Paper 266-2008

Creating Interactive Web-Based Reports With SAS®

Chun Fan and Alex Ushveridze
Capella University, Minneapolis, MN

ABSTRACT

In this paper we discussed two methods of using SAS for creating simple web-based interactive reports which would allow technically non-experienced people to easily interact with them and instantly get answers to their specific high-level questions. One of the key features of these reports is their complete independence of any external data processing applications and platforms. These reports are standalone html documents which could be placed anywhere on the web or on local PC and do not need anything to be executed except a browser (like Internet Explorer). The goal can be achieved by embedding JavaScript file inside HTML using SAS. We compared two different ways of creating such interactive reports.

1 INTRODUCTION

One of the nice features of SAS is the possibility of creating interactive reports. Conceptually, it is very easy. Indeed, by using SAS language it is easy to create any external text file with any textual contents. In particular, this could be an HTML file. In order to add interactivity to that file it is sufficient to use any scripting language like JavaScript – which again is nothing else but a text. As a result – we get a dynamic HTML document which can be placed anywhere on the web or on local PC and accessed through any browser.

Of course, in comparison with many low-level languages like (C, SAS, Java) JavaScript is a relatively slow, however for not very intense calculations this should not be an issue. A big advantage of using JavaScript is its richness, simplicity, and platform neutrality and pure client-side nature – i.e. complete independence on any server-side and other supporting programs – the only thing one needs to run the JavaScript-based programs is a web-browser – and no matter which one. One can say with confidence that practically any types of dynamical and interactive reports not requiring too heavy computations can easily be created by using techniques described in this paper.

To practically create an HTML /JavaScript-based interactive output within SAS it is sufficient to create an external text file by using the _NULL_ data-step and then fill-up this file line-by-line with the corresponding HTML and/or JavaScript code by using PUT statement . SAS allows one to create such a file manually (by explicitly adding each line like in the sample code shown below):

```
data _null_;
      file "C:\myfile.html";
      put "..some text .. ";
      put "..some text .. ";
      put "..some text .. ";
      . . . . . .
run;
```

or automatically (actually copying the rows of the external SAS dataset and modifying them on fly):

```
data _null_;
      file "C:\myfile.html";
      set mydataset (keep = myvar);
      put " ... " myvar " ... ";
run;
```

For referencing data and metadata obtained in previous SAS calculations one can effectively use SAS macros.

In this paper we consider two different ways of generating HTML /JavaScript-based interactive reports within SAS environment.

The first way is rather universal and very flexible. It allows one to design any imaginable GUI. At the same time it requires some efforts in writing explicit HTML or JavaScript code which may produce some inconveniences for people not familiar with these programming languages.

The second way is less universal but much simpler. It is based on the use of the existing SAS ODS HTML commands. In fact, the way the ODS HTML works is similar to what we have described above. However, many details of HTML coding are hidden behind the ODS HTML commands.

Respectively, we consider two different examples.

For demonstrating how the first way works, we consider the problem of creating graphical interface which allows one to visualize the character of correlation of two dataset variables. The user can select two variables from two drop-down lists and get a 2D-graph showing the strength of the correlation between selected variable deciles.

For demonstrating how the second way works, we consider the problem of creating sortable interactive tables with embedded sortable graphics. The users can click on the column headers to sort either the numeric or graphical results. The sortability of a graphical data is a nice feature by itself and it considerably simplifies the high level analysis of data tables.

