Paper 173-27

# Let *SAS*® Build Your Dynamic Web Site from the Data!

Jennifer Sinodis, Bank One, Phoenix, AZ
Mark Moran, Bank One, Phoenix, AZ

## ABSTRACT

In today's environment, the web has become the best channel to view meaningful information. Let SAS do the work and produce your web output straight from the data! It can do most of the work involved in creating great looking, maintenance-free, dynamic web pages for your users. This paper will show how to use Version 8 SAS/Base to build static HTML output to your web site with HTML formatting macros and ODS (Output Delivery System) and how to use SAS/IntrNet™ to create output dynamically from your web site. It will also introduce JavaScript code and show how SAS/Base can produce JavaScript to generate drop-down menu options built directly from your data to access static HTML output, or create dynamic HTML.

*For the purpose of this paper most of the SAS programs shown are simplified. The paper illustrates SAS/Base macro code, HTML, ODS, JavaScript and SAS/IntrNet code. This paper assumes an experienced/advanced knowledge of SAS/Base Version 8 and some knowledge of HTML.*

Extension examples and code are available from our web site http://www.mgmdata.com/sas .

## INTRODUCTION

This paper will describe how to use SAS to build static or dynamic reports in HTML (Hyper Text Markup Language). Before we begin, let's define the difference between static reports and dynamic reports. Static reports are prewritten before they are accessed, such as reports created by a SAS program ran in a nightly batch job. Dynamic reports do not exist until they are requested, therefore, the SAS program does not run and create the report until a request is received from the user. For example, a static document is like accessing a book from a library, and a dynamic document is like a book that doesn't get written until you request it. We will also explain how to use SAS to build a dynamic menu system from your data to access static and/or dynamic HTML reports.

## USING SAS TO PRODUCE STATIC HTML

The most common language for web browsers such as Internet Explorer and Netscape is HTML. SAS provides several options to create HTML from SAS data. HTML uses tags such as <h1> and </h1> to structure text into headings, paragraphs, lists, hypertext links etc. If you are interested in learning HTML there are several good resources available.

### USING THE PUT STATEMENT

The simplest way to create HTML with SAS is to use the PUT statement. The PUT statement can be used to write lines to an external file. It can write lines containing variable values, character strings, or both.

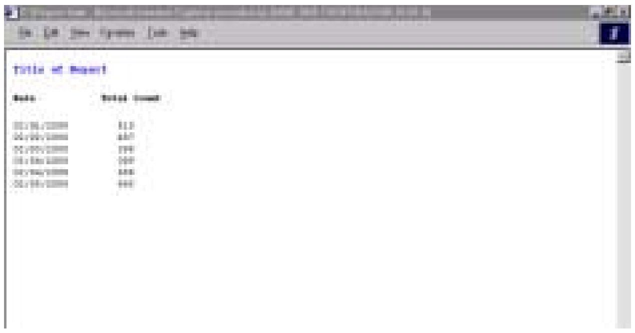
The following is an example of HTML.

```
<html><body bgcolor=white>
<pre><h3><font color=blue>Title of Report
</font></h3></pre>
<pre><strong>Date            Total Count</strong>
</pre><pre>02/01/2000         519
02/02/2000         497
…
02/05/2000         645</pre></body></html>
```

A SAS program could produce this HTML document with the following code:

```
  data _NULL_;
   File ‘c:\Report.html’;
   put ‘<html><head><body bgcolor=white>’;
   put ‘<pre><h3><font color=blue>
         Title of Report’;
   put ‘</font></h3></pre>’;
   put '<pre><strong>Date       Total Count
                         </strong></pre>';
   put '<pre>02/01/2000         519';
   …
   put '02/05/2000        645</pre>';
   put ‘</body></html>’;
  Run;
```

The browser interprets the HTML language and produces the following report.



```
Title of Report

Date          Total Count

[illegible]
```

Even though using the PUT statement is the simplest way to write HTML, unfortunately you need to know HTML. Luckily, SAS offers solutions in which the SAS programmer does not need to learn HTML.

