Paper 1737-2014

PROC STREAM and SAS® Server Pages: Generating Custom User Interfaces

Don Henderson, Henderson Consulting Services, LLC

ABSTRACT

Quite often when building web applications that use either the SAS Stored Process Server or the SAS/IntrNet Applications Dispatcher it is necessary to create a custom user interface to prompt for the needed parameters.

Generating a custom user interface can be accomplished by chaining, for example, two or more stored processes together. The first stored process generates the user interface where the user selects the desired options and can use PROC STREAM to process an input SAS Server Page to display the user interface. The second (or later) stored process in the chain generates the desired output.

This paper will describe and show several examples similar to those shown in the SAS Press book SAS Server Pages: Generating Dynamic Contact and on Don Henderson's blog Jurassic SAS® in the BI/EBI World".

INTRODUCTION

This paper presents a number of simple examples that highlight the core capabilities of the STREAM procedure and how those capabilities can be used, in conjunction with the SAS macro language, to create custom user interfaces and prompt pages for web applications that use the Stored Process Server or the SAS/IntrNet Application Dispatcher.

A companion paper addresses the basics of using PROC STREAM and SAS Server Pages as well as for foundation reporting: PROC STREAM and SAS Server Pages: Generating Custom HTML Reports.

All of the sample code and supporting files are available in the expanded zip file for the above mentioned SAS Press book.

The material is this book is a repackaging of content from the author's:

- ebook SAS Server Pages: Generating Dynamic Content
- blog: Jurassic SAS® in the BI/EBI World

The above ebook and blog include numerous additional examples of creating User Interfaces with SAS Server Pages and PROC STREAM (including more realistic real-world examples) can be found on the author's blog. Please consult those blog entries for additional examples as well as to discuss the examples presented in this chapter.

This paper also exists as an article on sasCommunity.org and can be found at PROC STREAM and SAS Server Pages: Generating Custom User Interfaces.

A FRAMEWORK FOR CREATING CUSTOM PROMPTS

Generating a custom user interface for a stored process (or a SAS/IntrNet Application Dispatcher program) can be accomplished by chaining two (or more) stored processes together. The first stored process generates the user interface where the user selects the desired options and uses the STREAM procedure with an input SAS Server Page to display the user interface. The second (or later) stored process in the chain generates the desired output.

For the examples presented here, we will use two generic stored processes. The sasServerPage stored process will typically be invoked to generate the prompt pages for the second stored process in the chain: the runMacro stored process. The runMacro stored process can be used to run any macro that is available to a SAS session (for example, macros defined in a SAS autocall directory). The use of such a runMacro stored process is a very powerful approach in a number of ways:

- If you have lots of reports and want to make them available from the SAS Stored Process Server, you need not create a distinct stored process for each one.
- Alternatively, if you have existing reports packaged as macros (or SAS programmers who are familiar with the SAS macro language), you can invoke the reports by using the runMacro stored process.

As a *Best Practice* consider registering the sasServerPage and runMacro stored processes in the same metadata folder, thus allowing for the stored process locations to reference each other using the &_metafolder macro variable created by the stored process server. The use of &_metafolder eliminates the need to hardcode stored process locations.

Note that the same technique can be used for SAS/IntrNet Application Dispatcher programs. The ability to generate a custom user interface for the stored process server enables you to develop user interfaces that might be preferred to the out-of-the-box user interfaces generated by the SAS BI prompting model. Because the SAS/IntrNet Application Dispatcher does not have a prompting model or metadata about the parameters for each program, the chaining technique is more commonly used with the SAS/IntrNet Application Dispatcher. The rest of this paper will address the stored process server and the code for that environment. Almost identical code can be used with the SAS/IntrNet Application Dispatcher. The details specific to, along with the needed code changes for, the SAS/IntrNet Application Dispatcher will be discussed in SAS/IntrNet SAS Server Page blog postings.

THE SASSERVERPAGE STORED PROCESS AND MACRO

The sasServerPage stored process simply calls a macro called sasServerPage. This allows the macro to be called from other stored processes or programs as needed. This may be desirable if, for example, you want to use the runMacro stored process so you can run a macro to set up the environments for the chained stored processes. That macro could then call the sasServerPage macro as its last step.

```
*ProcessBody;
%sasServerPage(fileref = &srvrpgs     /* SAS Server Pages aggregate fileref */
              ,page = &page           /* name of the SAS Server Page to use */
              ,quoting = &quoting     /* quoting option for PROC STREAM */
              ,asis = &asis           /* asis option for PROC STREAM */
              ,noAbSSCmt = &noAbSSCmt /* noAbsSCmt option for PROC STREAM */
              )
```

