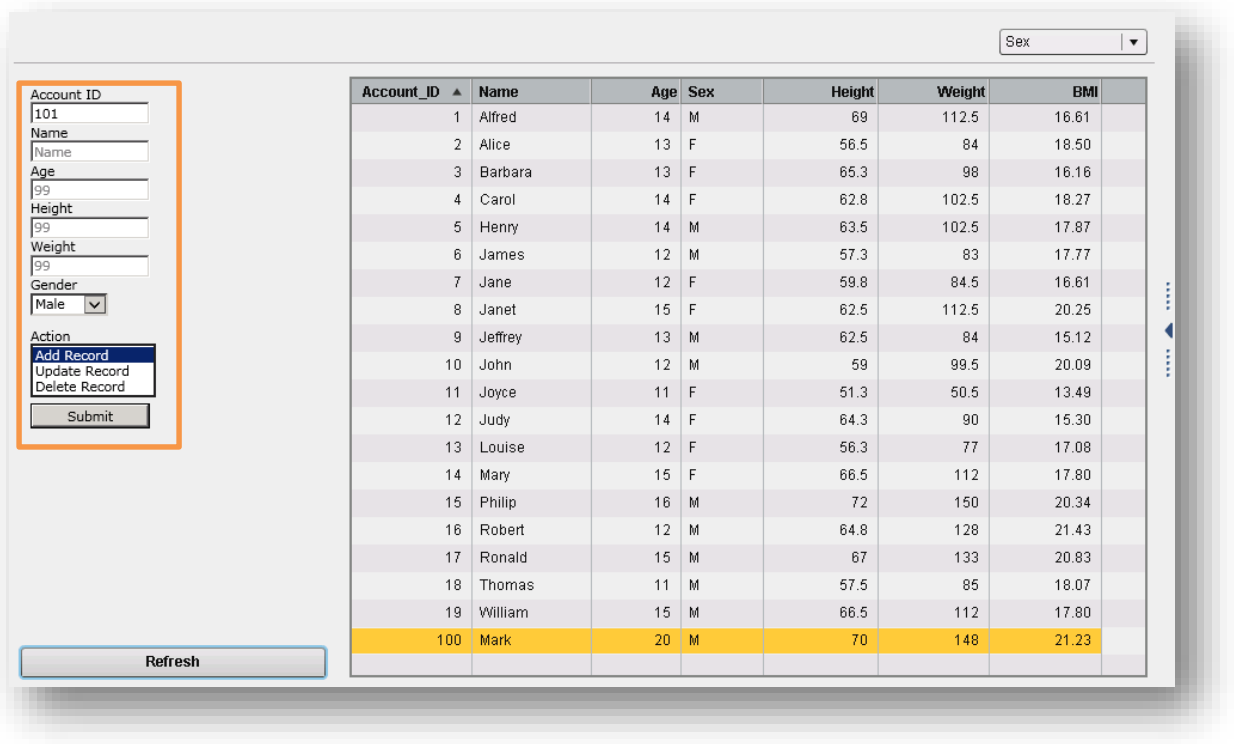
The external identify itself can be managed in SAS Management Console, as shown in Display 11, or bulk loaded in batch (see http://support.sas.com/documentation/cdl/en/bisecag/67045/HTML/default/viewer.htm#n0l2hp5m00a1z2n1b598q4pknfih.htm.)



Display 11. External Identity in SAS Management Console

EXAMPLE 3 – DATA ENTRY

In this example, we allow the user to add, delete, and update records that are loaded into memory. Those are the basic operations of a data entry application. Being able to manipulate data in SAS Visual Analytics is a huge benefit for when little tweaks or modifications are needed and the business users are allowed to make these. The report consumer saves much more time compared to having to go to an administrator to ask and wait for these changes. Note that the table stored on disk (not on the LASR Analytic Server) is not updated in this example. In order to keep tables in sync and information saved, it should be updated as well. The user interacts with an HTML form that the stored process generates. Those parameters are used by the stored process to make changes to the table. Row-level security is not needed in this example, because all users have access to all records in the table. We have also included feedback to the user. The user gets informed every time the submit button in the HTML form is clicked. In this way, the user knows when the stored process is ready to accept new changes to the data, when to wait until the data is ready because the stored process is still executing, and when to refresh the report after the execution is finished. This is because the label of the submit button is modified and changes to "Wait" while the stored process is executing. A screenshot of what the user interface looks like is shown in Display 12 below. The stored process HTML form is highlighted on the left.