### USING HTML FORMATTING MACROS

SAS offers four powerful HTML formatting macros that allow you to create HTML without having to learn the language: the output formatter (***%out2htm***), the data set formatter (***%ds2htm***), the tabulate formatter (***%tab2htm***), and the graph formatter (***%ds2graf***). The ***%out2htm*** macro converts standard SAS procedure output to HTML, the ***%ds2htm*** macro converts any SAS data set to HTML, the ***%tab2htm*** converts output from the SAS Tabulate procedure to HTML tables, and the ***%ds2graf*** converts graph output to HTML graphs.

The ***%out2htm*** formatting macro quickly converts existing standard SAS output code to HTML by sending captured SAS output to a file as HTML code. The ***%out2htm*** macro is applied by placing specific code before and after the standard SAS procedure output code.

The following program demonstrates how to use the ***%out2htm*** macro to convert a standard report to an HTML document. The ***%out2htm*** code needed is marked in bold in the sample code shown below.

```
%out2htm(capture=on);
  proc print data=SampleData label noobs;
    var day count;
    title ‘Title of Report’;
 run;
%out2htm(capture=off, htmlfile=’c:\Report.html’,
   tcolor=BLUE, bgtype=COLOR, bg=WHITE);
```

Essentially, this code calls the ***%out2htm*** macro and activates the capture mode, telling SAS to capture everything until the capture mode is turned off. SAS basically produces the same HTML sample code and results shown previously without the SAS programmer having to learn any HTML.

Obviously, if you want to build a report with SAS data, the formatting macros offer an easier solution than using the PUT statement. It builds the HTML and applies all the tags necessary.

This example also shows several parameters available with the ***%out2htm*** macro. Additional documentation about the HTML formatting macros is available through the SAS Institute at www.sas.com/rnd/web/intrnet/format/index.html.

### USING THE OUTPUT DELIVERY SYSTEM (ODS)

SAS/Base Version 7 introduced the Output Delivery System (ODS). This system has several new web publishing tools. The features of ODS were enhanced even more in SAS/Base Version 8.

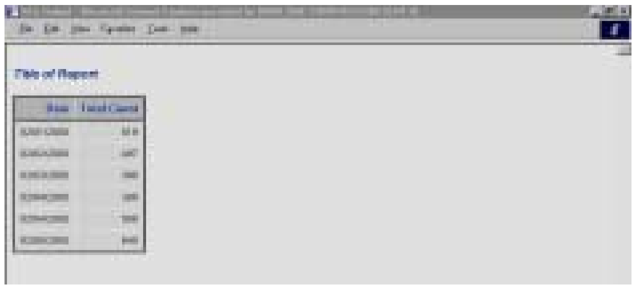
ODS creates more visually attractive reports. It controls the formatting of all output objects and transforms basic monospace output to enhanced visual output by allowing us to manipulate the color, style and fonts of the reports, as well as use style sheets and templates.

ODS is utilized by placing a few lines of code before and after the standard SAS procedure output code. ODS combines the resulting data from your ***PROC*** or ***DATA*** step with a template (or table definition) to several available destinations. Some of the destinations are: HTML, Listing, Output, RTF (rich text file), and PDF. The following program shows how to use ODS to convert the same ***PROC PRINT*** shown above to HTML. The ODS code is in bold.

```
ODS listing off;
ODS html file= “c:\Report.html”;
  proc print data=SampleData label noobs;
   … (same code as above )
  run;
ODS html close;
```

When using ODS we turn off the regular output listing window, and redirect our output to an HTML file. As you can see this code easily transforms standard SAS output to an HTML document, but the resulting HTML code and document in the browser look very different than ***%out2htm*** generated HTML.

The result of this HTML, as it appears in the browser, is shown below. This example is using the default template since we did not specify a template in our SAS program.



## USING SAS/INTRNET TO CREATE DYNAMIC HTML

The above examples showed how to use SAS to create static HTML documents, but with SAS/IntrNet Application Dispatcher 2.0 SAS can create HTML dynamically. SAS/IntrNet acts as a gateway between the web browser and SAS software, or a tool for executing SAS programs through a web interface. Basically, it allows a user to send a request from the web server to the SAS server to run a SAS program. In order to implement the SAS/IntrNet product you need to modify your configuration on your SAS server, as well as defining SAS/IntrNet program and data libraries. Once these changes are made a SAS program can be designed to execute upon receiving a request from the web server and then send the results back to the user.