The specified parameters to the macro are also stored process parameters. Default values are assigned in the definition of the stored process for each parameter. These values correspond to PROC STREAM's parameters:

- The **fileref** parameter specifies an assigned fileref for an aggregate location (a directory or set of concatenated directories) where the input SAS Server Pages are stored. As a *Best Practice*, consider assigning this fileref in the Stored Process Server Request Init program. Alternatively, it can be assigned in the autoexec for the stored process server itself.
- The **page** parameter specifies the name of the SAS Server Page to be processed.
- The **quoting** parameter specifies the handling of single and double quotation marks (the value to be used for the PROC STREAM quoting parameter).
- The **asis** (also documented as PRESCOL for Preserve Columns) parameter is either blank or has the value asis (corresponding to the like-named PROC STREAM parameter).
- The **noAbsSSCmt** parameter is either blank or has the value &noAbsSSCmt (corresponding to the like-named PROC STREAM parameter).

The sasServerPage macro source is reasonably straight-forward:

```
%macro sasServerPage
       (fileref = srvrpgs        /* fileref for server page locations  */
       ,page =                   /* list of server pages               */
       ,defaultExtension = html  /* default extension if not provided  */
       ,outfile = _webout        /* destination to write output to     */
       ,mod =                    /* mod if appending to &outfile       */
       ,quoting = both           /* single/double quote handling       */
       ,asis =                   /* attempt to preserve columns/ASIS   */
       ,noAbSSCmt =              /* include slash-star comments        */
       );

 %local quot amp apos lt gt nbsp copy reg;
 %let quot = &#034;
 %let amp = &#038;
 %let apos = &#039;
 %let lt = &#060;
 %let gt = &#062;
 %let nbsp =  
 %let copy = &#169;
 %let reg = &#174;
```

```
 %if %scan(&page,2,.) = %then %let page = &page..&defaultExtension;

 proc stream outfile = &outfile &mod &asis &noAbSSCmt
             %if %length(&quoting) > 0 %then quoting=&quoting;
 ;
   BEGIN
   %include &fileref(&page);
 ;;;;
 run;

%mend sasServerPage;
```

Common Character Entity References are quite often used in html pages that may serve as the source for your SAS Server Pages. For example, " is the double quote(“) character. When referenced in a SAS Server Page, suc references would either cause a bogus apparent symbolic reference warning message, or worse, an unexpected substitution. To avoid this, the sasServerPage macro creates those macro variables (the %local statement) and then assigns values that are the corresponding Numeric Entity References. As a result, if these character sequences are in the input file, SAS attempts to resolves them as macro variables to the numeric entity (note that trailing ; is preserved).

Note that the macro has three additional parameters not defined as parameters for the stored process because they rarely, if ever, need to be modified from their default values when the macro is run from the stored process. They are likely to be used when the sasServerPage macro is called directly from another SAS program:

- the output location, **outfile**, which for a stored process is almost always _webout.
- **mod**, which in the stored process (and SAS/IntrNet Application Dispatcher) environment is always enabled.
- the **defaultExtension**, which again is almost always html. (Note that in a stored process environment the default extension is not used if the extension is included in the *passed-in* value of the page parameter.)

THE RUNMACRO STORED PROCESS

The runMacro stored process has at least one parameter—macroToRun, which specifies the name of the macro to run. (It might also have more parameters, depending on your environment.) Additional information about this approach can be found in the sasCommunity.org article The runMacro Stored Process, which includes links to numerous blog entries that highlight the various benefits of this approach.

For the purposes of this paper, a very simple version of the runMacro stored process will be used:

```
*ProcessBody;
%stpbegin;
%put NOTE: Execute macro &macroToRun..;
%&macroToRun
%stpend;
```

The name of the macro to run is passed as an input parameter. As a result, ***%&macroToRun*** resolves to a % sign followed by the macro name, thus invoking the specified macro.

UTILITY MACROS

Utility macros play an important role in building custom user interfaces. For example, macros that can be referenced/run in a SAS Server Page that:

- Generate a pull down list (e.g., a select tag with multiple options) of choices where the choices are populated from data in an input SAS data set.
- Generate either a series of checkboxes or radio buttons, again from the information in a SAS data set.
- Create a data set of distinct values that can be used to create a select tag or a series of checkboxes.
- The SAS Press book SAS Server Pages: Generating Dynamic Contact and my blog Jurassic SAS® in the BI/EBI World contain numerous other examples of such macros, as well as more complete/robust versions of the macros described here.

Several of these that are used in the **SELECTED EXAMPLES** section (below) are briefly discussed in the following subsections.

