Paper 1738-2014

PROC STREAM and SAS® Server Pages: Generating Custom HTML Reports

Don Henderson, Henderson Consulting Services, LLC

ABSTRACT

ODS is a power tool for generating HTML based reports. Quite often however there are exacting requirements for report content, layout and placement that can be done with HTML (and especially HTML5) that can't be done with ODS.

This presentation will show several example that use PROC STREAM and SAS Server Pages in a batch (e.g., scheduled tasks, using SAS Display Manager, using SAS Enterprise Guide) to generate such custom reports. And yes, despite the name SAS Server Pages, this technology, including the use of JQuery widgets, does apply to batch environments

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", for example:

- A sample mail-merge application
- Creating a custom calendar
- Displaying data graphically, e.g., the use of Tag Clouds as an alternative to bar or pie charts

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 data-driven dynamic content. Basic techniques are illustrated, and those techniques are expanded upon in the by presenting a number of examples that use PROC STREAM for real-world use cases in a Foundation SAS environment.

A companion paper addresses using PROC STREAM and SAS Server Pages in BI environments: PROC STREAM and SAS Server Pages: Generating Custom User Interfaces

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 (including more realistic real-world examples).

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

DYNAMIC CONTENT GENERATION USING THE MACRO FACILITY AND THE STREAM PROCEDURE

The original motivation behind SAS Server Pages was a recognition that although the DATA step can be used to generate virtually any data-driven content, and specifically HTML, generating valid HTML can be complicated by the fact that SAS and HTML syntax conflict with each other at times. For example, both require strings to be quoted. For HTML, the double quotation mark (") character is used. For SAS, either a single (') or double quotation mark (") can be used. In addition, SAS uses the ampersand (&) as a trigger to indicate that the next word or token should be interpreted as a macro variable reference. In HTML, however, the & is interpreted as the separator character for name/value pairs in URLs and as the prefix for HTML character entity references (for example, &nbsp;—a non-breakable space). These differences complicate the standard SAS techniques (such as PUT statements) for generating custom output and make those programs tedious to write and debug. Quite often, a major portion of the HTML that needs to be generated already exists as an HTML file or could be easily created with an HMTL editor. This obviates the need to code PUT statements to generate such HTML. A better alternative was to embed logic into the HTML files themselves just as it is in Active Server Pages (ASP) and Java Server Pages (JSP), which were an integral part of the evolution of generating dynamic and rich content.

Previous implementations of SAS Server Pages (using SAS 9.1.3, 9.2, and, yes, even SAS 8) used the conditional and iterative processing capability of the SAS macro language to generate text. Those implementations were limited by a number of constraints inherent in how the DATA step and the RESOLVE function worked. Those limitations have been removed by PROC STREAM, which provides robust support for dynamic content generation using a language well known to SAS programmers: the macro language. The following is a list of some of the limitations that PROC STREAM removed, along with selected new features that PROC STREAM supports:

- The 32K limit for how much text a single input line can generate has been removed. This 32K limit was imposed because of the need to use DATA step variables.
- The %INCLUDE statement can now be used. It is used to implement what JSP and ASP applications refer to as Server-Side Includes.
- SAS code can be defined and executed inside of an input SAS Server Page. An input text file, referred to as a SAS Server Page, can include both SAS code to be executed as well as markup text (such as HTML, CSV, XML, and more).
- SAS functions and many SCL functions (for example, the data access functions such as OPEN, CLOSE, FETCH, GETVARC, GETVARN, and so on) can be used via the %SYSFUNC macro function, which provides facilities to access SAS data sets and include data values directly in a SAS Server Page.
- Macros can now be defined in a SAS Server Page, and macro calls and functions are no longer restricted to a single input line.

THE SAS MACRO LANGUAGE—A TEXT PROCESSING LANGUAGE

The SAS macro language has been a key component of SAS software since the early 1980s and is widely used to generate data-driven SAS code, as well as enabling conditional execution. At its core, the SAS macro language is simply a text-processing language that to date has been used almost exclusively to generate SAS code. The results of macro processing are passed along to the rest of SAS for processing and execution. Although generating code has been its primary use, it can generate any type of text.

The SAS Press book Building Web Applications with SAS/IntrNet®: A Guide to the Application Dispatcher introduced SAS Server Pages. It introduced the idea that a SAS DATA step using the RESOLVE function could enable macro execution with the results redirected to another location (such as a file or the user's browser when the DATA step is used in the SAS/IntrNet Application Dispatcher or the SAS Stored Process Server). As a result, using the SAS macro language to generate any other type of text (such as HTML, XML, CSV, and so on) became an option, since you could now control where the generated text is sent to.

A very simple example of this is Sample 37050: Creating a custom HTML input form for a SAS® Stored Process, available on the SAS Web site. In this example, hardcoding is unnecessary for the following reasons:

- The values for some of the name/value pairs could be obtained via macro variable resolution.
- A select tag that is created by a macro call could be added.

Although the use of the RESOLVE function in this example allows for macro processing, it is somewhat limiting for the reasons listed above. As just one example, the amount of text that the select tag macro can generate is limited to 32K when using the RESOLVE function approach. This limitation does not apply with PROC STREAM.

PROC STREAM provides a formal and robust tool to allow a SAS job stream to include any text-based content, including macro variables and macros, by redirecting the results of SAS tokenization of the input file to another location. Any text (also referred to as tokens) between a BEGIN keyword (the first token following the PROC STREAM statement) and four adjacent semi-colons (;;;;) is parsed by the SAS word scanner and passed to PROC STREAM for processing.

The following subsections provide an overview of some of the basic techniques that can be used with PROC STREAM to create dynamic and rich content:

- Macro Variables for Text Substitution
- Macros for Iterative and Conditional Text Generation
- Including Text from External Files
- Generating Dynamic Content from a SAS Data Set
- Executing SAS Code in a PROC STREAM Input File

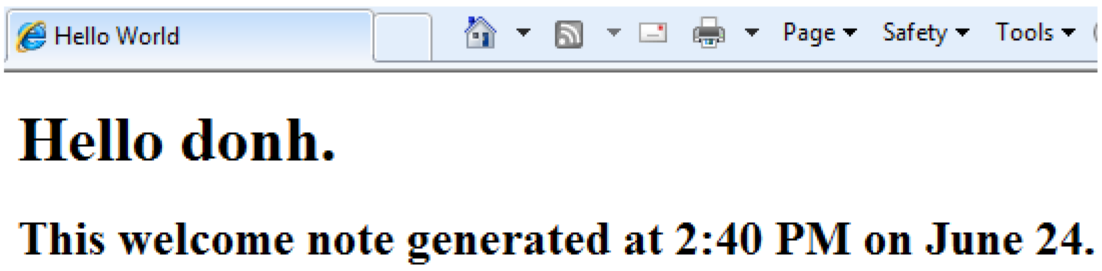
MACRO VARIABLES FOR TEXT SUBSTITUTION

This example illustrates a few key features:

- Macro variable references are resolved (&SYSUSERID).
- The %SYSFUNC macro function is used to execute functions, including the formatting of the result.
- The output, by default, is streamed to the client in such a way that line feeds, extra spaces, and so on, are not included.

