

Web Techniques For SAS ODS Output And More

Jules van der Zalm, OCS Consulting, Rosmalen, the Netherlands

ABSTRACT

SAS® programs can be defined as stored processes to output information to the SAS Information Delivery Portal®. They most commonly output static tables, listings or figures. When using HTML techniques within the stored process, dynamic content can be generated, for instance allowing the end-user to interactively browse through multiple dimensions of tables and listing. The approach for the development of such a program will be discussed in this paper. The approach consists of the following steps:

- Step 1: The objective;
- Step 2: The base program;
- Step 3: A dynamic program;
- Step 4: Creating a user interface;
- Step 5: Embedding the user interface;
- Step 6: Retaining the user selection.

INTRODUCTION

The purpose of this paper is to show how a SAS program can be used as a stored process in the SAS Information Delivery Portal to generate a dynamic summary table representing clinical data. This summary table is generated using PROC TABULATE combined with HTML techniques which allow the user to dynamically define the summary table by selecting in which way they want to analyse the data.

This paper will walk you through the process of creating such a SAS program step-by-step. For the example we are using dummy vital signs data, but of course the described techniques are applicable to any type of data that allows for tabulation.

STEP 1: THE OBJECTIVE

Before we can start to develop a program we need to establish the desired result. For this example we want to create an application that presents a tabular report of our vital signs dataset. The parameters to this tabular report are to be defined by the end-user, as intuitively as possible. Furthermore we want the user to be able to easily drill this table or modify the request. To achieve this end result we are going to break the development down into several logical steps.

STEP 2: THE BASE PROGRAM

The basis of our example is a tabulation program for vital signs data. The class variables to be used are 'Subject ID (SID)', 'Treatment Group (TREAT)', 'Position (POSITION)' and 'Day (DAY)'. As calculation variable we have 'Systolic Blood Pressure (SBP)', 'Diastolic Blood Pressure (DBP)' and 'Heart Rate (HR)'.

The tabulate statement has an accompanying ODS template which is suitable to be used for HTML, RTF, PDF, etcetera. This way we can maintain the ODS style and do not have to worry about the table not being presented correctly in HTML. The tabulate statement with the ODS statement looks like this:

```
ods path ocsstyle.customstyles (read) sashelp.tmplmst (read);
ods html style=TabulateReports file='.\test.html';

proc tabulate data=work.vitalsig noseps;
  keylabel sum=' ';
  class sid treat;
  var   sbp dbp hr;
  table sid, treat*(sbp dbp hr);
run;

ods html close;
```

When we run the program in the interactive SAS environment, this PROC TABULATE statement outputs this summary table to the result window.

PhUSE 2007

To use this specific program within the SAS Information Delivery Portal, it has to be converted to a stored process. This is done by placing the reserved macro statements *%stpbegin* and *%stpend* around the code that generates the output. Furthermore the ODS statement needs to be slightly modified. The ODS path statement remains, but the ODS HTML and ODS HTML close statements are removed, as they are 'replaced' by *%stpbegin* and *%stpend*. The style can be set by using the reserved macro keyword *_odsstyle*. When the program is ready, we have to make the program known to the metadata server by adding it to the BI Manager tree using the SAS Management Console.

The resulting code looks like this:

```
ods path odsstyle.customstyles (read) sashelp.tmplmst (read);
%let _odsstyle=TabulateReports;

%stpbegin;

proc tabulate data=work.vitalsig noseps;
  keylabel sum=' ';
  class sid treat;
  var    sbp dbp hr;
  table sid, treat*(sbp dbp hr);
run;

%stpend;
```

STEP 3: A DYNAMIC PROGRAM

Now the program can be used within the Information Delivery Portal, but it still produces identical output every time it's run. To make it more dynamic, user input is required. In its current form the program is not able to accept input. By replacing fixed variable names and values by macro variables, the program can receive input, for example, what vital signs parameters are displayed or where the treatment groups need to be presented, but it also gives the opportunity to 'change' the tabulate statement to generate dynamic tables.