Account_ID	Name	Age	Sex	Height	Weight	BMI
1	Alfred	14	M	69	112.5	16.61
2	Alice	13	F	56.5	84	18.50
3	Barbara	13	F	65.3	98	16.16
4	Carol	14	F	62.8	102.5	18.27
5	Henry	14	M	63.5	102.5	17.87
6	James	12	M	57.3	83	17.77
7	Jane	12	F	59.8	84.5	16.61
8	Janet	15	F	62.5	112.5	20.25
9	Jeffrey	13	M	62.5	84	15.12
10	John	12	M	59	99.5	20.09
11	Joyce	11	F	51.3	50.5	13.49
12	Judy	14	F	64.3	90	15.30
13	Louise	12	F	56.3	77	17.08
14	Mary	15	F	66.5	112	17.80
15	Philip	16	M	72	150	20.34
16	Robert	12	M	64.8	128	21.43
17	Ronald	15	M	67	133	20.83
18	Thomas	11	M	57.5	85	18.07
19	William	15	M	66.5	112	17.80
100	Mark	20	M	70	148	21.23

Display 12. Stored Process User Interface as HTML Form in SAS Visual Analytics

At a high level, the data entry stored process code can be represented with the flow chart in Figure 3.

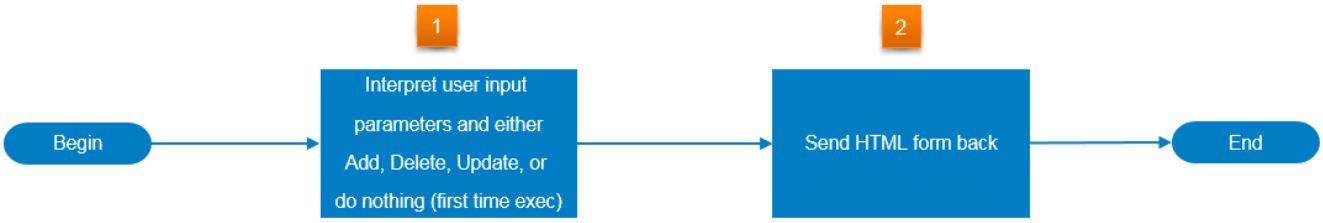


Figure 3. Flowchart of Data Entry Stored Process

Let's examine the first block of code, which performs the action to the table. It first takes care of the add portion if the user chooses to add data to the table. The remaining else-if statements respectively delete and update records from the table using PROC IMSTAT, which was specifically designed to manage in-memory tables in LASR. (See http://support.sas.com/documentation/cdl/en/inmsref/68736/HTML/default/viewer.htm#n0b077sznqbykqn195rut54vcwks.htm.)

1

```
%if &_action=Add %then %do;
    data Class_TMP;
      Account_ID=&Account_ID;
      Name="&Name";
      Sex="&sex";
      Age=&age;
      height=&height;
      Weight=&Weight;
    run;

    data LASRLIB.&out_table ( append );
      set Class_TMP;
    run;
%end;

%else %if &_action=Delete %then %do;
    proc imstat data=LASRLIB.&out_table;
      where Account_ID=&Account_ID;
      deleterows / purge;
    run;
    quit;
%end;
```

```
%else %if &_action=Update %then %do;
    data Class_TMP;
      _where_="Account_ID=&Account_ID";
      Name="&name";
      Sex="&sex";
      Age=&age;
      Height=&height;
      Weight=&weight;
    run;

    proc imstat data=LASRLIB.&out_table;
      update data=work.Class_TMP;
    run;
    quit;
%end;
```