The following PROC STREAM program produces the output HTML file shown in Figure 1 (as seen in the browser) and Figure 2 (the HTML source):

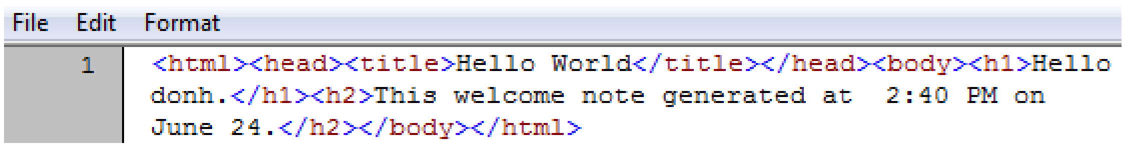
```
proc stream outfile=_webout;
BEGIN
<html>
<head><title>Hello World</title></head>
<body>
<h1>Hello &sysuserid..</h1>
<h2>
This welcome note generated at
%sysfunc(time(),timeampm8.) on
%sysfunc(date(),monname.) %sysfunc(date(),day.).
</h2>
</body></html>
;;;;
run;
```



Hello donh.

This welcome note generated at 2:40 PM on June 24.

Figure 1. Hello World Output

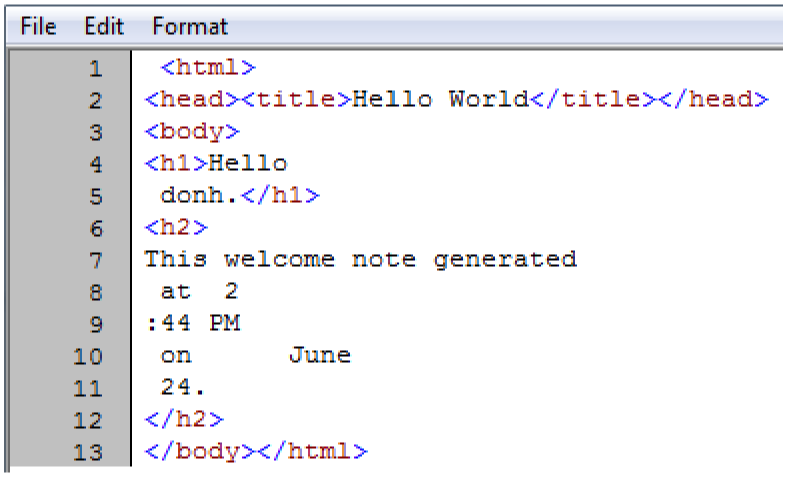


```
<html><head><title>Hello World</title></head><body><h1>Hello
donh.</h1><h2>This welcome note generated at  2:40 PM on
June 24.</h2></body></html>
```

Figure 2. Hello World Source HTML

Note that the output is a single (wrapped) line without the input embedded newline characters. If formatting of the generated HTML (for example, for readability) is desired, there are several options that can be used. If the ASIS option is added to the PROC STREAM statement, the generated HTML can be seen in Figure 3 (the appearance in the browser is the same as seen in Figure 1).

Note, however, that any text generated by a macro facility feature (for example, whenever the % or & are seen as macro triggers) causes the output to include extra newline characters.



```
 <html>
<head><title>Hello World</title></head>
<body>
<h1>Hello
 donh.</h1>
<h2>
This welcome note generated
 at  2
:44 PM
 on       June
 24.
</h2>
</body></html>
```

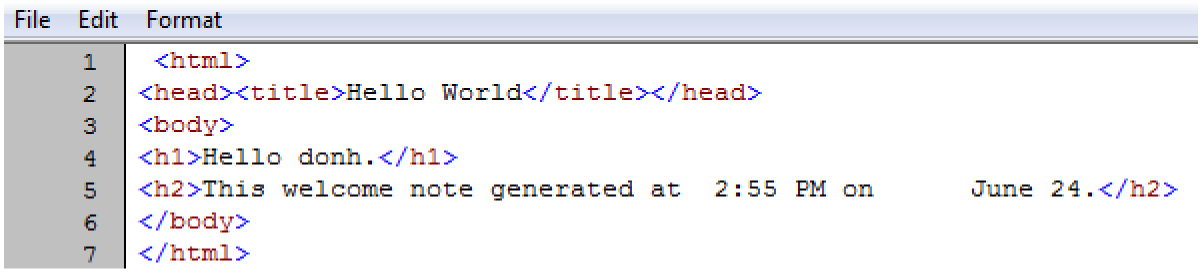
Figure 3. HelloWorld Source HTML Generated with ASIS Option

PROC STREAM provides another way for a programmer or developer to preserve newline characters in the generated output: using the facilities offered by the streamDelim macro variable. You can force a newline character to be included in the generated output by including the following text in the input to PROC STREAM:

```
&streamDelim newline;
```

The following program produces the HTML seen in Figure 4 (the appearance in the browser is the same as seen in Figure 1):

```
proc stream outfile=_webout; BEGIN
<html>
&streamDelim newline;<head><title>Hello World</title></head>
&streamDelim newline;<body>
&streamDelim newline;<h1>Hello &sysuserid..</h1>
&streamDelim newline;<h2>
This welcome note generated
 at %sysfunc(time(),timeampm8.)
 on %sysfunc(date(),monname.) %sysfunc(date(),day.).
</h2>
&streamDelim newline;</body>&streamDelim newline;</html>
;;;;
run;
```



```
 <html>
<head><title>Hello World</title></head>
<body>
<h1>Hello donh.</h1>
<h2>This welcome note generated at  2:55 PM on       June 24.</h2>
</body>
</html>
```

Figure 4. HelloWorld Source HTML Generated with &streamDelim newline;

Note these additional points of interest about this example:

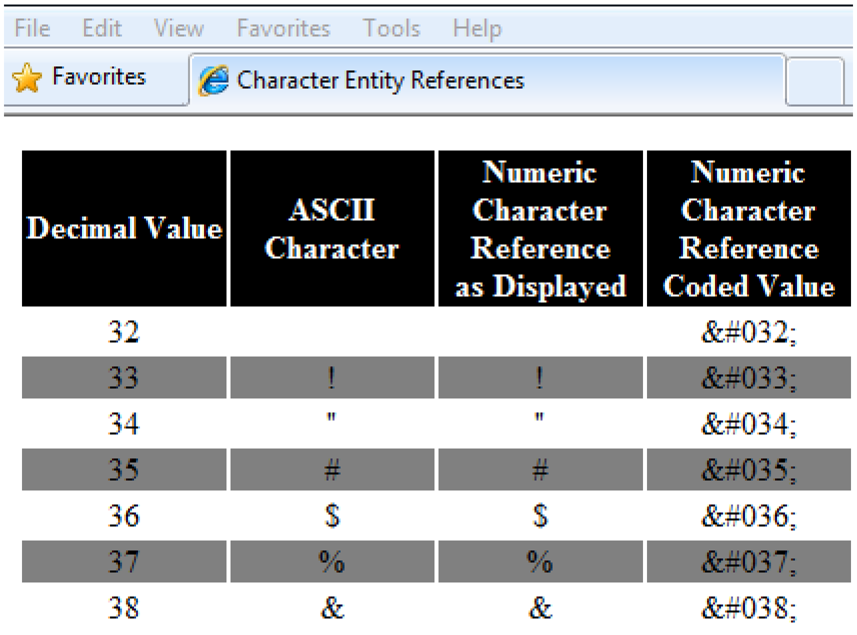
- The text &streamDelim newline; can be anywhere in the input stream (that is, it need not be at the beginning of a line).
- When the HTML is formatted by the browser, the extra spaces generated by the references to the date and time functions are ignored.

