Paper 187-26

How to Create Dynamic HTML and Javascript using your Data

Jennifer Sinodis, Bank One, Phoenix, AZ

ABSTRACT

With increasing information technology the Internet/Intranet offers an accessible channel for displaying and viewing meaningful information. By using SAS to generate HTML and JavaScript you can create great looking, maintenance-free, dynamic web pages for your users. This paper will show how to use SAS/Base to build HTML output to your web site using SAS Version 7 and Version 8, with HTML formatting macros and ODS (Output Delivery System). It will also show how SAS/Base can produce JavaScript to generate drop-down menu options built directly from your data, validate drill down options and dynamically access web output. The paper will also describe how SAS/IntrNet™ software can create this output dynamically from your web site.

For the purpose of this paper most of the SAS programs shown are simplified. Some of the programs use macro code, thus basic macro code knowledge is assumed. This paper will also show HTML, but it will only briefly explain some of its code for clarity.

INTRODUCTION

In the fall of 1998, my department (Direct Lending Information Solutions) produced and mailed hundreds of printed reports to retail managers all over the country. When we implemented our new retail reporting web site one of our first tasks was to replace these printed reports by publishing them to our web site.

We decided to convert our standard SAS reports to HTML files with SAS HTML formatting macros, using SAS Version 6.12. These static HTML files were updated nightly by a batch SAS program and stored on the web server. We continued to enhance our web site with SAS, building data-driven JavaScript drop-down menus. Finally, we used the new web publishing tools available in SAS Version 8 and SAS/IntrNet™ to convert this static web site to a completely dynamic web site!

Currently, we produce dynamic HTML files when a request is made from the retail web site, and store them in temporary directories on the web server.

USING THE %OUT2HTM MACRO TO PRODUCE HTML OUTPUT

There are several ways to produce HTML output using SAS/Base. HTML (Hyper Text Markup Language) is the common language for web browsers like Internet Explorer and Netscape. SAS offers four powerful HTML formatting macros that allow you to create HTML without having to learn the language: the output formatter (***%out2htm***), the dataset 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 dataset to HTML, the ***%tab2htm*** converts output from the SAS Tabulate procedure to HTML tables, and the ***%ds2graf*** converts graph output to HTML graphs.

We updated our existing retail SAS programs with the ***%out2htm*** formatting macro. The 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. This particular report summarizes the total number of Direct Lending applications by day for each market. The ***%out2htm*** code is marked in bold.

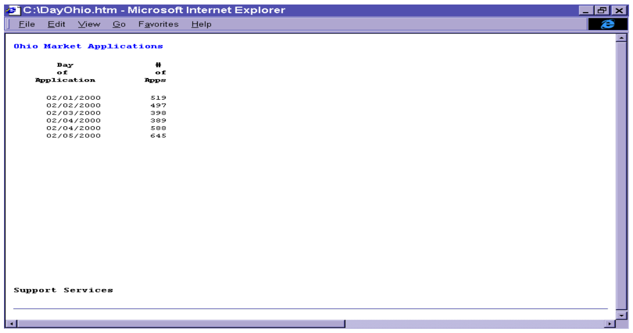
```
%macro runrpt(market);
 %out2htm(capture=on);
 proc report data=RetailData headskip nowd;
  where market="&market";
  column DAY NAPP;
  define DAY/group 'Day of Application';
  define NAPP/sum '# of Apps';
  title "&market Market Applications";
  footnote '<p style="page-break-after:
           always">Support Services</p>';
  run;
  %out2htm(capture=off,
    htmlfile="c:\Day&market..htm",
        openmode=REPLACE, runmode=B, encode=N,
        tcolor=BLUE, bgtype=COLOR,
        bg=WHITE);
%mend runrpt;
%runrpt(OHIO);%runrpt(TEXAS);%runrpt(ARIZONA);
```

Essentially, this code calls the ***%out2htm*** macro and activates the capture mode, telling SAS to capture everything until the capture mode is turned off. The program uses macro code in order to produce the same report for several different markets.

This example also shows several parameters available with the ***%out2htm*** macro. The ***htmlfile*** parameter stores the HTML output in the specified file and directory. The ***openmode*** parameter controls whether the generated HTML code should replace the contents of an existing file or append to it. The ***runmode*** parameter when defined as "B" indicates the HTML results will be sent to the output file using programming statements and that the macro is executed in batch mode. The ***encode*** parameter ensures special HTML characters are handled correctly by SAS. Several additional parameters control the properties of the captured output, such as ***ctext***, ***tcolor***, ***tface***, ***hcolor***, ***dcolor***, ***bgtype*** and ***bg***. We use ***bgtype***, ***bg*** and ***tcolor***. The ***bgtype*** parameter defines the type of background (color or image). The ***bg*** parameter controls the background color (when ***bgtype***=color) or the background image (when ***bgtype***=image). The ***tcolor*** parameter is used to set the color of the title lines. A sample of the HTML code generated is shown below.

```
<html>
<body BGCOLOR="WHITE">
<h3><fontCOLOR="BLUE">OHIO Market
Applications</font></h3>
…
<h3><p>style="page-break-after: always">
Support Services</p></h3>
</body>
</html>
```