The generateOptionTag Macro

The generateOptionTag macro reads the rows in the input data set using the data access functions (OPEN, CLOSE, FETCH, GETVARC, GETVARN) that are executed using the %SYSFUNC macro. It has the following parameters:

- The **data** parameter specifies the name of the data set to use to populate the select tag.
- The **var** parameter specifies the name of the variable that provides the value (that is, that value submitted to the server when the form is submitted) for the option tag.
- The **label** parameter specifies the name of the variable that provides the label (the visible text) for the option tag. If not provided, it defaults to the value of the var parameter.
- The **name** parameter specifies the name of the select tag form field. If not provided, it defaults to the value of the var parameter.
- The **where** parameter specifies an optional WHERE clause used to subset the data set specified in the data parameter.
- The **selected** parameter specifies a currently selected value (only one value can be provided; the macro does not support multiple selections). Select tags that allow for multiple selections are best handled by a dual list box selector. A sample SAS Server Page implementation of a dual list box selector is described in this blog posting of a utility/template SAS Server Page.
- The **blank** parameter, when given a non-blank value, specifies an additional option tag that is listed as the first option. This parameter can be used to specify the first listed value such as *--Please Select an XXXX--*.
- The **blankValue** parameter specifies an optional value to use for the option tag generated by the blank parameter.
- The **otherOptions** parameter specifies a parameter that can be used to pass any other text to be used in the select tag. This parameter is typically used to specify additional attributes for the select tag such as onSubmit, style, and so on.

The generateCheckBoxes Macro

Like the generationOptionTag macro, this macro uses the data access functions (OPEN, CLOSE, FETCH, GETVARC, GETVARN) that are executed using the %SYSFUNC macro. It also uses the **getDistinct** macro (discussed below) to create the data set that is used to populate the list of checkboxes. It has the following parameters:

- The **data** parameter specifies the name of the data set used to create the list of checkboxes.
- The **var** parameter specifies the name of the variable that provides the value (that is, that value submitted to the server when the form is submitted) for the checkbox.

The getDistinct Macro

The getDistinct macro is an example of a simple utility macro that can be stored in a macro autocall directory and used whenever a list of distinct values is needed. The version presented here generates SQL code and so when it is referenced/called in a SAS Server Page, the DOSUBL function must be used to invoke it.

It has the following parameters:

- The **data** parameter specifies the name of the data set from which the distinct values are to be obtained.
- The **vars** parameter specifies the variable(s) for which the distinct values (or distinct combinations if multiple variables are listed) are desired. A distinct list of combinations of values could be used, for example, when creating cascading select tags.
- The **out** parameter specifies the name of the output data set containing the default values. The DATAn convention is the default.

UTILITY SAS SERVER PAGES

The framework suggested here includes functionality that is commonly requested, for example the ability to:

- Display the output in the same window as the user interface where the user specified the options for producing the desired output.

- Toggle the user interface of the available selections so as to maximize the amount of screen real estate for the output, while still allowing for easy access to the selection user interface for making changes.
- Display a processing image to alert the user that the report is running—something that is important for reports that don't return results immediately.

Instead of duplicating the HTML (and JavaScript) that provides this functionality in each and every SAS Server Page that is used to prompt the user for parameters, the framework presented here leverages the %INCLUDE facility of PROC STREAM. That facility allows for the development of a standard structure for prompt pages and reports without requiring duplication of code. Storing common code in an external file and using the %INCLUDE statement to include the output created by PROC STREAM should be considered a *Best Practice* that is well known to most/all SAS programmers and developers. Such SAS Server Pages contain at least three sections:

- The first section (see **The stpHeader SAS Server Page** below) simply uses %INCLUDE to define a standard page header.
- The next part is the specific HTML, including macro variable references and macros, to generate the user interface for the subject report accessed as a stored process (in this case using the runMacro stored process).
- The last section (see **The stpTrailer SAS Server Page** below) simply uses %INCLUDE to define a standard page trailer.

This technique has broad applicability as the header or trailer section can also be used to display application or corporate logos, including confidentiality or copyright text. And, of course, multiple header and trailer sections can be mixed/matched as needed.

The stpHeader SAS Server Page

The stpHeader SAS Server page include the HTML and JavaScript necessary to implement the features mentioned above:

- Display the output in the same window as the user interface.
- Toggle the user interface.
- Display a processing image to alert the user that the report is running.