MACROS FOR ITERATIVE AND CONDITIONAL TEXT GENERATION

The sample here illustrates using the SAS macro language for text generation. The code in this case generates the HTML shown in Figure 5. Note that this HTML could be written in virtually any scripting language. It is included here simply to demonstrate the use of the SAS macro language to generate text other than SAS code.

This example highlights a number of features and capabilities where PROC STREAM gives the programmer or developer complete control over the look and feel of the page:

- Embedding HTML text inline along with SAS macro language statements to control the generated HTML text.
- Generating table rows using a %DO loop.
- Toggling the value of a macro variable using the %IF statement in order to create a striped table display.
- Executing SAS functions using the %SYSFUNC and %QSYSFUNC macro functions.
- Providing complete control over such HTML content as colors and table attributes. Note that some of the values are hardcoded (the table cell width), whereas the gray and white for the background colors of alternating rows are specified as macro parameters.



Decimal Value	ASCII Character	Numeric Character Reference as Displayed	Numeric Character Reference Coded Value
32			&#032;
33	!	!	&#033;
34	"	"	&#034;
35	#	#	&#035;
36	$	$	&#036;
37	%	%	&#037;
38	&	&	&#038;

Figure 5. Macro Generated Table of HTML Entities

Note that the Numeric Character Reference as Displayed column in Figure 5 is the resolved or display value of the actual numeric entity text (for example, **&#060;** is the value, but the display value is **<**).

In addition to illustrating iterative and conditional processing, this example also introduces HTML Entities (also known as Character Entity References and Numeric Character References) as they are an effective technique when generating HTML using SAS. For example, the less than (**<**) and greater than (**>**) characters are triggers for HTML tags. As a result, they should not be included as text in an HTML page. Typically, Web and HTML developers handle this situation by using a Character Entity Reference: **&lt;** for **<** and **&gt;** for **>**. However, this use of the & character forces the macro facility to attempt to resolve **&lt** and **&gt**. Fortunately, a simple workaround is to use Numeric Character References. Numeric Character References take the form &#nnn; where nnn is a number. Since #nnn is not a valid SAS name, the macro language does not interpret a numeric character reference as a macro variable reference.

The following macro code generated the output in Figure 5. It uses an iterative %DO loop to generate an HTML table with rows corresponding to the ASCII character for decimal values between 32 and 255. A %IF statement is used to alternate the background colors of the table rows. Four columns are shown:

- The decimal value corresponding to the character
- The ASCII character

- The display value for the Numeric Character Reference
- The coded Numeric Character Reference corresponding to the previous column

```
%macro characterSet
    (evenRowBGColor = grey
    ,oddRowBGColor = white
    );
 %local i bgcolor char;
 <html>
  <head><title>Character Entity References</title>
  /* Define the style to use for the table cells so that the values need
     not be specified for each cell. */
  <style>
   td {text-align:center; width:100px;}
   th {text-align:center; width:100px;}
  </style>
  </head>
  <body>
  <div>
  <table>
   /* Define table header information */
   <thead>
   <tr style="background:black; color:white">
     <th>Decimal Value</th>
     <th>ASCII Character</th>
     <th>Numeric Character Reference<br>as Displayed</th>
     <th>Numeric Character Reference<br>Coded Value</th>
   </tr>
  </thead>
  <tbody>
 /* Use a macro %DO loop to generate the values from 32 to 255. Start at 32 since
    some of the unprintable values below 32 cause problems) */
 %do i = 32 %to 255;
    %if &bgcolor ne &oddRowBGColor
        %then %let bgcolor = &oddRowBGColor;
        %else %let bgcolor = &evenRowBGColor;
    <tr style="background:&bgcolor">
    <td>&i</td>
    %let char = %qsysfunc(byte(&i));
    <td>&char</td>
    <td>&#%qsysfunc(putn(&i,z3.));</td>
    /* The "Numeric Character Reference as Displayed" column is generated by
       prefixing &# to the decimal value. The "Numeric Character Reference Coded
       Value" uses &#038; as the prefix instead of &, so the coded valued is
       displayed by the browser. If the value was generated using & (as in the
       prior column), the value seen in the browser would be the same. */
    <td>&#038;#%qsysfunc(putn(&i,z3.));</td>
    </tr>
 %end;
 </tbody>
 </table>
 </div>
 </body>
</html>
%mend characterSet;

proc stream outfile=_webout;
BEGIN
%characterSet(evenRowBGColor=white,oddRowBGColor=grey)
;;;;
```

For most character values, there is no need to worry about HTML encoding them. However, there are some characters (as mentioned above **<**, **>**, **&**, and a few others) for which such encoding is either required or desirable. There are also some characters corresponding to decimal values above 127 (for example, the registered trademark symbol, the copyright symbol, and so on), that are commonly used in HTML pages but are not available on standard keyboards. Using HTML entities for such characters should be considered a *Best Practice*. The reader might want to run the code for this example and save the generated HTML file as a reference table. Simply consult the table and use the value in the last column to get the numeric entity (such as **&#060;** for **<** and **&#062;** for **>**. There are numerous resources that discuss HTML Entities and provide a good overview of how and when they should be used.

INCLUDING TEXT FROM EXTERNAL FILES

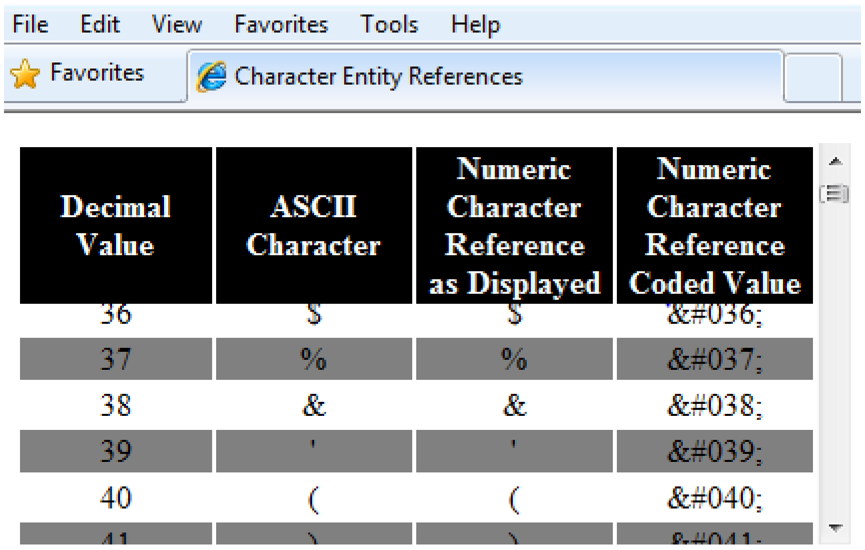
The examples shown so far have used inline text to be processed by PROC STREAM inline. Typically, however, the input text will be in an external file. The syntax for including an external file is as follows:

```
&streamDelim; %include [file-specification];
```

The following program is a revised version of the example above:

```
proc stream outfile=_webout;
BEGIN
&streamDelim; %include srvrpgs(characterSet.html);
;;;;
```

The output from running this program is shown in Figure 6.