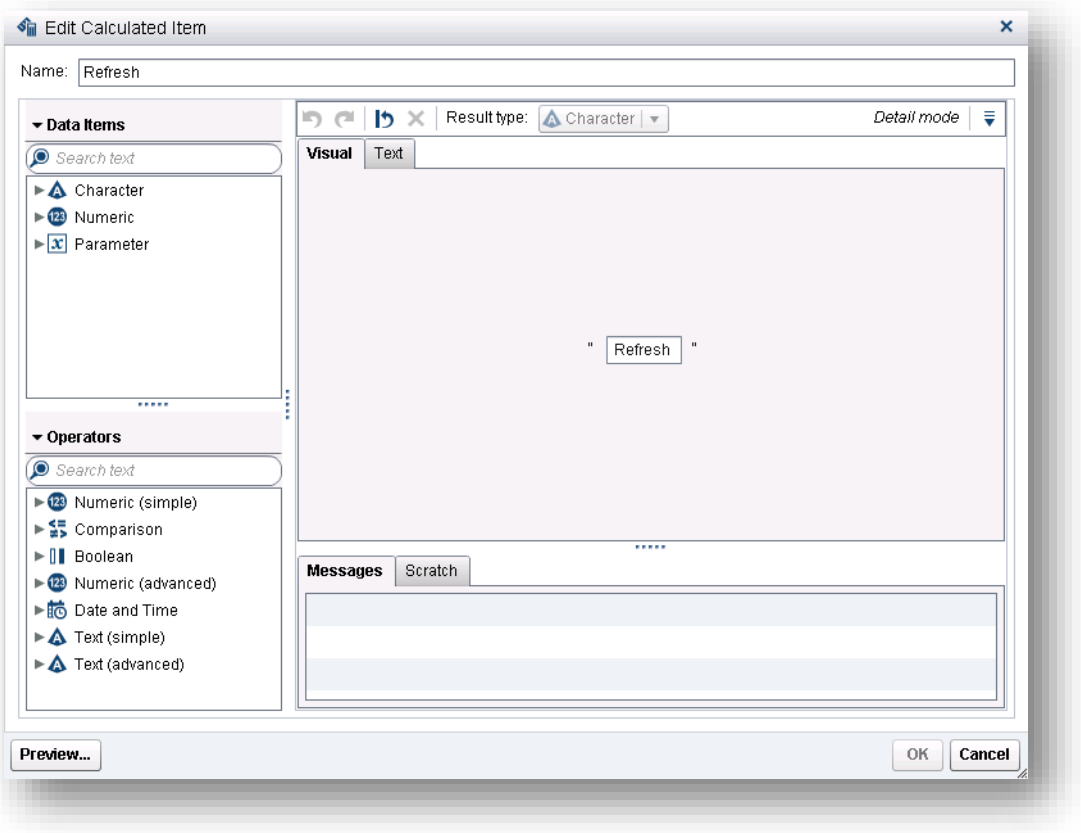
Finally, here is the code for the HTML form, including the JavaScript on-click event handler that changes the button's label to "wait":

2

```
data _null_;
  format infile $char256.;
  input;
  infile = resolve(_infile_);
  file _webout;
  put infile;
cards4;
<HTML>
<HEAD>
  <script type="text/javascript">
    function mychangeText() {
      document.getElementById('MySubmitButton').value='Wait';
      return true;
    };
  </script>
</HEAD>
<BODY>
<FORM ACTION="&_URL">
<INPUT TYPE="HIDDEN" NAME="_program" VALUE="&_program">
<!-- Other HTML input form tags go here -->
<INPUT TYPE="SUBMIT" VALUE="Submit" ID="MySubmitButton" onclick="return mychangeText();" />
</FORM>
</BODY>
</HTML>
;;;;
run;
```

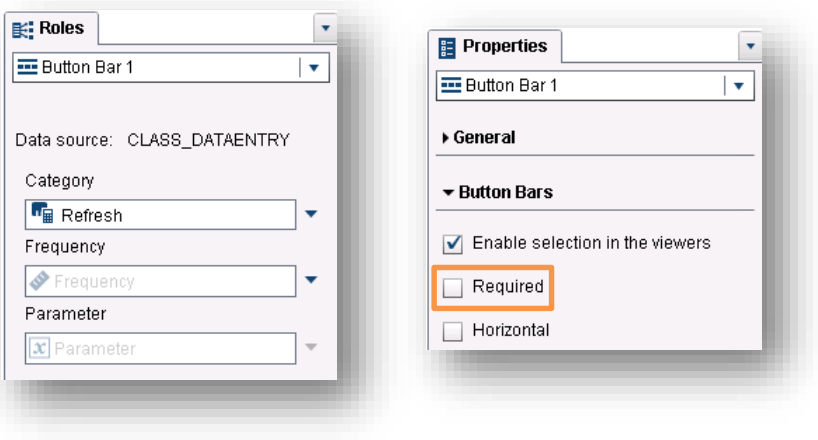
The "Refresh" button you see on the bottom left of Display 12 is a nice addition to the report and is very easy to implement in SAS Visual Analytics Designer. One of the advantages of this custom refresh button is that you can refresh specific report objects in a more controlled manner, preventing the entire report to be refreshed and the stored process from re-executing when it might not be needed. Here are the steps to create it:

1. Create a new Calculated Item in SAS Visual Analytics and call it “Refresh”.



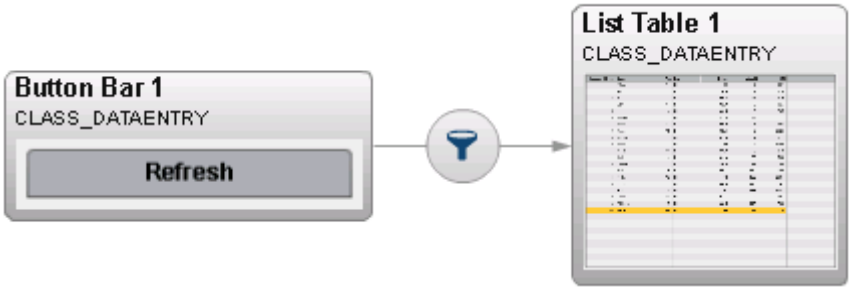
Display 13. New Calculated Item: Refresh

2. Create a new Button Bar (drag the Button Bar object to the work canvas) and add the newly created “Refresh” item to the Category role. On the “Properties” tab for the button bar, make sure that “Required” is NOT selected. The ability to toggle the button bar on and off is what triggers data to be fetched from the LASR Analytic Server when the button is filtering other objects in the report. When new data is fetched, the information displayed on the report object is refreshed.



Display 14. Calculated Item "Refresh" as Category Role in Button Bar

3. Set up filter interactions between the Button Bar you created and the other objects that you want to refresh.



Display 15. Interaction between Button Bar and List Table

Your refresh button can now interact with your list table and will filter it accordingly. Every time you click the refresh button and select or deselect it, the filter changes and new data needs to be queried. The filter doesn't really subset the data because the calculated item has the same value for all rows in the LASR table. If refresh is selected, all rows are returned. If it is deselected, SAS Visual Analytics returns all rows by default.

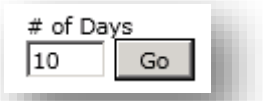
EXAMPLE 4 – MOVING AVERAGE

The moving average is a type of calculation that is used to define a trend, and it is largely used to analyze stock prices. It is useful because it filters out "noise" and random price fluctuations. The moving average differs from a regular average because it drops the oldest data points in favor of newer ones. The data "moves" by taking into account the newest data points, as they become available, while removing old ones. Financial experts put moving averages into two categories: the simple moving average (SMA) and the exponential moving average (EMA). The simple moving average is the average stock price over a period of time, with equal weighting given to each data point. The exponential moving average (EMA) gives bigger weight to the most recent prices.

Once the moving average is calculated, Bollinger bands can be derived from it by applying standard deviations and can then be graphically represented. In this example, the report consumer can define the number of days (which is the stored process parameter) that he or she wants as the time frame. Moving average values and other categorical information is loaded in memory and is displayed in the report, as shown below in Display 16.

Display 16. Bollinger Bands and Moving Average Based on Number of Days That the Report User Inputs

We used the same method presented in example 3 to obtain the stored process input parameter via HTML form and give feedback to the report consumer. In this example, the “Go” button changes its label to “Wait” while the stored process is executing. Display 17 gives us a closer look into the stored process interface that was embedded in the report and highlighted in Display 16.



Display 17. Stored Process User Interface for Moving Averages

Once the data is ready and loaded in memory, the user can click the refresh button (similar to the one in the previous example) to refresh specific report objects. Row-level security is necessary to keep each user’s data separated, so each person can have his or her own parameter values. Keeping user’s contexts separated is a concept similar to the WORK area in Base SAS®, but the difference here is that instead of multiple tables stored in multiple libraries, user’s private tables are appended into a single table and this table is loaded in the LASR Analytic Server.