2 THE FIRST WAY: DIRECT SOLUTION

2.1 INPUT DATASET

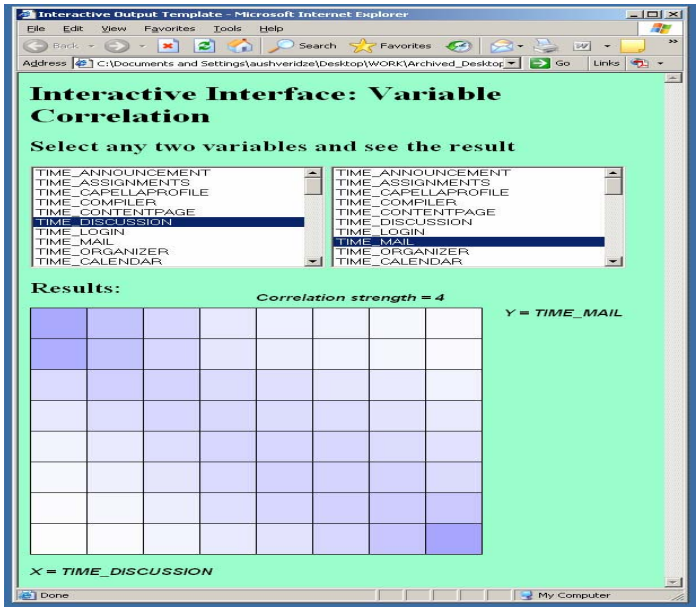
We start with a dataset having 5 columns (variables) and 50,000 rows (records). The sample portion of this dataset is shown below:

Key	Var1	Var2	Var3	Var4
…	…	…	…	…
K15640	2141	0	257	864
K15641	2311	60	5413	1277
K15642	11920	300	1700	1770
K15643	9937	120	1320	4047
K15644	58778	0	0	6607
K15645	53658	0	8981	5267
K15646	152924	120	16572	6656
K15647	23758	0	3192	810
…	…	…	…	…

The first variable is categorical and plays the role of ID, while the other four are numeric and represent data of our interest.

2.2 DESIRED OUTPUT

Our goal is to create an interactive graphical user interface which would allow one to find the character and strength of correlation between these four numeric variables. There are many ways of defining the correlation matrix. For the sake of simplicity, we select here a way which we think is the most universal and insensitive to possible variation of variable ranges. The idea is to replace the variable values with corresponding deciles which always range from 0 to 9 and, after that, count the number of observations corresponding to each pair of deciles. For each pair of variables we obtain a correlation matrix whose cells will contain the number of observations corresponding to different pairs of deciles. Our final goal is however to replace these numbers with different levels of a gray color – with the white color corresponding to no observations and the black color corresponding to maximal number of observations. We want to have a GUI which would allow a user to select two variables from two drop-down lists and after that immediately see the graph representing the correlation matrix for these variables. Here is how we want the GUI look like:



2.3 BEHIND THE SCENES

The corresponding HTML file is shown below:

```
<html>
<head>
<title>Interactive Output Template</title>
<script type='text/javascript' src='wz_jsgraphics.js'></script>
<script type='text/javascript' src='my_jslibrary.js'></script>
</head>
<body bgcolor='#99FFCC'>
<h1>Interactive Interface: Variable Correlation</h1>
<h2>Select any two variables and see the result</h2>
<p>
<select id='SelectA' size='10' onChange='ShowCorrelationMatrix(jg)' >
<option value=0 selected>var1</option>
<option value=1 >var2</option>
<option value=2 >var3</option>
<option value=3 >var4</option>
</select>
<select id='SeletB' size='10' onChange='ShowCorrelationMatrix(jg)' >
<option value=0 selected>var1</option>
<option value=1 >var2</option>
<option value=2 >var3</option>
<option value=3 >var4</option>
</select>
```

```
</p>
<h2>Results:</h2>
<p><div id='myCanvas' style='position: relative; top: 0px; left: 0px; width: 0px;
height: 0px'>
<script type='text/javascript' src='my_jslibrary2.js'></script>
</script>
</body>
</html>
```

The only JavaScript function responsible for processing the data is the ShowCorrelationMatrix(jg) function. This function is defined in an external text file 'my_jslibrary1.js' which can be placed in the same directory as the html file. To activate this function we need two additional external library files 'my_jslibrary2.js' and 'wz_jsgraphics.js' which can be placed at the same place. All these external files are general-purpose reusable text files which play the roles of libraries for the program and are not expected to change if the initial dataset changes. Actually, we do not need SAS to create these files.