File Edit View Favorites Tools Help
Favorites
Character Entity References

Decimal Value	ASCII Character	Numeric Character Reference as Displayed	Numeric Character Reference Coded Value
36	$	$	&#036;
37	%	%	&#037;
38	&	&	&#038;
39	'	'	&#039;
40	(	(	&#040;
41	)	)	&#041;

Figure 6. Using %INCLUDE to Define the Input SAS Server Page

This output also illustrates additional functionality for the generated output. The table is now scrollable with the headers locked. This is done by specifying a style sheet that causes the table header section to be frozen.

The following is the source for the characterSet.html input SAS Server Page:

```
%characterSet(styleSheet=scrollableTable.css)
```

Typically, input SAS Server Pages contain more than just a macro call. This input SAS Server Page has only a macro call because the macro generates all the needed HTML. The macro does have a new option, the name of a style sheet to use. That style sheet is included using another facility of the streamDelim macro variable in PROC STREAM, which is the ability to read in a file while preserving line feeds and suppressing macro resolution. This feature is typically used to include input SAS Server Pages that contain JavaScript code. The revised macro source is:

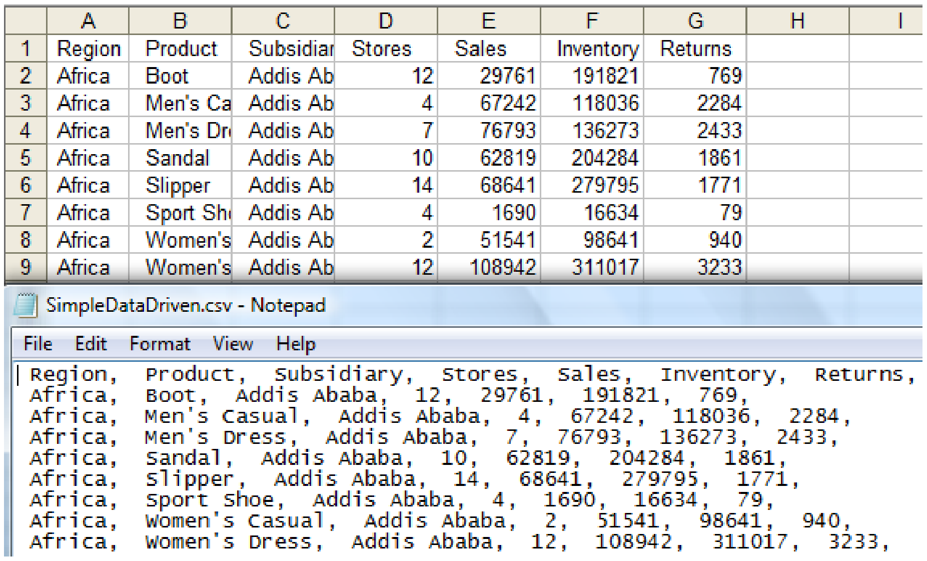
```
%macro characterSet
    (evenRowBGColor = grey
    ,oddRowBGColor = white
    ,styleSheet=   /* add a stylesheet parameter */
    );
 %local i bgcolor char;
 <html>
```

```
  <head><title>Character Entity References</title>
  %if %length(&styleSheet) gt 0 %then
     %str(&streamDelim readfile srvrpgs(&styleSheet););
  </head>
  <body>
  <table>
   <thead>
   <tr>
     <th>Decimal Value</th>
     <th>ASCII Character</th>
     <th>Numeric Character Reference<br>as Displayed</th>
     <th>Numeric Character Reference<br>Coded Value</th>
   </tr>
  </thead>
  <tbody>
 %do i = 32 %to 255;
    %if &bgcolor ne &oddRowBGColor %then %let bgcolor = &oddRowBGColor;
    %else %let bgcolor = &evenRowBGColor;
    <tr style="background:&bgcolor">
    <td>&i</td>
    %let char = %qsysfunc(byte(&i));
    <td>&char</td>
    <td>&#%qsysfunc(putn(&i,z3.));</td>
    <td>&#038;#%qsysfunc(putn(&i,z3.));</td>
    </tr>
 %end;
 </tbody>
 </table>
 </body>
</html>
%mend characterSet;
```

The following best practices are strongly encouraged:

- Using aggregate syntax to define the input files (referred to in this book as SAS Server Pages) that should be processed by PROC STREAM. Aggregate syntax enables you to assign a fileref to a directory and then work with any file in that directory by specifying its filename in parentheses after the fileref.
- Storing any macros referenced in SAS Server Pages in an autocall directory.

GENERATING DYNAMIC CONTENT FROM A SAS DATA SET



	A	B	C	D	E	F	G	H	I
1	Region	Product	Subsidiar	Stores	Sales	Inventory	Returns		
2	Africa	Boot	Addis Ab	12	29761	191821	769		
3	Africa	Men's Ca	Addis Ab	4	67242	118036	2284		
4	Africa	Men's Dr	Addis Ab	7	76793	136273	2433		
5	Africa	Sandal	Addis Ab	10	62819	204284	1861		
6	Africa	Slipper	Addis Ab	14	68641	279795	1771		
7	Africa	Sport Sh	Addis Ab	4	1690	16634	79		
8	Africa	Women's	Addis Ab	2	51541	98641	940		
9	Africa	Women's	Addis Ab	12	108942	311017	3233		

```
Region, Product, Subsidiary, Stores, Sales, Inventory, Returns,
Africa, Boot, Addis Ababa, 12, 29761, 191821, 769,
Africa, Men's Casual, Addis Ababa, 4, 67242, 118036, 2284,
Africa, Men's Dress, Addis Ababa, 7, 76793, 136273, 2433,
Africa, Sandal, Addis Ababa, 10, 62819, 204284, 1861,
Africa, Slipper, Addis Ababa, 14, 68641, 279795, 1771,
Africa, Sport Shoe, Addis Ababa, 4, 1690, 16634, 79,
Africa, Women's Casual, Addis Ababa, 2, 51541, 98641, 940,
Africa, Women's Dress, Addis Ababa, 12, 108942, 311017, 3233,
```

Figure 7. Sample CSV File as Seen in Excel and Notepad

Many of the examples here have used the %SYSFUNC (or %QSYSFUNC) macro function to execute SAS or SCL functions outside of a DATA step. This technique can be used to access the information in SAS data sets. This allows SAS code and specifically SAS macros used in the input stream to PROC STREAM to access data set information, both attributes and data. Thus, PROC STREAM can generate data-driven HTML content without requiring the generation of any SAS code.

The example here uses a sample macro called %delimited (included below) that that accesses a SAS data set and produces a CSV file, including a header row shown in Figure 7. The point of this example is to illustrate accessing data as well as creating output content other than HTML. This use of this technique to create CSV files is just one of many ways to generate CSV files in SAS.