Only a slight change needs to be made to the actual SAS program that produces the HTML in order to produce the HTML dynamically. Simply redirecting your output to **_webout** will allow SAS to place the resulting HTML back to the user who requested it. The necessary change is marked in bold below.

```
ODS listing off;
ODS html file= _webout;
 proc print data=SampleData label noobs;
   … (same code as above )
 run;
ODS html close;
```

Now that we have shown how to create static and dynamic HTML we want to show how you can use SAS to access these static and dynamic HTML reports from a web page with a menu access system. You could build a static HTML document with links for each HTML report you want to view, but this can create long lists of reports that become tedious to users who would have to scroll through all of the links in order to make their selection. To make menu pages easier, we suggest you use drop-down lists.

These drop-down lists can be built as static documents or dynamically from your data. Dynamic drop-down lists will give you the ability to limit your second selection based upon your first selection. This capability requires additional programming not available in HTML. You can use either VBScript or JavaScript to enhance the programming capabilities of HTML. We selected JavaScript due to its capabilities across browsers.

## JAVASCRIPT

*JavaScript is a unique programming language that functions only within a web browser. A full overview of the language is beyond the scope of this paper, but it is necessary to learn some basic JavaScript code and syntax in order to understand how to use SAS to build JavaScript.*

### INTRODUCTION TO JAVASCRIPT

JavaScript is a programming language that allows users to add interactive content to web pages. The browser reads the code and executes its instructions. JavaScript commands are included in the HTML code for a web page and are enclosed in ***<script>*** tags. The browser knows to run the JavaScript program because it’s enclosed in these tags. It only works with HTML elements and can enhance the interactivity of a web page by creating HTML dynamically, validating form fields and performing basic calculations.

JavaScript is essentially an object-based event-driven language. An object is selected, triggering an event and a piece of JavaScript code is executed. For example, when a user clicks a button on a web page, the button object is selected, triggering a “click” event which may activate certain instructions. For JavaScript purposes, objects are defined as computer entities that can be referenced in code. The major web objects are ***document***, ***elements***, ***form***, ***frame***, ***image***, ***link***, ***window***, ***history*** and ***navigator***. Each object consists of properties, methods and events.

A property is an attribute or characteristic of an object, and can be used to modify an object. Each object has its own specific properties. Some properties exist for several different objects, while other properties only exist for certain objects. For example, the ***document*** object has properties of ***title***, ***bgColor*** and ***fgColor***. A ***form*** object also has a property of ***title***, but not ***bgColor*** or ***fgColor***. Each object has several properties that describe it.

A method is a predefined action that an object can perform. Certain objects can perform certain methods. For example, the ***Write***(“string”) method is used with the ***document*** object, and requires a “string” parameter. This method writes the “string” to the current window.

JavaScript connects objects, properties and methods using the dot syntax notation. This notation consists of placing a period, or dot between the objects, properties and methods.

*Object property syntax:*

```
   Object.property = new value
   Example:document.MyForm.FirstOption.length = 0;
```

*Method syntax:*

```
   Object.method()
   Example: document.MyForm.FirstOption.focus();
```

Users interact with a web page by typing or clicking on the elements within it. These actions are called events. In JavaScript, event handlers process events. An event handler is a script or function that returns a True or False value. Event handlers are also predefined in JavaScript and are recognized by the event name preceded by the word “on,” such as ***onSubmit*** or ***onClick***. Certain event handlers are appropriate for certain objects. To use an event handler with an object it must be added to the HTML tag that defines the object.