As mentioned above, the following source can be easily changed and customized in order to suit the needs of your stored processes, thus enabling you to create any number of alternative user interfaces. A basic understanding of HTML and JavaScript by the reader is assumed.

```
%global pageTitle _debug stpToRun whatToRun;
%let _debug = %sysfunc(coalesceC(&_debug,0));
%let stpToRun = %sysfunc(coalescec(&stpToRun,runMacro));
<html>
<head>
<script>
function toggleUI() {
 if (document.getElementById('showUI').style.display == 'none')
 { document.getElementById('showUI').style.display = 'block';
   document.getElementById('hideUI').style.display = 'none';
   document.getElementById('UI').style.display = 'none';
 }
 else
 { document.getElementById('showUI').style.display = 'none';
   document.getElementById('hideUI').style.display = 'block';
   document.getElementById('UI').style.display = 'block';
 }
}
function processing() {
 window.frames('results').document.body.innerHTML =
   '<center>'
 + '<b><i>Please Wait</i></b>'
 + '<p><img src="/SASStoredProcess/images/progress.gif">'
 ;
 toggleUI();
```

Make sure the macro variables whose values should be defined in the SAS Server Page that includes this file exist.

Default to runMacro if stpToRun not specified.

Define the JavaScript function that toggles the form display on and off.

Define the JavaScript function that displays the “Please Wait” text and image when the form is submitted.

The display of the form is automatically toggled off when the user clicks the submit button.

```
}
</script>
<title>&pagetitle</title>
</head>
<body>
<a id="showUI"
   href="JavaScript:toggleUI();"
   title="Show Selections"
   style="text-decoration:none; display:none;">
<img src="/SASStoredProcess/images/arrow-right-default.gif"
     border="0">
</a>
<a id="hideUI"
   href="JavaScript:toggleUI();"
   title="Hide Selections"
   style="text-decoration:none; display:block;">
<img src="/SASStoredProcess/images/arrow-left-default.gif"
     border="0">
</a>
<table style="width:100%; height:100%;">
<tr>
<td id="UI"
    style="float:left; display:block; vertical-align:top">
<form action="&_url"
      target="results"
      onSubmit="processing();">
<input type="hidden"
       name="_debug"
       value="&_debug">
<input type="hidden"
       name="_program"
       value="&_metafolder&stpToRun">
<input type="hidden"
       name="macroToRun"
       value="&whatTorun">
<input type="hidden"
       name="page"
       value="&whatTorun">
```

Clicking this image toggles the form display on. The form is displayed by default on load, so the display of this image is set to none on load.

Clicking this image toggles the form display off. The form is displayed by default on load, so the display of this image is set to block on load.

Define an HTML table with two columns. The table fills the entire browser window. The left column is the HTML form (note that the ID value matches the value used in the toggleForm JavaScript function).

Define the form for the prompts. The target attribute directs the output of the form submission to the iframe defined in the stpTrailer SAS Server Page. The onSubmit calls the processing JavaScript function that informs the user that the request is processing.

Assuming that the *sasServerPage* and *runMacro* stored processes are used and are in the same metafolder, the required form fields (_program, page and macroTorun) are specified. These could be further parameterized to support other stored processes. *&whatToRun* is defined as the value for both *macroToRun* and *page*. When *stpToRun* is *runMacro*, *macroToRun* is used and *page* is ignored; when *stpToRun* is *sasServerPage*, *page* is used and *macroToRun* is ignored.

This SAS Server Page uses images provided by SAS for the right and left arrows. If other images are desired, the above img tags can be replaced with other images. Alternatively, an image can be displayed using the Base 64 URI Data Scheme. For example, the right arrow above could be replaced with the following text:

<img
src="" border="0">

The left arrow could be replaced with the following text:

<img
src="" border="0">

Using this scheme allows for the image source to be included as text in the SAS Server Page. In addition, referencing utility images such as the arrows in this way also allows for such SAS Server Pages to be used without concern for the location of the directory or folder where the images are located. This can be advantageous for output that is not created by the stored process server or for the use of images that are not provided by SAS. See the blog posting The getImage macro: embedded images using text strings for a utility macro that can be used to parameterize the loading of the image text from a SAS dataset.

The stpTrailer SAS Server Page

The stpTrailer SAS Server Page is included by the calling SAS Server Page to close the form and to define the display area for the stored process output. Like the stpHeader SAS Server page, it can be easily changed and

customized in order to suit the needs of your stored processes.

```
</form>
</td>
<td style="overflow-x:hidden; display:block; vertical-align:top">
<iframe name="results"
        width="100%"
        height="100%"
        frameborder="0">
</iframe>
</td>
</tr>
<tr>
<td colspan="2" style="height:15; background-color:#E4E4E4; color:#999999;
font-family:Geneva, Arial, Helvetica, sans-serif; font-size:10px; font-
style:italic;">
<div style="float:left;">
<a
href="http://www.sascommunity.org/wiki/SAS%C2%AE_Server_Pages:_Generating_Dynam
ic_Content"
 style="text-decoration:none" target="_blank">
SAS&reg; Server Pages: Generating Dynamic Content
</a>
</div>
<div style="float:right;">
<a href="http://www.hcsbi.com"
   style="text-decoration:none" target="_blank">
Henderson Consulting Services
</a>
</div>
</td>
</tr>
</table>
</html>
```

Close the form and the left-hand table cell and open the right-hand cell using *overflow-x:hidden* to designate that it should use all the remaining horizontal screen real estate.