The following program generates this CSV file:

```
%let subsid = Addis Ababa; /* use a macro variable to define the subset of data
to be read */
proc stream outfile=_webout quoting=single;
BEGIN
%delimited
  (data=sashelp.shoes
  ,where=subsidiary=:"&subsid"
  )
;;;;
run;
```

The source for the sample %delimited macro is:

```
%macro delimited
    (data=
    ,where=
    ,delimiter=%str(,)
    );
 %local dsid i type value;
 %let dsid = %sysfunc(open(&data(where=(&where))));
 %do i = 1 %to %sysfunc(attrn(&dsid,nvars));
     %let value = %sysfunc(varname(&dsid,&i));
     %let value = %sysfunc(strip(&value));
&value&delimiter
 %end;
 %do %while(%sysfunc(fetch(&dsid))=0);
&streamDelim newline;
    %do i = 1 %to %sysfunc(attrn(&dsid,nvars));
       %let type = %sysfunc(vartype(&dsid,&i));
       %let value = %sysfunc(getvar&type(&dsid,&i));
       %let value = %sysfunc(strip(&value));
&value&delimiter
    %end;
 %end;
 %let dsid = %sysfunc(close(&dsid));
%mend delimited;
```

In addition to illustrating how to access and include data from SAS data sets, this example highlights several important features of PROC STREAM:

- The QUOTING=SINGLE option treats the single quotation mark (') as a character and does not consider it to be the delimiter for a quoted string. Thus, when a single quotation mark is seen in the selected values of Product (for example, Men's Casual, Women's Casual) it is treated as just another character.
- The program uses &streamDelim newline; to start each row of data on a new line. Although new lines are not important in markup languages such as HTML and XML, they are important in text files such as CSV files.
- Macro variables whose values can be set before the PROC STREAM input stream (for example, the `%let subsid= Addis Ababa;` statement) are resolved correctly, thus illustrating that macro variables can be used to pass parameters to the PROC STREAM input stream.
- Since spacing can sometimes be important, note how the %SYSFUNC macro calls that generate the variable names and values are not indented. This is done to prevent extra spaces in the output CSV file. Just like newlines, spaces can be important, depending on the type of file being generated.

EXECUTING SAS CODE IN A PROC STREAM INPUT FILE

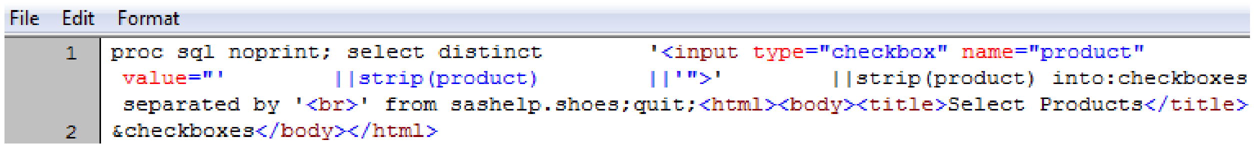
The facility to include SAS code that is to be executed by PROC STREAM can be an important feature.

Initially a developer might just include some executable SAS code in the program that calls PROC STREAM. An additional level of abstraction could be developed by incorporating generic routines within a SAS Server Page to perform specific types of functions. For example creating a data set of distinct values for a column in a SAS data set in order to create a dynamic selection list (either as a select tag or a series of check boxes) is a function that is commonly needed when creating a user interface to a stored process or SAS/IntrNet Application Dispatcher. By allowing code to be executed within a SAS Server Page, creating a single stored process to run PROC STREAM with different input files for specific stored processes (for example, reports) becomes an option.

The following example expands on the SAS Sample referenced in the introduction to this chapter. Consider the case of a reporting stored process where the user is allowed to select one or more products to include in the report. There are a number of options to present such a list in an HTML page. A series of check boxes or a selection list that allows multiple selections provide two examples. As mentioned previously, the desire is to execute some SAS code that is defined in an input SAS Server Page as follows:

```
proc stream outfile=_webout quoting=both;
BEGIN
<html>
<body>
<title>Select Products</title>
<body>
proc sql noprint;
 select distinct
        '<input type="checkbox" name="product" value="'
        ||strip(product)
        ||'">'
        ||strip(product)
 into:checkboxes separated by '<br>'
 from sashelp.shoes;
quit;
&checkboxes
</body>
</html>
;;;;
run;
```

When this code is run, the output is not as expected. As shown in Figure 3.8, the SAS code is simply included as text in the generated output.



```
proc sql noprint; select distinct        '<input type="checkbox" name="product"
 value="'         ||strip(product)        ||'">'        ||strip(product) into:checkboxes
 separated by '<br>' from sashelp.shoes;quit;<html><body><title>Select Products</title>
&checkboxes</body></html>
```

Figure 8. First Attempt to Include SAS Code in a SAS Server Page

The problem is that PROC STREAM treats all the generated text as text to be included in the output file. Thus our SAS code is never executed. In order to address the need to include and execute SAS code in an input SAS Server Page, there are two new functions, DOSUB and DOSUBL (experimental in SAS 9.3 and production in SAS 9.4), that are described in the PROC STREAM documentation. The DOSUB function has a single argument that is a fileref that contains SAS code to be executed, and the DOSUBL function has a single character argument that is the SAS code to be executed. Sample syntax to use these functions in a SAS Server Page is as follows:

```
%let rc = %sysfunc(dosubl(
SAS Code - potentially spanning multiple lines
));
```

The following is a corrected (that is, the SAS code is executed and the resulting macro variable can be referenced to generate the check boxes) version of the preceding program:

```
proc stream outfile=_webout quoting=both;
BEGIN
<html>
<body><title>Select Products</title><body>
%let rc =%sysfunc(dosubl(
proc sql noprint;
```

```
     select distinct
            '<input type="checkbox" name="product" value="'
            ||strip(product)
            ||'">'
            ||strip(product)
     into:checkboxes separated by '<br>'
     from sashelp.shoes;
    quit;
    ));   /* End both the DOSUBL function call and the
    %SYSFUNC macro function */
    &checkboxes
    </body>
    </html>
    ;;;;
    run;
```

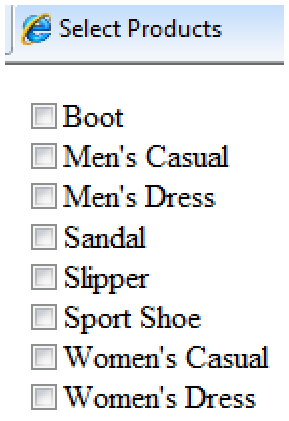


Figure 9. Corrected Output with Generated Check Boxes

The generated output is shown in Figure 9.

SELECTED EXAMPLES

Three examples of using SAS Server Pages in a batch environment are described here. While the origins of SAS Server Pages, and thus the name, was to simplify the generation of HTML in Stored Processes and SAS/IntrNet Application Dispatcher programs, they can also be used in batch processing that do not involve any of the SAS web applications or tools. So first let's clarify what is meant by *in batch*:

- jobs run on a scheduled or ad-hoc basis in the background (e.g., overnight job to produce reports)
- code submitted interactively by a SAS user/programmer in SAS Display Manager
- code submitted interactively by a SAS user/programmer in SAS Enterprise Guide.

Numerous additional examples are available in the author's SAS Press book SAS Server Pages: Generating Dynamic Contact and blog Jurassic SAS® in the BI/EBI World"

This presentation will focus on three examples:

- A sample mail-merge application
- Creating a custom calendar
- Displaying data graphically, e.g., the use of Tag Clouds as an alternative to bar or pie charts

A SAMPLE MAIL-MERGE APPLICATION

This example illustrates a well-known use case—a mail-merge where the requirement is to generate custom content for each defined subset of data, usually each observation in a data set. Such observations can be individual observations in a SAS data set or aggregated summary statistics for each value of a class variable. The sample presented here is the creation of a custom document (i.e., a letter) as an HTML file for each observation in the well-known sashelp.class data set that is shipped with SAS software. There are numerous other potential use cases that can be implemented using a comparable approach, for example, producing:

- a custom formatted report for each department or product
- a case report form for clinical trials data in a pharmaceutical environment
- and more

Let's first look at the very simple input SAS Server Page (class.html) which contains the desired text for the letter as HTML with the values to be *merged* into the letter referenced as macro variables. The macro references are highlighted in blue here to make them easier to spot.