*Event Handler syntax:*

```
  <TAG onEventName = “do a function()”>
  Example:
  <select name="FirstOption"
   onChange="FillSecondOption()"</select>
```

As mentioned earlier, JavaScript programs are included in the HTML code and are enclosed in the ***<script>*** tag. They can also be saved in separate files (MyJavaScript.js) and simply referenced inside the HTML program by the ***<script>*** tag. For example, if we create a JavaScript program called MyJavaScriptProgram.js we would include the following in our HTML code, generally at the beginning of the HTML file.

```
<script language=”javascript”
        src=”MyJavaScriptProgram.js”>
```

It is always important to put comments in your code, and this remains true for JavaScript programs. Comments are text or other characters ignored by the interpreter. Surprisingly, comments in JavaScript can be defined by using “/* */” combination, similar to the comment syntax in SAS and C++. It is also useful to know that JavaScript is extremely case sensitive.

## ADDING JAVASCRIPT TO OUR HTML

Now that we have briefly defined some JavaScript language, we want to show you how to build a menu page with a two level drop-down list. We will build the web page in HTML and reference a JavaScript program to tell the browser how to populate the drop-down list selection objects.

In order to communicate these concepts we will build a menu page that will access banking center reports that are located throughout the country in different markets The banking center drop-down list will be limited based on the market a user selects from the market drop-down list.

First, you need to build an HTML file with two drop-down lists--one for market selection and one for banking center selection. Next, define the events that you want to take place based on the user’s actions and decide the functions you want the browser to perform. For our example we want the browser to perform the following actions:

- Set up the web page after a user enters it
- Limit the selection list of banking centers available after a user selects a market from the market drop-down list
- Display the report of the banking center level chosen when the user selects the “Display Report” button

We named these functions Setup(), FillSecondOption() and DisplayReport().

The following HTML code shows the JavaScript references in bold with a brief description in italics.

```
<html>
<head>
<title>Retail Banking Reporting</title>
<script LANGUAGE="JavaScript"
     SRC="MyJavaScriptProgram.js"></script>
```

*(References the JavaScript program in which the SetUp(), FillSecondOption(), and DisplayReport() functions are defined)*

```
</head>
<body topmargin="0" leftmargin="0"
onLoad="Setup()">
```

*(Defines the event handler to perform the function Setup() when the body object is completed downloading into the browser)*

```
<p align="center"><img src="title.jpg"
 alt="Retail Banking Reporting"></p>
<form name="MyForm">
<p align="center"><b>
<font face="Arial" color="#0000FF" size="5">
Market Level:</font></b><select name="FirstOption"
onChange="FillSecondOption()"</select></p>
```

*(This code defines the event handler to perform the function FillSecondOption() when the select object changes)*

```
<p align="left">&nbsp…&nbsp<b><font
face="Arial" color="#0000FF" size="5">
Banking Center Level:</font></b>
<select name="SecondOption"</select></p>

<p><input type="button" value="Display
Report" name="DisplayBtn"
onClick="DisplayReport()"</p>
```

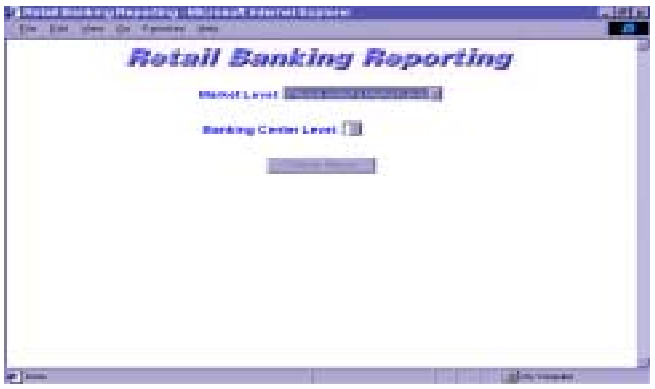
*(This code instructs the browser to perform the function DisplayReport() when a user has pressed and released the mouse button or keyboard equivalent on the button input object)*

```
</form>
</body>
</html>
```

In the HTML code above the ***<select>*** tags are used to create the drop-down list (or selection) objects named “FirstOption” and “SecondOption”. The display button named “DisplayBtn” is created with the ***<input*** type="button"**>** tag. These objects are referenced throughout the JavaScript program.