Define the iframe for the stored process output (the form targets the *results* window for the output) and have it use the full width and height of the containing table cell.

Optionally a second table row can be used to display standard text at the bottom of the page. This text illustrates that option using information and links about the author's eBook.

SELECTED EXAMPLES

Three examples of using this framework are included here:

- Generating a select tag for the user to select which product to include in a report.
- Generating a series of checkboxes for the user to select which products to include in a report.
- Prompting the user to select a drill down hierarchy for a summary data set.

Thanks to the framework discussed above, the html snippets for each of these examples is fairly simple. The complexities of the interface are built into the framework so you need not worry about them for each individual stored process.

The sasCommunity version of this paper, will be updated to include live links to the author's demo server for these examples. You can also add the page to your watchlist so you are notified of changes.

GENERATING A SELECT TAG

The SAS Server Page below generates the user interface seen in Figure1.

- Lines 1 through 3 ensure the needed macro variables exist and have then needed values. Note that since stpToRun defaults to runMacro and we are running a macro, the default value for stpToRun will be used and macro selectedProductsReport will be run.
- Line 4 includes the standard header.
- Line 5 uses the DOSUBL function to invoke the getDistinct macro to create a data of the distinct values of product.

- Line 6 invokes the generateOptionTag macro to create a select tag of the distinct products using the data set created on line 5. Note that &sqlobs is available from the DOSUBL call to the getDistinct macro and is used to customize the size of the select tag.
- Line 7 is the submit button (Run Report) that the user selects to submit a report. Upon clicking the Run Report button, the output shown in Figure 2 is displayed.
- Line 8 includes the standard trailer.

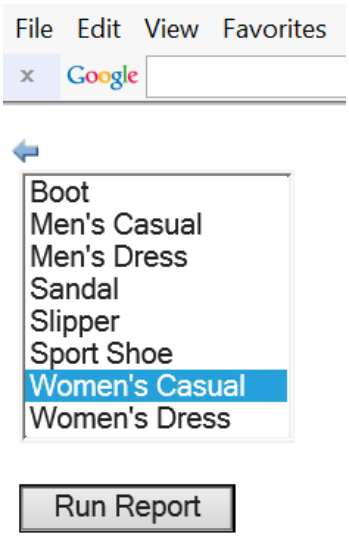


Figure 1. Select Tag

```
1.  %global pageTitle _debug stpToRun  whatToRun;
2.  %let pageTitle=Products Select Tag;
3.  %let whatTorun=selectedProductsReport;

4.  &streamDelim;%include &srvrpgs(stpHeader.html);

5.  %let rc = %sysfunc(dosubl(%nrbquote(
              %getDistinct(data=sashelp.shoes
                           ,vars=product

                           ,out=productList)
    )));
6.  %generateOptionTag(data=productList
                       ,var=product
                       ,otherOptions=multiple size=&sqlobs
                       )

7.  <p><input type="submit" value="Run Report">

8.  &streamDelim;%include &srvrpgs(stpTrailer.html);
```

Note in Figure 2 that the select tag is hidden once the Run Report button was selected. The left pointing arrow is also replaced by a right pointing arrow (which can be clicked on to bring back the user interface, i.e., the select tag) so the user can select a different product. This toggling of the user interface both maximizes the screen real estate for the output report, while at the same time allowing the user to run a different report without having to worry about browser back buttons or multiple browser windows.

In addition to how the above example is packaged, here are numerous alternative ways to package this functionality:

- A driver SAS Server Page could have been used where the custom user interface for the specific report was included. An example of this can be seen in the blog posting A SAS Server Page AJAX Template, which as the name implies uses AJAX (instead of iframes).
- The above lines could have been packaged as a macro, in which case macro parameters could have been used instead of %let statements.