Figure 4 shows the simplified process flow that explains the stored process code used in Example 4.

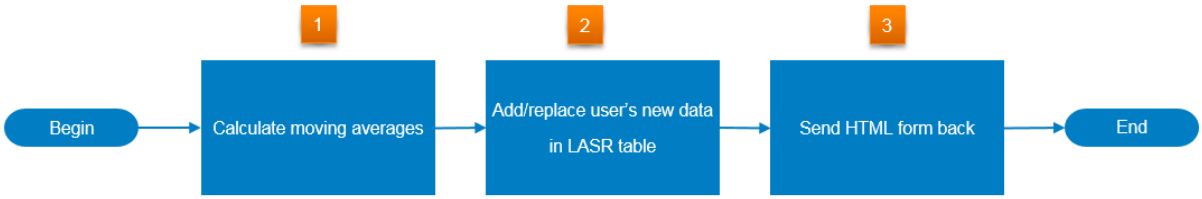


Figure 4. Flow Chart of Stored Process Used in Example 4

The first step in getting this report to work is to create the code to calculate the moving averages. We have removed the details about this calculation because it is not relevant to the ideas presented here:

1

```
data WORK.bollinger;
  set SOURCE.bollinger;
  * code to calculate moving averages SMA and EMA
    based on input parameter T;
run;
```

Once that's done, we add or replace the user's current data with the new data in the LASR table. If this is the first time the user is opening the report and his or her data is not loaded yet, the code below just adds the new data in memory:

2

```
/*
  The next two steps replace user's current data
  with the new calculated data that was based on parameter T.
*/
proc imstat data=LASRLIB.BOLLINGER;
  where User = "&_metauser";
  deleterows / purge;
run;

data LASRLIB.BOLLINGER ( append );
  set WORK.bollinger;
run;
```

Finally, the code that creates the HTML form that captures the number of days for the calculations and re-executes the stored process when the user clicks the Go button.

3

```
data _null_;
  format infile $char256.;
  input;
  infile = resolve(_infile_);
  file _webout;
  put infile;
cards4;
<HTML>
<HEAD>
   <script type="text/javascript">
     function mychangeText() {
       document.getElementById('MySubmitButton').value='Wait';
       return true;
     };
   </script>
</HEAD>
<BODY>

<FORM ACTION="&_URL" METHOD="post" style="font-family:verdana;font-size:8pt;color:black">
<INPUT TYPE="HIDDEN" NAME="_program" VALUE="&_program" />
# of Days
<br>
<INPUT TYPE=TEXT NAME="T" VALUE="&T" />
<INPUT TYPE="SUBMIT" VALUE="Go" ID="MySubmitButton" onclick="return mychangeText();" />
</FORM>

</BODY>
</HTML>
;;;;
run;
```

SECURITY CONSIDERATIONS AND ADDITIONAL BEST PRACTICES

In order for each user to have proper access using their credentials, the SAS code needs to impersonate manipulation of LASR tables by using a different metadata identity with the right access permissions. We do this by using system options for metadata, such as metauser and metapss. (See http://support.sas.com/documentation/cdl/en/lrmeta/67971/HTML/default/viewer.htm#p0e6xowhvfgr7wn1lyz69id8uw4b.htm.) Information that goes in these system options is sensitive and if you store this code in a protected location, you reduce the chances of users gaining access to it, but this will still not be 100% secure. The risk here is that a fairly advanced user could gain access to the code and view the “hidden” credentials that are passed through.