The first event handler referenced (***onLoad***="Setup()") basically tells the browser to execute the function Setup() when the web page is opened. The second event handler (***onChange***="FillSecondOption()") instructs the browser to execute the function FillSecondOption() when the user changes or selects an entry in the drop-down list "FirstOption". Finally, the third event handler (**onClick**="DisplayReport()") tells the browser to run the function DisplayReport() when the user clicks on the button named "DisplayBtn" .

The result of this HTML, as it appears in the browser, is shown below.



If you had implemented SAS/IntrNet and wanted to access dynamic HTML reports from this menu web page some additional code would be needed. You need to link the SAS program you want executed that will allow SAS/IntrNet to pass the values of the selections from the drop-down lists to the indicated SAS program as macro values. A sample of the enhanced HTML code from above with the necessary code for SAS/IntrNet is shown in bold below .

```
<p align="center"><img src="title.jpg"
 alt="Retail Banking Reporting"></p>
<form name="MyForm"

action="../cgi-bin/broker.exe" method="POST"
OnSubmit="returnValidate(document.TheForm)">
<input type=hidden name=_service value=default>
<input type=hidden name=_program
value="dynamic.RunReports.sas">
…
<p><input type="submit" value="Display
Report" name="DisplayBtn"</p>
```

By adding this code our select objects named "FirstOption" and "SecondOption" become macros that pass to the SAS program "RunReports.sas" and we no longer need the function DisplayReport(). SAS will use the macros in the referenced RunReport SAS program to generate the HTML of the specific banking center level requested.

## BUILDING THE JAVASCRIPT PROGRAM

The JavaScript program referenced in the ***<script>*** tags defines the three functions listed in the HTML: Setup(), FillSecondOption() and DisplayReport().

The Setup() function sets up the drop-down lists on opening the web page. It creates the options or choices to show in our first option drop-down list, such as "North", "South", or "East". It also clears out any previous selections and automatically shows the first selection available in the first option drop-down list upon entering the page. The example below shows some of this code.

*(Note: Due to space limitation some spacing was removed from the following programs to help readability)*

```
function setup()
{
 document.MyForm.FirstOption.disabled = true;
 document.MyForm.FirstOption.length = 0;
 document.MyForm.SecondOption.disabled= true;
 document.MyForm.SecondOption.length = 0;
```

*(This code defines both objects' **disabled** property as true to disable the selection objects while the function is running. It also defines their **length** property as 0 to clear out these selection lists)*

```
 var OptElem=document.createElement("OPTION");
OptElem.text = "North";
OptElem.value = "NORTH";
document.MyForm.FirstOption.options.add(OptElem);
```

*(This code creates option elements and adds them to the selection object "FirstOption". It also defines the text and value properties of these options. Similar code will generate several options in the "FirstOption" drop-down list.)*

The FillSecondOption() function creates elements or choices to show in the second option drop-down list. A sample of this code is shown below.

```
function FillSecondOption()
{
var FO = document.MyForm.FirstOption.value;
```

*(This code creates a variable FO equal to the FirstOption selection list value. In other words, the variable FO equals the market chosen in the first drop-down list.)*

```
if (FO == "NORTH")
 document.MyForm.SecondOption.disabled= true;
 document.MyForm.SecondOption.length = 0;

var OptElem=document.createElement("OPTION");
OptElem.text = "MAIN";
OptElem.value = "M00002.HTML";
document.MyForm.SecondOption.options.add(OptElem);
```

*(Basically this code creates option elements if the "FirstOption" selection option equals "NORTH" and adds them to the selection object "SecondOption". It also defines the text and value properties of these options. Similar code will generate several options in the "SecondOption" drop-down list.)*

This code shows that by defining the banking center elements, *if FO = "NORTH"* , we can control the options added to the "SecondOption" selection list. Additional code would use an ***else if*** statement. For example, after the statement ***else if*** *FO="SOUTH"* we would define the options added to "SecondOption" for this "FirstOption" value.