Figure 2. Report Output from the Figure 1 User Interface

GENERATING A SERIES OF CHECKBOXES

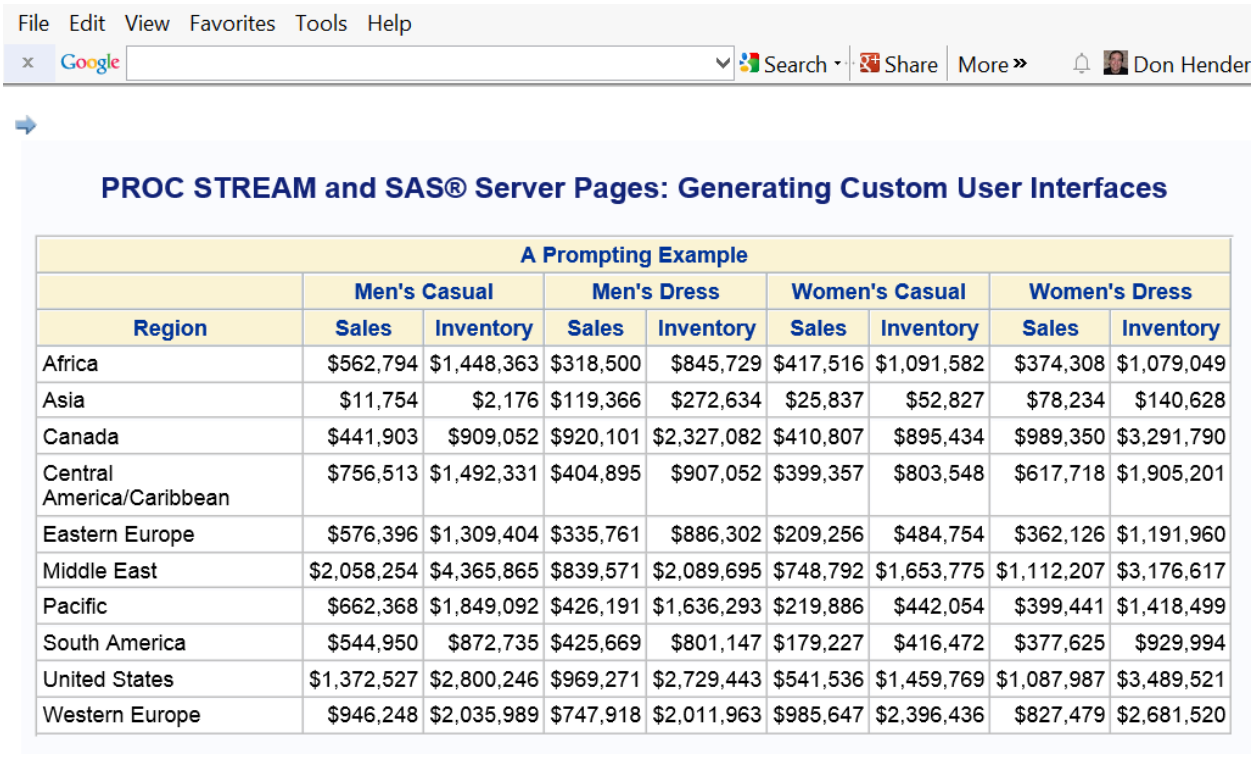
When more than one choice is to be allowed it is quite often more intuitive for the user (i.e., users don't need to know about using the CNTL key) if the selections are a series of checkboxes as seen in Figure 3. The SAS Server Page below generates this user interface.

Figure 3. Checkboxes

- Lines 1 through 4 and lines 7 and 8 are the same as the corresponding lines in the example above.
- Line 5 uses the DOSUBL function to a PROC SQL statement that uses a select distinct on the product code and the CATS function to create a macro variable whose value is the set of checkboxes, one for each product.
- Line 6 simply references the macro variable created in line 5.

Upon clicking the Run Report button, the output shown in Figure 4 is displayed.

```
1. %global pageTitle _debug stpToRun  whatToRun;
2. %let pageTitle=Products Checkboxes;
3. %let whatTorun=selectedProductsReport;

4. &streamDelim;%include
   &srvrpgs(stpHeader.html);

5. %let rc = %sysfunc(dosubl(%nrbquote(
     proc sql noprint;
      select distinct cats('&streamDelim newline;<input type="checkbox"'
                          ,' name="product" value="'
                          ,strip(product)
                          ,'">'
                          ,strip(product)
                          )

      into:checkBoxes separated by '<br>'
      from sashelp.shoes;
     quit;
   )));

6. &checkboxes

7. <p><input type="submit" value="Run Report">

8. &streamDelim;%include &srvrpgs(stpTrailer.html);
```