2.4 THE ROLE OF SAS

There are only two non-trivial places where we actually need SAS code in order to associate the GUI we want to create with given datasets and variables. These are:

1. The portion of a SAS code which converts an initial SAS dataset containing original values of numeric variables into a number of JavaScript files containing all the information about these variables in the form of JavaScript arrays.
2. The portion of a SAS code which reads the variable names from the initial SAS dataset and creates corresponding drop-down menus.

However, before we discuss these two portions of SAS code, there is an auxiliary MACRO which creates the list of dataset variables.

EXAMPLE CODE

```
%macro nvarnames(library, dataset);
/* creating temporary dataset containing list of numeric variables of a given
input dataset;*/
   data numdataset;
       set &library..&dataset;
       length varname $ 32;
       array nums[*] _numeric_;
       do i = 1 to dim(nums);
              varname = upcase(vname(nums[i]));
              if varname ~= 'VARNAME' then output;
       end;
       keep varname;
       stop;
   run;
/* creating a macro array containing names of selected numeric variables;*/
  data _null_;
       set numdataset end = eof;
       n = left(put(_n_,3.));
       call symput("var"||n, varname);
       if eof then call symput("max", n);
  run;
/* deleting temporary dataset;*/
  proc datasets library = work;
       delete numdataset;
  quit;
%mend nvarnames;
```

THE FIRST MACRO PORTION. It creates external JavaScript files storing values of variables in the form of JavaScript arrays.

EXAMPLE CODE

```
%macro createjsvars(library, dataset, folder, quantile);
```

```
/* call macro for creating macro array of numeric variable names in input
dataset;*/
   %nvarnames(&library, &dataset);
/* enumerate rows to create key;*/
    data D0;
              retain N R;
              set &library..&dataset;
              N = _N_;
    run;
/* start looping over all numeric variables;*/
     %do n = 1 %to &max;
              %let input = &&var&n;
/* keep only the key and selected variable and sort by its value;*/
       proc sql;
              create table D1 as
              select N, &input
              from D0
              order by &nput;
       quit;
/* enumerate variable values in the increasing order;*/
       data D2;
              set D1;
              N_P = _N_ - 1;
       run;
/* create quantiles and drop all other variables except the original key;*/
       proc sql;
              create table D3 as
              select N, int(&quantile*N_P/count(1)) as Q_&input
              from D2
              order by N;
       quit;
/* write dataset to an external javascript file in the form of javascript array
definition;*/
       %let ind = %eval(&n - 1);
       data _null_;
               set D3 end=last;
               file "&folder.\src&ind..js";
               ind=resolve('&ind');
               if _n_ = 1  then put 'v[' _ind +(-1) '] = [ ';
               if not last then put Q_&input +(-1) ',';
               else put Q_&input +(-1) '];';
       run;
    %end;
/* delete temporary datasets;*/
       proc datasets library = work;
              delete  D0 D1 D2 D3;
       quit;
%mend createjsvars;
```

THE SECOND MACRO PORTION. This portion creates the final HTML file.