- Note that the macro variables have the same name, by convention, as the corresponding variables in the data set. In the code (shown below) formats are used for the variables. This allows the value of age, for example, to be displayed in words (e.g., eleven instead of 11). Likewise the values of the variable sex are formatted as son/daughter for M/F.
- The %sysfunc macro is used to insert the date and time into the letter.

```
<html>
<head>
```

```
<title>&Name</title>
</head>
<body>
%sysfunc(date(),worddate.)
<br><br>
<b>Dear Parents</b>:
<p>As &name's teacher I wanted to make you aware of a project
 we are doing this year.
<p>Every month we will be collecting your &age year old &sex's height and weight
 and recording it. At year end, &name will be asked to produce a chart of their
 growth over the school year.
<p>For your information, here are the measurements that we just collected:
<ul><li><b>Height:</b> &height inches.
    <li><b>Weight:</b> &weight pounds.
</ul>
<p>Please don't hesitate to contact me if you have any questions,
<p>Regards,
<br><br>
&name's teacher
</body>
</html>
```

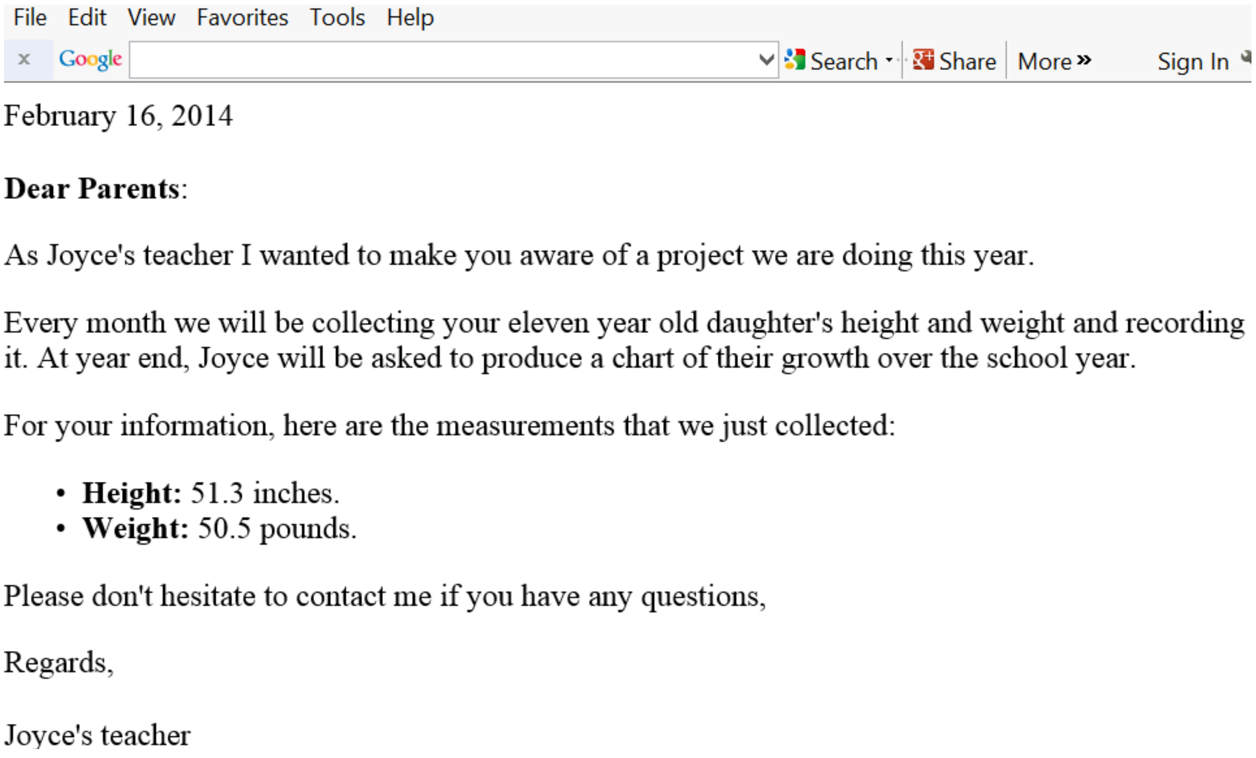
Next, let's look at the program that generates the letters.

- We first define a format that maps the value of the variable sex.
- Next, we determine how many letters need to be generated.
- Then a macro %DO loop is used to execute a DATA Step and PROC STREAM for each observation in the input data set
- Each DATA Step execution reads just one observation from the data sets and loads the formatted values of the variables into like-named macro variables.
- A filename statement is used to define the location for the generated letter. Note that by convention I typically use a macro variable, &root, to define the base path for any project.
- And finally PROC STREAM is called with the quoting=single option since our input SAS Server Page contains single quote signs that are to be treated as apostrophes instead of the marker for a text string.

```
proc format;
 /* create a format that maps the value of sex to daughter/son */
 value $gender 'F' = 'daughter'
               'M' = 'son'
 ;
run;

%macro generateDocument;

 %local numDocuments obsToRead;

 /* get the number of letters (i.e., observations to use) to generate */
 %let numDocuments=0; /* ensure a default value */
 proc sql noprint;
  select count(*) into:numDocuments
  from sashelp.class;
 quit;

 /* loop thru the number of observations and execute the DATA Step and
    PROC STREAM once for each observation in the data set */
 %do obsToRead = 1 %to &numDocuments;
    data _null_;
     /* Read the observation and create macro variables for that observation.
        The vvalue function is used to cause the formatted values to be used. */
        set sashelp.class (firstobs=&obsToRead obs=&obsToRead);
     /* associate the desired formats with sex and age*/
```

```
       format sex $gender. age words12.;
       call symputx('Name',vvalue(Name),'L');
       call symputx('Sex',vvalue(Sex),'L');
       call symputx('Age',vvalue(Age),'L');
       call symputx('Height',vvalue(Height),'L');
       call symputx('Weight',vvalue(Weight),'L');
      run;

      filename sspout "&root\output\&Name..html" lrecl = 32767;

      /* invoke PROC STREAM with quoting=single so the single quote character
         in the input SAS Server Page is treated as an ordinary character. */
      proc stream outfile=sspout quoting=single;
       /* use %include to define the input SAS Server Page */
       BEGIN
       &streamDelim;%include srvrpgs(class.html);
       ;;;;
      run;
   %end;

  %mend generateDocument;

  %generateDocument
```

Figure 10 shows the output for one of (for Joyce) the generated letters.



February 16, 2014

Dear Parents:

As Joyce's teacher I wanted to make you aware of a project we are doing this year.

Every month we will be collecting your eleven year old daughter's height and weight and recording it. At year end, Joyce will be asked to produce a chart of their growth over the school year.