So, the PROC TABULATE statement mentioned above is modified to:

```
proc tabulate data=&dsname noseps;
  keylabel sum=' ';
  class &dimensions;
  var    &statistics;
  table &tablestatement;
run;
```

The highlighted text shows the new macro variables which replace the fixed variables. The next step is to create an interface that allows the user to assign values to these variables.

STEP 4: CREATING A USER INTERFACE

The user interface provides the means for the user to choose which variables will represent the summary table's rows and columns and which vital signs calculation parameters will be presented. It consists of a collection of HTML elements which make up the input form.

The information from HTML input elements such as radio buttons, drop downs and checkboxes will automatically be available to the stored process in the form of macro variables. Our interface uses JavaScript to make it work more intuitively, but plain HTML will work as well. In order to dynamically generate the tabulate procedure, the following information is required:

- The CLASS variable(s) to present in the rows of the table, and the order in which to present them;
- The CLASS variable(s) to present in the columns of the table, and the order in which to present them;
- The VAR calculation variable(s) to include in the result table.

PhUSE 2007

To obtain this information we will let the user drag the class variables from a list of available class variables to the desired position on the row or column of the table. The order will be determined by the order of the items in the row or column list.

The calculation variables are in the centre of the table and can be selected by the user, who can specify to present the information in the row or in the column. The order of the calculation variables is fixed. When the user clicks on the button 'Create table', the stored process is called with this information.

This is what our interface looks like:

Next to table elements used to position the information this is what the interesting part of the HTML code looks like:

```
<!--List of available classes -->
<ul id='dimensions'>
  <li id='DAY'>Day</li>
  <li id='POSITION'>Position</li>
  <li id='SID'>Subject</li>
  <li id='TREAT'>Treatment group</li>
</ul>

<!--List of selected column classes -->
<ul id='columns'>
</ul>

<!--List of selected row classes -->
<ul id='row'>
</ul>

<!--List of available calculation variables -->
<br>Select calculation variables:<br>
<input type='checkbox' name='measures' value='DBP' unchecked>Diastolic blood
pressure<br>
<input type='checkbox' name='measures' value='HR' unchecked>Heart rate<br>
<input type='checkbox' name='measures' value='SBP' unchecked>Systolic blood
pressure<br>
<br>Present calculation in:<br>
<input type='radio' name='measure_position' checked value='column'>Column.
<input type='radio' name='measure_position' value='Row'>Row.
```

Because list items are not HTML input types they will not be converted to SAS macro variables in the stored process. To make sure we capture that information we use JavaScript to store the list information into a special HTML input

PhUSE 2007

type called 'hidden' to ensure the information is passed through to the stored process as macro variable. As an example we display the input type hidden to store the column information. This would result in the macro variable 'columnndimension' in the SAS program.

```
<!-- Storage for the selected columnndimensions -->

```

As you can see the value is currently empty. When the user clicks on the 'Create table' button the SAS stored process is called, but before we do that, JavaScript will obtain the variables present in the column dimension list in the order as defined on the screen. The following shows the JavaScript required to retrieve and store this information.

```
var SelectedColumns=' ';

var columnndimensions=document.getElementById('columns');
columnndimension=columnndimensions.getElementsByTagName('li');
for (var i = 0; i < columnndimension.length; i++)
{
    variableinfo=columnndimension[i].id.split('#');
    SelectedColumns = SelectedColumns + ' '+variableinfo[1];
}
document.Formname.columnndimension.value=SelectedColumns;
```

Within the SAS program we will manipulate the user input to define the tabulate statement so the user can create a table produced by our PROC TABULATE which will use the personal selection of row and column variables and vital signs parameters.