The DisplayReport() function actually instructs the browser to load a new page when you click the button object. An example of this code follows:

```
function DisplayReport()
 {
   var Report=document.MyForm.SecondOption.value;
  if (Report != "") location.href=Report;}
```

In the DisplayReport() function a new variable called "Report" is created which tells the browser the name of the report the user wants to display. For example, if the user selects the first option selection value of "NORTH" and the second option selection value of “00002” the function will concatenate these values and redefine the ***href*** property of the ***location*** object to a new HTML document called “AZ00002.html.” If the new location exists, the function will display it; otherwise, if the HTML file does not exists the browser will generate an error message.

All of the functions explained above contain other JavaScript code that this paper will not cover in detail. Copies of the entire JavaScript program will be available at our web site http://www.mgmdata.com/sas .

The majority of the JavaScript program was written and tested within the Internet Explorer web browser only.

## USING SAS TO BUILD THE JAVASCRIPT

Now that we have explained the JavaScript code needed, you probably realize how tedious it would be to build a static JavaScript program for a market hierarchy that has over 2000 banking centers!

SAS can do the work for you and build the selections available in your drop-down lists dynamically from the data. In order to achieve this you need three SAS programs. The first program will create distinct market and banking center data sets using ***PROC SQL.*** The next program will actually build the JavaScript program described above from these distinct data sets using Put statements. If you want to build the banking center level reports as static HTML reports your final program will produce all of them. If you have implemented SAS/IntrNet your final SAS program will utilize the macros passed from the web page and build the specific banking center level report requested when the user clicks the “Display Report” button.

The first program will consist of two simple ***PROC SQL***s that create distinct market and banking center data sets called DistinctMkt and DistinctBC respectively in the below examples. A copy of this program will be available on our web site http://www.mgmdata.com/sas.

The second program will consist of two data steps. The first data step builds the Setup() function, whereas, the second data step builds the FillSecondOption() and DisplayReport() functions. A sample of this SAS program is shown below.

```
data _NULL_;
 file “c:\MyJavaScriptProgram.js";
 set DistinctMkt end=lastrec;
 if _N_ = 1 then do;

 put 'function setup()';
 put '{';
 put 'document.MyForm.FirstOption.disabled=true;';
 put 'document.MyForm.FirstOption.length=0;';
 put 'document.MyForm.SecondOption.disabled=true;';
 put 'document.MyForm.SecondOption.length=0;';
end;
```

*(This code is only produced once when _n_ equals 1 because it is only needed in the JavaScript program once)*

```
 put 'var OptElem=document.createElement("OPTION");';
 put 'OptElem.text = "' MARKETNAME + (-1) '";';
 put 'OptElem.value = "' MKTCODE + (-1) '";';
 put ‘document.MyForm.FirstOption.options.add(OptElem);';
 put ' ';
```

*(This SAS code produced the JavaScript element options code for every record in the distinct market data set; thereby, populating the market or first option selection list with all of the markets in the data set.)*

The SAS code needed to build the FillSecondOption() function is slightly more complicated. A sample of this code is shown below:

```
data _NULL_;
 file “c:\MyJavaScriptProgram.js" mod;
 set DistinctMkt end=lastrec;
 …
 put '{';
 put 'else if (FO == "' MKTCODE + (-1) '")';
 put '  {';
 put 'document.MyForm.SecondOption.disabled=true;';
 put 'document.MyForm.SecondOption.length = 0;';
 put ' ';

 do I = 1 to noobs;
  set DistinctBC point=I nobs=nobs;
  if (MKTCODE EQ MKTCODE2) then do;
  put 'var OptElem=
              document.createElement("OPTION");';
  put 'OptElem.text = "' BANKNAME +(-1)'";';
  put 'OptElem.value = "' BANKCODE +(-1)'.HTML";';
  put 'document.MyForm.SecondOption.
                         options.add(OptElem);';
  put ' ';
 end;
```

*(This SAS code produced the JavaScript element options for every record in the distinct market/banking center data set; thereby, populating the second option selection list with only those banking centers within the market directly from the data.)*

Unfortunately, the SAS code above is difficult to read in this format and due to space limitations we were not able to show the entire program. A copy of the program will be available at our web site http://www.mgmdata.com/sas.