EXAMPLE CODE

```
 %macro createjshtml(library, dataset, folder, quantile);
   %nvarnames(&library, &dataset);
   %createjsvars(&library, &dataset, &folder, &quantile);

 /* we create reference for an external html file which has to be dynamically
 created in this macro */
   filename raw "&folder.\demo.html";
 /* we do not need to create any sas dataset -- therefore _null_ */
   data _null_;
      file raw;
 /* we create first row */
      ms="<!-- BEGINHTML -->
 /* we start adding rows to the external file */
      "; put ms; ms=" <html>
      "; put ms; ms=" <head>
```

```
      "; put ms; ms=" <title>Interactive Output Template</title>
/* first appearance of javascript -- adding reference to an external file
containing javascript graphics library */
      "; put ms; ms=" <script type='text/javascript'
                      src='wz_jsgraphics.js'></script>
/* adding our own javascript functions */
       "; put ms; ms=" <script type='text/javascript'
                      src='my_jslibrary1.js'></script>

/* continuing html */
       "; put ms; ms=" </head>
       "; put ms; ms=" <body bgcolor='#99FFCC'>
       "; put ms; ms=" <h1>Interactive Interface: Variable Correlation</h1>
/* creating drop-down menu interface */
       "; put ms; ms=" <h2>Select any two variables and see the result</h2>
       "; put ms; ms=" <p>
       "; put ms; ms=" <select id='var1' size='10' onChange='f()' >
       "; put ms;
/* first automatically generated drop-down box */
       %do n = 1 %to &nvars;
             %let varname = &&nvar&n;
             %let varnumb = %eval(&n - 1);
             %let sel = ;
             %if &n = 1 %then %let sel = selected;
             ms=" <option value=&varnumb &sel>&varname</option>";
             put ms;
       %end;
        ms=" </select>
        "; put ms; ms=" <select id='var2' size='10' onChange='f()' >
        "; put ms;
       %do n = 1 %to &nvars;
             %let varname = &&nvar&n;
             %let varnumb = %eval(&n - 1);
             %let sel = ;
             %if &n = 1 %then %let sel = selected;
             ms=" <option value=&varnumb &sel>&varname</option>";
             put ms;
       %end;
       ms=" </select>
       "; put ms; ms=" </p>
       "; put ms; ms=" <h2>Results:</h2>
       "; put ms; ms=" <p><div id='myCanvas' style='position: relative; top:
                       0px; left: 0px; width: 0px; height: 0px'>
       "; put ms; ms=" <script type='text/javascript'
                        src='my_jslibrary2.js'></script>

       "; put ms; ms=" </body>
       "; put ms; ms=" </html>
       "; put ms;
    run;

%mend createjshtml;
```

Finally we call these macros to allow them to do their work using the following statement.

```
%createjshtml(io, a, C:\Demo, 8);
```

3 THE SECOND WAY: INDIRECT SOLUTION BY USING ODS

3.1 INPUT DATASET

For demonstration, we consider a SAS dataset with three variables: Predictor, Importance and Shape. This is a typical dataset appearing as an the output of predictive modeling work and containing some basic properties of predictors.

Predictor	Importance	Shape
variable1	6.70	4321
variable2	4.50	1234
variable3	6.20	4312
variable4	5.30	3241

. . .

One of these properties is the Importance which shows how well does that predictor predict a target. The importance is traditionally well represented by a number – and nothing is wrong with that, but we want a little bit more: to make reading of the table easier we want to colorize the table cells containing these numeric values: the higher importance, the darker color.

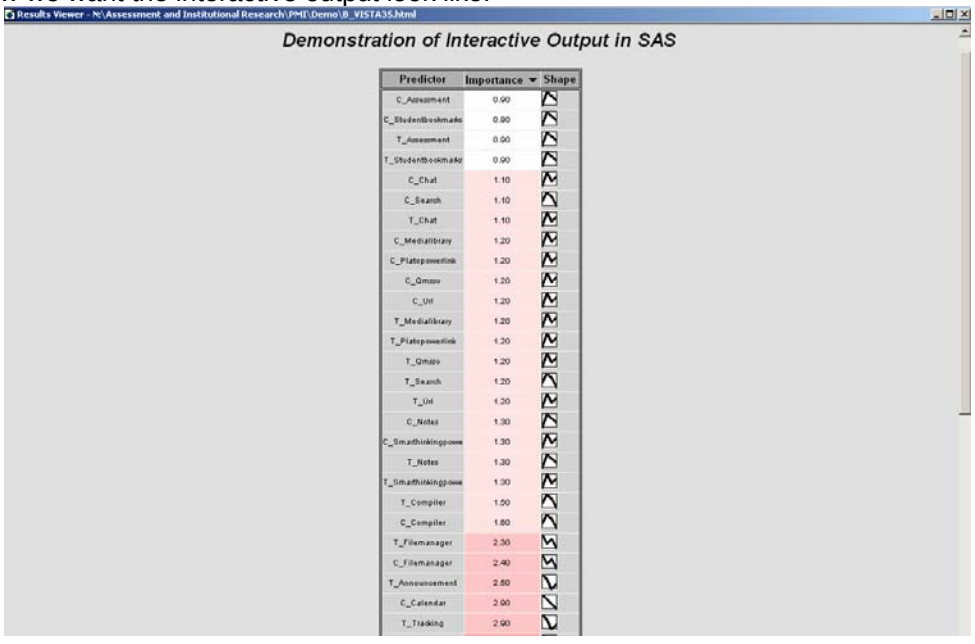
Another property is Shape, which is a symbolic representation of the relationship between target and predictor. Roughly speaking, if Shape is 1234, we have an increasing graph, if Shape is 4321, we have a decreasing graph, and if Shape is 1342, we have a slightly skewed bell-shaped graph. Again, these numbers do their work rather well but we want a little bit more – we want to have instead some real tiny graphs – we hope that this will make the output results better understandable for people preferring visual display.