STEP 5: EMBEDDING THE USER INTERFACE

Now that we have created the user interface we can embed the user interface into the stored process using the SAS 'data _null_'-step with 'file _webout', which is a SAS reserved filename reference to output directly to the current HTML document. The HTML statement will be written to the web via PUT statements.

```
data _null_;
    file _webout;
    put "<!--List of vailable classes -->";
    put "<ul id='dimensions'>";
    /* etcetera */
run;
```

We have to ensure that the tabulate does not run when no information has been selected, so that the first time the stored process is executed, only the selection screen will be presented. Only when information is submitted from the user interface the tabulate report is presented, whilst above the report the user interface is still present on the screen.

STEP 6: RETAINING THE USER SELECTION

The final step in our program is to retain the user selection. With the user interface still present on the screen above the tabulate output and the HTML code output via SAS, we will have all the required information present in SAS macro variables allowing us to create the user interface and setting the previous user selection. This way, the user selection is not lost to the user, and the user can make changes to the selection shown.

To display the retained selection we have to:

- Store the received macro parameters into the equivalent HTML element of type 'hidden';
- Update the list items in the HTML table according to the user selection which is available to our stored process as a macro variable;
- Set the HTML elements such as checkboxes and radio buttons to their current selection.

PhUSE 2007

The following example stores the information into the input type hidden.

```
data _null_;
  file _webout;
  put "<!-- Storage for the selected columndimensions -->";
  put "<input type='hidden' name='columnndimension' value='&columndimension'>";
  /* etcetera */
run;
```

The following example will create the column list according to the selected variables:

```
data _null_;
  file _webout;
  put "<ul id='columns'>";
  %*****
  %* Present the dimensions available
  %*****

  %let i=1;
  /* htmlcolumns are the selected variable names in the correct order
     This macro variable was processed by the SAS program from the input received.
     The variable htmcolumnsdesc is the same variable, only containing
     the description */
  %let parameter1=%scan(%quote(&htmlcolumns),&i,%str(,));
  %let parameter2=%scan(%quote(&htmlcolumnsdesc),&i,%str(,));

  %*****
  %* Loop through the htmlcolumns macrovar until all items are processed.
  %*****
  %do %while(%quote(&parameter1) ne %quote());

    put "<li id='&parameter1'>&parameter2</li>";

    /* Retrieve the next variable in the list */
    %let i=%eval(&i+1);
    %let parameter1=%scan(%quote(&htmlcolumns),&i,%str(,));
    %let parameter2=%scan(%quote(&htmlcolumnsdesc),&i,%str(,));
  %end;
  put "</ul>";
  /* etcetera */
run;
```

The following example will set the radio button to select the calculation variable in the row or column:

```
data _null_;
  file _webout;
  %if %lowercase(&measurepos) eq row %then %do;
    put "<input class='radio' type='radio' name='measure_position'
        value='column'>Column. ";
    put " <input class='radio' type='radio' name='measure_position'
        checked value='Row'>Row. ";
  %end;
  %else %do;
    put "<input class='radio' type='radio' name='measure_position'
        checked value='column'>Column. ";
    put "<input class='radio' type='radio' name='measure_position'
        value='Row'>Row. ";
  %end;
  /* etcetera */
run;
```