For your information, here are the measurements that we just collected:

- **Height:** 51.3 inches.
- **Weight:** 50.5 pounds.

Please don't hesitate to contact me if you have any questions,

Regards,

Joyce's teacher

Figure 10. Sample Generated Letter

An additional mail-merge example (as well as possible future examples) that uses the **DOSUBL** function in the DATA step instead of macro looping can be found on the author's blog.

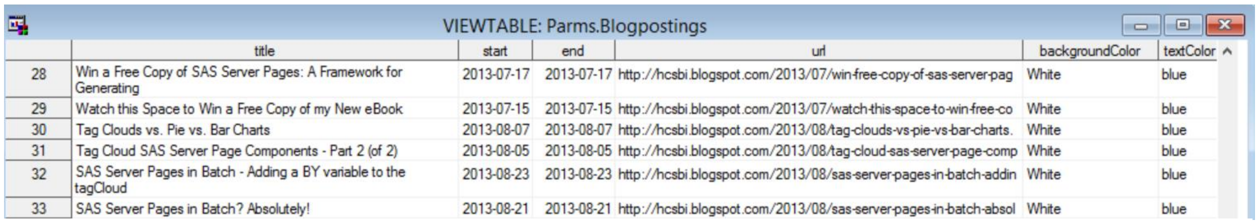
CREATING A CUSTOM CALENDAR

This example (and the next one) 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 born when a fellow consultant asked about publishing an events calendar on the SAS Portal. SAS PROC CALENDAR was not an option and so in researching options I found the FullCalendar plugin for JQuery. Creating a SAS Server Page to do that is discussed in the blog entry Leveraging JQuery Widgets - An Events Calendar. The example shown there illustrated using both the SAS Stored Process Server and the SAS/IntrNet Application Dispatcher to produce the content. This paper will modify that example as follows:

- Run in batch as discussed above.
- The use of PROC JSON, new in SAS 9.4, to produce the JSON input (JavaScript Object Notation) used by the FullCalendar JQuery plugin.

The data for the example is a SAS data set of blog postings (see Figure 11). The FullCalendar plugin has lots of display options. The sample data shown in Figure 11 uses just a few of them:

- Title: the text to display.
- Start: the start date for the event.
- End: the end date for the even.
- URL: the url to display when the user clicks on the display text.
- BackgroundColor: the background color for the text
- TextColor: text color (only used for text that is not a hyperlink)


VIEWTABLE: Parms.Blogpostings

	title	start	end	url	backgroundColor	textColor
28	Win a Free Copy of SAS Server Pages: A Framework for Generating	2013-07-17	2013-07-17	http://hcsbi.blogspot.com/2013/07/win-free-copy-of-sas-server-pag	White	blue
29	Watch this Space to Win a Free Copy of my New eBook	2013-07-15	2013-07-15	http://hcsbi.blogspot.com/2013/07/watch-this-space-to-win-free-co	White	blue
30	Tag Clouds vs. Pie vs. Bar Charts	2013-08-07	2013-08-07	http://hcsbi.blogspot.com/2013/08/tag-clouds-vs-pie-vs-bar-charts.	White	blue
31	Tag Cloud SAS Server Page Components - Part 2 (of 2)	2013-08-05	2013-08-05	http://hcsbi.blogspot.com/2013/08/tag-cloud-sas-server-page-comp	White	blue
32	SAS Server Pages in Batch - Adding a BY variable to the tagCloud	2013-08-23	2013-08-23	http://hcsbi.blogspot.com/2013/08/sas-server-pages-in-batch-addin	White	blue
33	SAS Server Pages in Batch? Absolutely!	2013-08-21	2013-08-21	http://hcsbi.blogspot.com/2013/08/sas-server-pages-in-batch-absol	White	blue

Figure 11. Calendar Input Data

To get the FullCalendar plugin to display my calendar, I only have to pass the data to it using JSON. The following SAS Server Page does that.

Note that the following HTML (with interspersed explanatory text) was based on one of the FullCalendar examples. It was simply necessary to download it and modify it as needed so it could be used as a SAS Server Page with SAS data dynamically inserted.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<style>
```

Use the SAS readfile autocall macro (provided with PROC STREAM) to include the cascading style sheet for the FullCalendar plugin. The style sheet was downloaded and saved in an aggregate location containing SAS Server Pages.

```
&streamDelim;%readfile(srvrpgs,memname=fullcalendar.css,nl=no,tok=no)
</style>
```

Provide a link for the JQuery framework. There are numerous hosted sources for JQuery frameworks.

```
<script type='text/javascript' src='http://code.jquery.com/jquery-
1.7.1.min.js'></script>
```

Now include the FullCalendar JavaScript code. It has also been downloaded and saved.

```
<script type='text/javascript'>
&streamDelim;%readfile(srvrpgs,memname=fullcalendar.min.js,nl=yes,tok=no)
```

Define the attributes for the calendar.

```
$(document).ready(function() {
$('#calendar').fullCalendar({
      height: 350,
```

By default, upon display, the calendar will show the current day. In this case we want to show the month of the most recent blog entry. That is accomplished by using **DOSUBL** to query the data to get the year and month of the most recent postings as macro variables.

```
%let rc = %sysfunc(dosubl(%str(proc sql noprint; select year(start),
month(start)-1 into:yr,:mo from parms.blogpostings having
start=max(start);quit;)));

      year: &yr,
      month: &mo,
```

Define the event for what happens when the user clicks on the text - open a new window for the defined URL.

```
      eventClick: function(event) {
        if (event.url) {
            window.open(event.url);
            return false;
        }
      },
      header: {left: 'prev,next today',
               center: 'title',
               right: 'month,agendaWeek,agendaDay'
              },
      events:
```

We now need to include the JSON for the events in our data. We invoke PROC JSON using **DOSUBL** and have it write the JSON to a temp file, which is then included using the readfile option.

```
%let fref=jsontxt;
%let rc = %sysfunc(filename(fref,,temp));
%let rc = %sysfunc(dosubl(proc json out = jsontxt pretty nosastags; export
parms.blogpostings / fmtdatetime; run;));
&streamDelim readfile jsontxt;
});
});
</script>
<style type='text/css'>
body {text-align: center;
      font-size: 14px;
      Font-family: "Lucida Grande",Helvetica,Arial,Verdana,sans-serif;
     }
#calendar {width: 900px; margin: 0 auto; }
</style>
</head>
<body>
```

A standard header is included. Using readfile (or %include) for standard headers/footers should be considered a *Best Practice*.

```
&streamDelim readfile srvrpgs(hcsHeader.html);
<hr>
<h3 style="clear:both">SAS&#174; Server Pages: Generating Dynamic Content Blog
Postings Calendar</h3>
```

Define the display area for the calendar.