3.2 DESIRED INTERACTIVE OUTPUT

Here is the list of desired features to include in the interactive output:

1. to make it a web-based output;
2. to replace numeric values of Shape variable with tiny graphs, at the same time hide the text content;
3. to colorize cells representing the Importance variable;
4. to make entire table sortable, for both numeric and graphical values.

Below is how we want the interactive output look like:



3.3. HOW IT WORKS

CREATING WEB-BASED OUTPUT. This can be done easily and quickly by using ODS HTML in SAS. The general form of ODS HTML for SAS 9.1 is as follows:

EXAMPLE CODE

```
ods html3 file='c:\temp\output.html' ;
  proc report data=originaldata nowd;
  run;
ods html3 close;
```

CREATING TINY GRAPHS. This stage can be done in four steps.

Step 1. Creating SAS dataset (table) with 24 columns and 4 rows which represent different shapes in both textual (variable names) and numeric (variable values) form. The example of this dataset named 'shape' is shown below:

1234	1423	2134	…
1	1	2	…
2	4	1	…
3	2	3	…
4	3	4	…

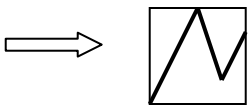
EXAMPLE CODE

```
data transformeddata;
   set originaldata;
   a=int(shape/1000);              /*get the first number*/
   b=int(shape/100)-a*10;          /*get the second number*/
   c=int(shape/10)-a*100-b*10;     /*get the third number*/
   d=mod(shape,10);                /*get the last number*/
 run;

proc sort data= transformeddata out=outdata nodupkey;          /*get rid of
duplicates */
     by shape;
run;
proc transpose data=outdata (keep=shape a b c d) out= transposed;
  id shape;
  var a b c d;
run;
```

Step 2. Use SAS MACRO to plot 24 combinations of "shape" values , and export the plots in .gif format. For example, for "shape" of "1423", plot "1423" on vertical axis as shown below.

1423
1
4
2
3



EXAMPLE CODE

```
%macro plot;
%local vars;
%let f = %sysfunc(open(transposed));
%if &f %then %do;
  %do i = 2 %to %sysfunc(attrn(&f,nvars));
    %let vars = %sysfunc(varname(&f,&i));
    /*export the plot in gif format in specified name and place*/
    filename grafout "c:\temp\&vars..gif";
    goptions reset=all device=gif hsize=0.20in vsize=0.20in
           gsfname=grafout gsfmode=replace noaxis;
```

```
     title;
    proc gplot data=transposed;
       plot &vars * _name_ / vaxis=axis1 haxis=axis2;
       axis1 label=none value=none major=none minor=none;
       axis2 label=none value=none major=none minor=none;
       symbol v=none i=join w=2;
    run;

    quit;

    filename grafout clear;

  %end;
 %let f = %sysfunc(close(&f));
 %end;
 %mend;
 %plot;
```

Step 3. Put shape values into .gif format using PROC FORMAT.