PhUSE 2007

A COMPLETE EXAMPLE

Finally we present a small, ready to run SAS Program that uses a standard SAS dataset and demonstrates the elements discussed in this paper. The only thing to do is to register the stored process in the SAS metadata server in the 'Samples\Stored Processes' folder using the Management Console calling it 'example', all lower case. If you want to change this, you must change the highlighted section accordingly.

```
/* *****  
** Copyright OCS Consulting **  
*****  
* Program      : example.sas *  
* Type         : Stored process (web specific) *  
* Author       : Jules van der Zalm *  
* Location     : Phuse2007\Demo *  
* Date        : August 2007 *  
* Version     : 1.0 *  
* Description  : This program will demonstrate how to: *  
*               - Create dynamic output by accepting input *  
*               - Retain the information provided by the user *  
*****  
** Version control **  
*****  
* Mod * Ver. * Date      * Description *  
*****  
*****/  
  
/* *****  
* Ensure the expected variables exist by defining them globally.  
*****/  
%global displaytable displaygraph outputtitle;  
  
%macro processform;  
/* *****  
* Create the HTML page for the user input.  
*****/  
data _null_;  
  file _webout;  
  put "<html>";  
  put "<body>";  
  put "<form method='POST' action='/_SASStoredProcess/do' target='_self'>";  
  put "  <input type='hidden' name='_program' " value='SBIP://Foundation/Samples/Stored  
Processes/example (StoredProcess)'>";  
  put "  Please select the desired output type:<br>";  
  
  /* Retain the user setting for the table checkbox */  
  %if %lowcase(&displaytable) eq %lowcase(true) %then  
    put "    <input type='checkbox' name='displaytable' value='true' " checked>Table<br>";  
  %else  
    put "    <input type='checkbox' name='displaytable' value='true' " unchecked>Table<br>";  
  /* Retain the user setting for the graph checkbox */  
  %if %lowcase(&displaygraph) eq %lowcase(true) %then  
    put "    <input type='checkbox' name='displaygraph' value='true' " checked>Graph<br>";  
  %else  
    put "    <input type='checkbox' name='displaygraph' value='true' " unchecked>Graph<br>";  
  
  put "<br>Specify the title:<br>";  
  put "    <input type='text' size=40 name='outputtitle' " value='&outputtitle'><br><br>";
```

PhUSE 2007

```

put " <input type='submit' value='Create output'>";
put "</form>";
put "</body>";
put "</html>";
run;

%mend processform;
%processform;
/*****
* Set the title to be used
*****/
title "&outputtitle";

%macro generateOutput;
  %if %lowcase(&displaygraph) eq %lowcase(true) %then %do;

    %*****;
    %* Define the graphical options
    %*****;
    goptions xpixels=400 ypixels=200 ftext="Arial" htext=12.2 gunit=pt;

    %*****;
    %* Define the presentation of line plot
    %*****;
    symbol1 interpol=join height=10pt value=none cv=darkblue line=1 width=2;

    symbol2 interpol=join height=10pt value=none cv=darkblue line=4 width=2;

    legend1 frame;

    axis1 style=1 width=1 minor=none label=none;

    axis2 style=1 width=1 minor=none label=none;

    %*****;
    %* Create the actual line plot
    %*****;
    %stpbeg;

    proc gplot data = sashelp.class;
      plot weight * name=sex/ vaxis=axis1 haxis=axis2 frame legend;
    run;
    quit;

    %stpend;

  %end;

  %if %lowcase(&displaytable) eq %lowcase(true) %then %do;

    %*****;
    %* Create the table
    %*****;
    %stpbeg;

    proc print data = sashelp.class;
    run;

    %stpend;
  %end;
%mend generateOutput;

%generateOutput;

```

PhUSE 2007

CONCLUSION

Combining SAS programs and HTML techniques in the SAS Information Delivery Portal allows you to create powerful applications which are relatively quickly designed compared to a regular user interface. These kinds of programs allow users to intuitively browse through study data and perform quick analysis on lots of different types of data.

ACKNOWLEDGMENTS

I would like to take the opportunity to thank the following people for contributing to this paper:

Raymond Ebben, OCS Consulting

Bas van Bakel, OCS Consulting

CONTACT INFORMATION

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

Author Name	:	Jules van der Zalm
Company	:	OCS Consulting
Address	:	PO BOX 490 5240 AL Rosmalen the Netherlands
Work Phone	:	+31 (0)73 523 6000
Fax	:	+31 (0)73 523 6600
Email	:	sasquestions@ocs-consulting.com
Web	:	www.ocs-consulting.nl

Brand and product names are trademarks of their respective companies.