One way to solve this problem is by storing the SAS code as a macro in a catalog, without the source code. In this way, users can run the macro, but don’t have access to the code for it. The code block below creates that macro, which is stored in a catalog (without the source code). By using options nosource, nonotes, and so on, the user cannot view those parts of the code in the log file (which could be displayed in case of error). The following block of code defines the macro and stores it in catalog in a location given by the library macrolib. Storing the catalog in a protected location where only the stored process shared account has access increases security.

```
options mstored sasmstore=macrolb;

%macro impersonate /store secure;
  %local src src2 nts mpr mlo;
  %let src=%sysfunc(getoption(source));
  %let src2=%sysfunc(getoption(source2));
  %let nts=%sysfunc(getoption(notes));
  %let mpr=%sysfunc(getoption(mprint));
  %let mlo=%sysfunc(getoption(mlogic));

  options nosource nosource2 nonotes noprint nomlogic;

  /* main code begin */
 options
   metauser="saslasradm@saspw"
   metapass="{SAS002}1D57933958C580064BD3DCA81A33DFB2";
  /* main code end */

  options &src &src2 &nts &mpr &mlo;
%mend impersonate;
```

The code below must be executed before you attempt to make changes to tables in LASR. Adding options nosource nonotes (and possibly others) as the first line of code in these types of stored processes is highly recommended too.

```
libname macrolb "D:\MyDemo\sassrv Protected Folder";
options mstored sasmstore=macrolb;
%impersonate;
/*
 Here goes the code that make changes to tables in LASR
 */
```

Next, if you allow users to load tables and records in memory on demand, it makes sense to clean them up when they are no longer needed. However, determining when tables or specific records are not being used might be a little tricky. Since the stored processes presented here are capable of loading information back in memory if it's not there, it should be safe to delete these records after some elapsed time, just to simplify the process. In order to do this, we first need to have a timestamp column added. The timestamp would indicate when the records were loaded. Secondly, a job that removes "expired" data based on the timestamp needs to be scheduled. A PROC IMSTAT similar to the one discussed in Example 3 could be used to remove records from a LASR table.

Finally, when tables are not loaded in memory, report objects will be flagged with a red cross, and users might get an error message. To avoid this, you can load empty tables (same structure, but no records), which will flag the report objects with a yellow triangle icon but no error message.

CONCLUSION

By using stored processes alongside SAS Visual Analytics and some HTML and JavaScript, you are putting some great flexibility and power in the hands of your report consumers. Use stored processes to create more complex output, load tables on demand, load records on demand, add, delete, and update rows in a table, and create complex calculations based on user input. The possibilities are seemingly endless, and your report consumers will thank you for giving them a little more power to know.

REFERENCES

Bollinger, John. “Bollinger Bands.” Accessed February 15, 2016. Available at http://www.bollingerbands.com/.

Bostock, Mike. “Data-Driven Documents.” Accessed February 15, 2016. Available at https://d3js.org/.

SAS Institute Inc. *SAS/Graph 9.4: Reference, Fourth Edition*. Accessed February 15, 2016. Available at http://support.sas.com/documentation/cdl/en/graphref/67881/HTML/default/viewer.htm.

SAS Institute Inc. *SAS 9.4 Intelligence Platform: Security Administration Guide, Second Edition*. Accessed February 15, 2016. Available at http://support.sas.com/documentation/cdl/en/bisecag/67045/HTML/default/viewer.htm.

SAS Institute Inc. *SAS 9.4 Language Interfaces to Metadata, Third Edition*. Accessed February 15, 2016. Available at http://support.sas.com/documentation/cdl/en/lrmeta/67971/HTML/default/viewer.htm.

SAS Institute Inc. *SAS 9.4 Stored Processes: Developer’s Guide, Third Edition*. Accessed February 15, 2016. Available at https://support.sas.com/documentation/cdl/en/stpug/68399/HTML/default/viewer.htm.

SAS Institute Inc. *SAS Visual Analytics 7.3: Administration Guide*. Accessed February 12, 2016. Available at http://support.sas.com/documentation/solutions/va/73/en/vaag.pdf.

SAS Institute Inc. *SAS LASR Analytic Server 2.7: Reference Guide*. Accessed February 13, 2016. Available at http://support.sas.com/documentation/cdl/en/inmsref/68736/HTML/default/viewer.htm.

RECOMMENDED READING

- *The 50 Keys to Learning SAS Stored Processes: Must Have Guide for SAS® Developers* by Tricia Aanderud and Angela Hall. Siamese Publishing, 2012.
- *SAS® For Dummies, Second Edition,* by Stephen McDaniel and Chris Hemedinger. Wiley Publishing Inc., 2010.

CONTACT INFORMATION

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

Renato Luppi
SAS Institute Inc.
Renato.luppi@sas.com

Varsha Chawla
SAS Institute Inc.
Varsha.chawla@sas.com