```
        <div id='calendar'></div>
```

Provide the appropriate acknowledgement for the use of the FullCalendar plugin.

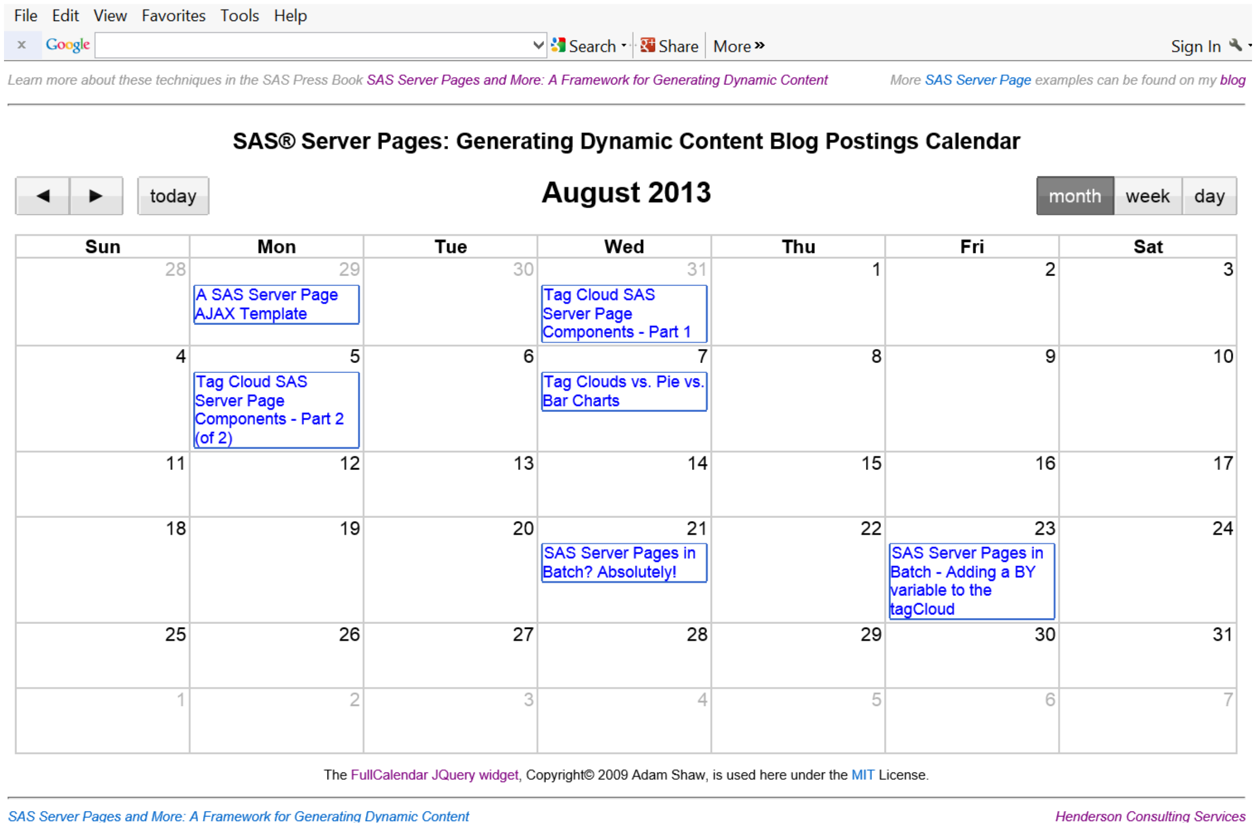
```
        <p style="height:15; font-family:Geneva, Arial, Helvetica, sans-serif; font-
        size:10px;">The <a href="http://arshaw.com/fullcalendar/">FullCalendar JQuery
        widget</a>, Copyright&#169; 2009 Adam Shaw, is used here under the
         <a href='/js/fullcalendar-1.5.3/MIT-LICENSE.txt'>MIT</a> License.
        <hr>
```

Display a standard footer.

```
        &streamDelim readfile srvrpgs(hcsFooter.html);
        </body>
        </html>
```

To generate the calendar seen in Figure 12, a simple PROC STREAM step that includes the SAS Server Page is all that is needed.

```
        proc stream outfile=sspout quoting=single;
         /* use %include to define the input SAS Server Page */
         BEGIN
         &streamDelim;%include srvrpgs(blogCalendarProcJson.html);
        ;;;;
        run;
```



Learn more about these techniques in the SAS Press Book SAS Server Pages and More: A Framework for Generating Dynamic Content *More SAS Server Page examples can be found on my blog*

SAS® Server Pages: Generating Dynamic Content Blog Postings Calendar

August 2013

Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29 A SAS Server Page AJAX Template	30	31 Tag Cloud SAS Server Page Components - Part 1	1	2	3
4	5 Tag Cloud SAS Server Page Components - Part 2 (of 2)	6	7 Tag Clouds vs. Pie vs. Bar Charts	8	9	10
11	12	13	14	15	16	17
18	19	20	21 SAS Server Pages in Batch? Absolutely!	22	23 SAS Server Pages in Batch - Adding a BY variable to the tagCloud	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

The FullCalendar JQuery widget, Copyright© 2009 Adam Shaw, is used here under the MIT License.

SAS Server Pages and More: A Framework for Generating Dynamic Content *Henderson Consulting Services*

Figure 12. Sample Blog Posting Calendar

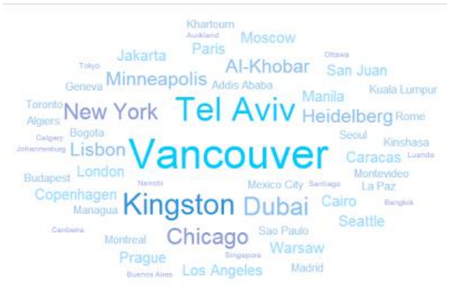
TAG CLOUDS AS AN ALTERNATIVE TO BAR OR PIE CHARTS

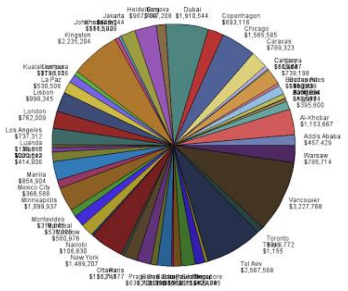
Tag clouds are commonly used in blog entries to highlight keywords and other data. They can also a great way to display BI data as illustrated in Figures 13-15 which display the sashelp.shoes data using:

- The Tag Cloud facility described here
- A Pie chart produced with SAS PROC GCHART

- A Bar chart produced with SAS PROC GCHART

Upon reviewing these three graphs, it seems intuitively obvious that the Tag Cloud provides a more meaningful display of the data. By scaling the actual text instead of either a rectangle or a pie slice, it is clear at first glance that the data for Vancouver, Tel Aviv and Kingston dominate the results.





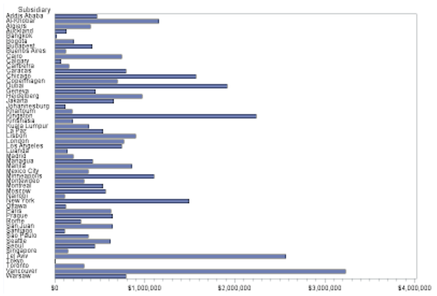


Figure 13. Tag Cloud **Figure 14. Pie Chart** **Figure 15. Bar Chart**

This example is fully discussed in the following blog entries and those details will not be repeated here:

- [Tag Cloud SAS Server Page Components - Part 1]
- [Tag Cloud SAS Server Page Components - Part 2 (of 2)]
- [Tag Clouds vs. Pie vs. Bar Charts]
- [SAS Server Pages in Batch? Absolutely!]
- [SAS Server Pages in Batch - Adding a BY variable to the tagCloud macro]

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). Additonal 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 using PROC STREAM and SAS Server Pages in BI environments: PROC STREAM and SAS Server Pages: Generating Custom User Interfaces

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