A Prompting Example								
	Men's Casual		Men's Dress		Women's Casual		Women's Dress	
Region	Sales	Inventory	Sales	Inventory	Sales	Inventory	Sales	Inventory
Africa	$562,794	$1,448,363	$318,500	$845,729	$417,516	$1,091,582	$374,308	$1,079,049
Asia	$11,754	$2,176	$119,366	$272,634	$25,837	$52,827	$78,234	$140,628
Canada	$441,903	$909,052	$920,101	$2,327,082	$410,807	$895,434	$989,350	$3,291,790
Central America/Caribbean	$756,513	$1,492,331	$404,895	$907,052	$399,357	$803,548	$617,718	$1,905,201
Eastern Europe	$576,396	$1,309,404	$335,761	$886,302	$209,256	$484,754	$362,126	$1,191,960
Middle East	$2,058,254	$4,365,865	$839,571	$2,089,695	$748,792	$1,653,775	$1,112,207	$3,176,617
Pacific	$662,368	$1,849,092	$426,191	$1,636,293	$219,886	$442,054	$399,441	$1,418,499
South America	$544,950	$872,735	$425,669	$801,147	$179,227	$416,472	$377,625	$929,994
United States	$1,372,527	$2,800,246	$969,271	$2,729,443	$541,536	$1,459,769	$1,087,987	$3,489,521
Western Europe	$946,248	$2,035,989	$747,918	$2,011,963	$985,647	$2,396,436	$827,479	$2,681,520

Figure 4. Report Output from the Figure 3 User Interface

The same PROC SQL technique could have also been used to generate the select tag in the previous example. The use of PROC SQL to create a macro variable whose value is the HTML code however is not nearly as robust as using macros like generateOptionTag and generateCheckboxes. The issue is that macro variables are limited to 32K and quite often that limit can be exceeded, especially if the display values are long text strings. The SAS Server Page below uses the generateCheckBoxes macro instead of PROC SQL to create a macro variable and is more robust as there is no risk of exceeding the 32K limit. It produces the same user interface and output as seen in Figures 3 and 4.

The code is basically the same as that seen for the select tag above. The only differences are the page title value specified in line 2 and calling the generateCheckBoxes macro in line 6 instead of the generateOptionTag macro.

```
1. %global pageTitle _debug stpToRun  whatToRun;
2. %let pageTitle=Products Checkboxes;
3. %let whatTorun=selectedProductsReport;

4. &streamDelim;%include &srvrpgs(stpHeader.html);

5. %let rc = %sysfunc(dosubl(%nrbquote(
   %getDistinct(data=sashelp.shoes
                ,vars=product
                ,out=productList)
   )));

6. %generateCheckBoxes(data=productList
                       ,var=product
                       )

7. <p><input type="submit" value="Run Report">

8. &streamDelim;%include &srvrpgs(stpTrailer.html);
```

SELECT A DRILL DOWN HIERARCHY FOR A SUMMARY DATA SET

This example illustrates the integration of a resource found on Web. All of us programmers know about copying and modifying something that already exists instead of building something from scratch. And the Web, with the help of a Google search, provides a rich source of samples and starting points for readers to create their own SAS Server Pages that can do most anything.

The idea for this example was a result of the observation that the SAS BI tools did not contain any facilities to easily generate a drillable table using data from a SAS data set. Web Report Studio and Enterprise Guide both can create drillable tables, but the source data must be a cube, not a SAS data set.

In researching options I found the treetable plugin for JQuery. The SAS Server Page below is a simple user interface for another SAS Server Page that embeds the JQuery treetable widget. As seen in Figure 5, the user selects the drill hierarchy to use for the drill-down.

Again, this SAS Server Page is very similar to the examples above (since we are leveraging the same framework).