If you want the third SAS program to create static reports for every distinct banking center in your data set, you simply modify the above ***PROC PRINT*** by implementing macros to produce a report for each banking center. The SAS code below creates a macro value for each banking center's unique identifier and then creates a static HTML report for each banking center. A copy of this program will be available at our web site http://www.mgmdata.com/sas .

```
%macro bccount;
  proc sort data=sampledata;
    by bankingcenter;
  run;
  proc summary;
    by bankingcenter;
    output out=DistinctBC;
  run;
  data _Null_;
    set DistinctBC end=no_more;
    list_count=put(_n_,3.);
    call sympût (‘bccode’||left(list_count),
              compress(put(bankingcenter,20.)));
    if no_more then
       call symput(‘list_count’,list_count);
   run;
  %mend bccount;
  %bccount;
```

*(This SAS code creates a count and macro code of every distinct banking center to use in the following code to produce a report for every banking center in the data set.)*

```
%macro runrpt;
%local i;
```

```
   %do i=1 %to &list_count;
    ODS listing off;
    ODS html file = “c:\&&bccode&i..html”;
     proc print data=sampledata label noobs;
       where bankcode = “&&bccode&i";
       var day count;
       title ‘Title of Report’;
     run;
   ODS close html;
   %end;
   %mend runrpt;
   %runrpt;
```

If you want the third SAS program to create the banking center level reports dynamically, you would need to modify the SAS code shown above slightly. The changes needed are in bold below.

```
    ODS listing off;
   ODS html file =_webout;
    proc print data=SampleData label noobs;
      where bankcode=”&SecondOption”;
      var day count;
      title ‘Title of Report’;
    run;
   ODS html close;
```

This program is much simpler and reduces the amount of storage needed, since you will not need to store static reports. By combining the tools of SAS, SAS/IntrNet, HTML and JavaScript you can meet all of your customers reporting needs.

## CONCLUSION

Since 1998, we have greatly enhanced our reporting capabilities and increased our productivity by using SAS, JavaScript and HTML to provide static and dynamic reporting to our users. Our users are very happy with the visually attractive reports, our reporting capabilities, and the ease of use of our web site. Using SAS to generate JavaScript and HTML from our data has reduced reporting errors and decreased the time required to maintain our web site. By combining SAS/Base and SAS/IntrNet the possibilities seem endless. Our department has been known to say, “we can do anything” and with SAS it’s true.

## REFERENCES

LaFler, Kirk Paul, “Creating HTML Output with Output Delivery System”, *Proceedings of the Twenty-Fifth Annual SAS Users Group Inernational Conference*, 2000.

Sloan, Faith R., “Introduction to Dynamic Web Publishing”, *Proceedings of the Twenty-Fifth Annual SAS Users Group Inernational Conference*, 2000.

Haworth, Lauren, “HTML for the SAS Programmer”, *Proceedings of the Twenty-Fifth Annual SAS Users Group Inernational Conference*, 2000.

Goodman, Danny, *Dynamic HTML: The Definitive Reference*, O’Reilly & Associates, Inc., 1998.

SAS Institute, Inc., *SAS Web Tools: Using Web Publishing Tools with SAS Output Course Notes*, SAS Institute, Inc., 1998.

SAS Institute, Inc., *SAS Web Tools: Running SAS Applications on the Web Course Notes*, SAS Institute, Inc., 1998.

The Professional Development Group, Inc., *Introduction to JavaScript*, The Professional Development Group, Inc., 1998.

## ACKNOWLEDGMENTS

We would like to acknowledge EJ Harank for encouraging our participation at SAS conferences and always supporting our training. We also would like to thank our families for their understanding and support in our careers. Thanks to our proof readers Susanne Young, Stacy Ries and Tina Dunning. Thanks also to Vanessa Hayden for all of your advice and training, we miss working with you.

## CONTACT INFORMATION

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

Jennifer Sinodis
Bank One
201 N Central
Phoenix, AZ 85004
(602)221-4771
Email: Jennifer_R_Sinodis@bankone.com

Mark Moran
Bank One
201 N Central
Phoenix, AZ 85004
(602)221-4725
Email: Mark_E_Moran@bankone.com

Or visit us at
http://www.mgmdata.com/sas