EXAMPLE CODE

```
proc format;
  value graph
    1234='_1234.gif'
    1243='_1243.gif'
    1324='_1324.gif'
    1342='_1342.gif'
    1423='_1423.gif'
    1432='_1432.gif'
    2134='_2134.gif'
    2143='_2143.gif'
    2314='_2314.gif'
    2341='_2341.gif'
    2413='_2413.gif'
    2431='_2431.gif'
    3124='_3124.gif'
    3142='_3142.gif'
    3214='_3214.gif'
    3241='_3241.gif'
    3412='_3412.gif'
    3421='_3421.gif'
    4123='_4123.gif'
    4132='_4132.gif'
    4213='_4213.gif'
    4231='_4231.gif'
    4312='_4312.gif'
    4321='_4321.gif'
   other = 'ne.gif';
```

Step 4. Set PREIMAGE= attribute in PROC REPORT like in this portion of code:

```
define shape /style(column)={ preimage=graph. }
```

Note: Full version of PROC REPORT code will be given in the last section.

HIDING THE TEXT. This can be done by modifying STYLE(COLUMN) = attribute in PROC REPORT as follows:

```
define shape /style(column)={ foreground=#D3D3D3}
```

Note: Full version of PROC REPORT code will be given in the last section.

COLORIZING CELLS REPRESENTING VARIABLES. This can also be achieved by formatting. For example, in this case, background color for the variable Importance will be changed based on their value ranges. First of all , we format background colors as follows.

EXAMPLE CODE

```
proc format;
    value bgi
      0-0.9='#FFFFFF'
      1-1.9='#FFE3E3'
      2-2.9='#FFC6C6'
      3-3.9='#FFAAAA'
      4-4.9='#FF8E8E'
      5-5.9='#FF7171'
      6-high='#FF5555';
run;
```

Then in PROC REPORT procedure, the style of those variables should be defined as the specified formats.

EXAMPLE CODE

```
proc report data= originaldata nowd ;
   define Importance / sum style(column)={background=bgi.} ;
run;
```

MAKING TABLE SORTABLE. This can be achieved by embedding an external JavaScript file sorttable.js inside ODS HTML. To make the table sortable, there are three steps:

Step 1. Download sorttable.js file from the JavaScript library (for more detail see ref. [1])

Step 2. Embed JavaScript code inside ODS HTML within the HEADTEXT= attribute, and include sorttable.js by putting a link to it in the head of your page.

Step 3. Add the sorting behavior to the table tag by giving it a class of "sortable" within the TAGATTR= attribute.

EXAMPLE CODE

```
ods html3 file='c:\temp\output.html'
   headtext='<SCRIPT LANGUAGE=JAVASCRIPT TYPE="TEXT/JAVASCRIPT"
   SRC="c:\temp\sorttable.js"></script>' ;
   proc report data= originaldata nowd
                  style(report)={tagattr=" class=""sortable"" "};
   column Predictor Importance Shape;
   define Predictor  / order order=data 'Predictor';
   define Importance / sum style(column)={background=bgi. foreground=black};
   define Shape / style(column)={ preimage=graph. foreground=#D3D3D3} ;
   run;
ods html3 close;
```

CONCLUSION

In this paper we created interactive web-based reports using SAS. These reports would allow technically non-experienced people to easily interact with them and instantly get answers to their specific high-level questions, and they are completely independent of any external data processing applications and platforms. The goal was achieved

by embedding JavaScript file inside HTML by using SAS. We discussed two different ways of creating such interactive reports. The first way is rather universal and very flexible. It allows one to design any imaginable GUI. However, it requires some efforts in writing explicit HTML or JavaScript code which may produce some inconveniences for people not familiar with these programming languages. The second way is less universal but much simpler. It is based on the use of the existing SAS ODS HTML commands.

REFERENCES

[1] Stuart Langridge. 2007. SortTable version 2. http://www.kryogenix.org/code/browser/sorttable/

[2] SAS Institute Inc. 2004. SAS® 9.1 Language Reference: Dictionary, Volumes 1, 2,and 3. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

Chun Fan
Capella University
225 S 6th Street 9th floor
Minneapolis, MN 55402
612 977 4273
Chun.Fan@capella.edu

Alex Ushveridze
Capella University
225 S 6th Street 9th floor
Minneapolis, MN 55402
612 977 5669
Alex.Ushveridze@capella.edu