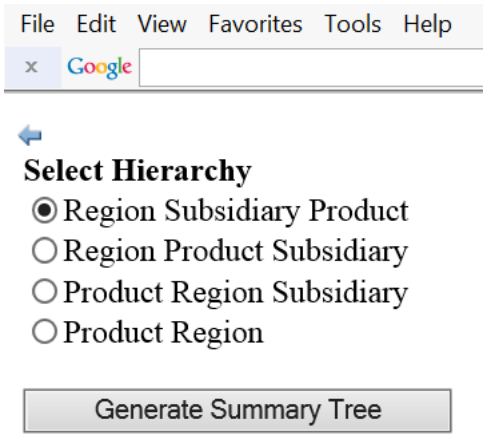


Figure 5. Select Hierarchy User Interface

- Since the target page for the user's selection is another SAS Server Page, in line 3 we specify that the stored process to run is sasServerPage instead of macroTorun.
- The name of the SAS Server Page which embeds the treetable plugin/widget is specified in line 4.
- Line 6 includes the radio buttons that the user will click on to specify what drill hierarchy they want to examine.
- Upon clicking the Generate Summary Tree button the user sees the output in Figure 6.The user can drill and collapse nodes, without waiting on server requests, by clicking on the right pointing arrow (to expand) or the left pointing arrow (to collapse).

```
1. %global pageTitle _debug stpToRun  whatToRun;
2. %let pageTitle=Summary Drill Down - a Tree View;
3. %let stpToRun=sasServerPage;
4. %let whatTorun=jqueryTreetable;

5. &streamDelim;%include &srvrpgs(stpHeader.html);

6. <b>Select Hierarchy</b>
   <br><input type="radio" name="class" value="Region Subsidiary Product"
   checked> Region Subsidiary Product
   <br><input type="radio" name="class" value="Region Product Subsidiary">
   Region Product Subsidiary
   <br><input type="radio" name="class" value="Product Region Subsidiary">
   Product Region Subsidiary
   <br><input type="radio" name="class" value="Product Region">
   Product Region
   </select>

7. <p><input type="submit" value="Generate Summary Tree">

8. &streamDelim;%include &srvrpgs(stpTrailer.html);
```

Drill Hierarchy: Region Subsidiary Product

LevelText	Number of Stores	Total Sales	Total Inventory	Total Returns
Overall	4601	$33,851,566	$99,105,051	$1,172,092
Africa	532	$2,342,588	$7,101,073	$74,087
Asia	65	$460,231	$1,176,139	$10,895
Canada	442	$4,255,712	$13,110,709	$129,394
Central America/Caribbean	539	$3,657,753	$10,173,878	$126,898
Eastern Europe	379	$2,394,940	$7,952,471	$86,701
Middle East	397	$5,631,779	$14,208,749	$206,880
Pacific	356	$2,296,794	$7,971,291	$77,129
South America	632	$2,434,783	$5,986,094	$102,851
United States	617	$5,503,986	$16,582,397	$187,502
Western Europe	642	$4,873,000	$14,842,250	$169,755

Figure 6. Drillable Table of Summary Data from the Figure 5 User Interface

The detailed discussion of the jqueryTreeTable SAS Server Page and the macro that it uses to run PROC SUMMARY to create the summary data set (via DOSUBL) will be addressed in a blog entry (the link for that blog entry will be added to the sasCommunity article for this paper: at PROC STREAM and SAS Server Pages: Generating Custom User Interfaces.

CONCLUSION

PROC STREAM is a very powerful new facility in SAS as it greatly extends the kinds of output that can be generated by SAS programs. It also is a great tool for leveraging other web and HTML based facilities (e.g., JQuery plugins).

The examples presented in this paper only scratch the surface of what can be done with PROC STREAM (and SAS Server Pages). Additional examples and discussion can be found in the author's eBook and blog:

- ebook SAS Server Pages: Generating Dynamic Content
- blog: Jurassic SAS® in the BI/EBI World

Please also review the companion paper which addresses the basics of using PROC STREAM and SAS Server Pages as well as for foundation reporting: PROC STREAM and SAS Server Pages: Generating Custom HTML Reports.

REFERENCES

Henderson, Don. 2013. SAS Server Pages: Generating Dynamic Content. Cary, NC: SAS Institute Inc.

Henderson, Don. 2007. Building Web Applications with SAS/IntrNet: A Guide to the Application Dispatcher. Cary, NC: SAS Institute Inc.

Henderson, Don. Jurassic SAS® in the BI/EBI World, Available at http://hcsbi.blogspot.com.

Henderson, Don. 2014. "PROC STREAM and SAS Server Pages: Generating Custom User Interfaces" *Proceedings of the SAS Global 2014 Conference*>. Cary, NC: SAS Institute. Available online at PROC STREAM and SAS Server Pages: Generating Custom User Interfaces

ACKNOWLEDGMENTS

Thanks to SAS Press for allowing me to use material from SAS Server Pages: Generating Dynamic Content for this paper.

RECOMMENDED READING

Henderson, Don. 2013. SAS Server Pages: Generating Dynamic Content. Cary, NC: SAS Institute Inc.

Henderson, Don. Jurassic SAS® in the BI/EBI World, Available at http://hcsbi.blogspot.com/.

CONTACT INFORMATION

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

Name: Don Henderson
Organization: Henderson Consulting Services, LLC
Address: 3470 Olney-Laytonsville Road, Suite 199
City, State ZIP: Olney, MD 20832
Work Phone: (301) 570-5530
Fax: (301) 576-3781
Email: Don.Henderson@hcsbi.com
Web: http://www.hcsbi.com
Blog: http://hcsbi.blogspot.com
sascommunity.org: http://www.sascommunity.org/wiki/User:Donh and
http://www.sascommunity.org/wiki/Presentations:Donh_Papers_and_Presentations