This generated HTML won't mean anything to a SAS programmer unless he or she is familiar with HTML tags. Learning HTML can be useful and allows manipulation of your HTML output; however, the idea is to use SAS to produce this type of complex HTML code. The result of this HTML, as it appears in the browser, is shown below.



```
Ohio Market Applications

        Day               #
         of               of
    Application          Apps

      02/01/2000          519
      02/02/2000          497
      02/03/2000          398
      02/04/2000          389
      02/04/2000          588
      02/05/2000          645




Support Services
```

In order to page-break after each full page of output and have column headings roll to each page (as requested by our users), we built some extra HTML code. We used ***<p></p>*** (paragraph) tags in our ***footnote*** statement to instruct the web browser to break between paragraphs. The SAS ***encode*** parameter allowed the browser to interpret these HTML tags and create the page-break code (shown in bold in the HTML code above). The style option "page-break after: always" tells the web browser to page-break when the ***footnote*** is encountered. This style option works only in Internet Explorer.

USING ODS TO PRODUCE HTML OUTPUT

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.

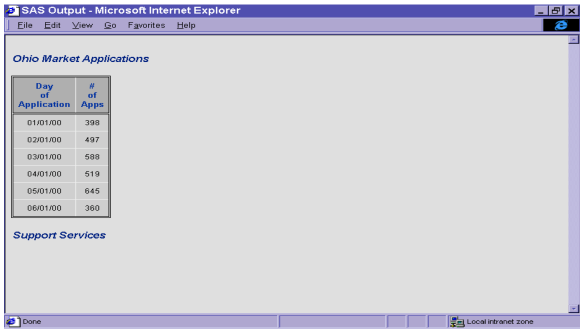
For the purpose of this paper we used ODS instead of ***%out2htm*** because ODS creates more visually attractive reports. ODS controls the formatting of all output objects and transforms basic monospace output to enhanced visual output by allowing us to manipulate the color, style and fonts of the reports, as well as use style sheets and templates.

We updated several retail reports to enhance their visual appearance using ODS. 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, and RTF (rich text file). The following program shows how to use ODS to convert the same proc report shown above to HTML. The ODS code is in bold.

```
%macro runrpt(market);
ODS LISTING OFF;
ODS HTML FILE= "c:\Day&market..htm";
proc report data=RetailData headskip nowd;
  … (same code as above )
run;
ODS HTML CLOSE;
%mend runrpt;
%runrpt(OHIO);
%runrpt(TEXAS);
%runrpt(ARIZONA);
…
```

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 very easily transforms standard SAS output to an HTML document, but the resulting HTML code and document in the browser looks 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.



Ohio Market Applications

Day of Application	# of Apps
01/01/00	398
02/01/00	497
03/01/00	588
04/01/00	519
05/01/00	645
06/01/00	360

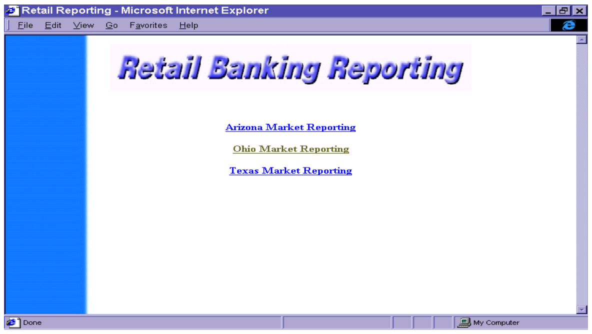
Support Services

HTML AND JAVASCRIPT

After we re-created our reports using ***%out2htm*** and ODS we still had to create a menu page for the retail web site so the users could access their market reports from the web. Using Microsoft FrontPage 98, we built a permanent HTML document with links for each market (Note: HTML files can also be built in a regular text editor application). A sample of the HTML code is shown below.

```
<html>
<head>
<title>Retail Reporting</title>
<body BACKGROUND="blueback.gif">

<p align="center"><img SRC="title.jpg"
alt="Retail Reporting" HEIGHT="83"
WIDTH="495"></p>

<p align="center"><font SIZE="+2"
FACE="Arial"><a href="DayArizona.htm"><b>
Arizona Market Reporting</b></a></font></p>

<p align="center"><font SIZE="+2"
FACE="Arial"><a href="DayOhio.htm"><b>Ohio
Market Reporting</b></a></font></p>

<p align="center"><font SIZE="+2"
FACE="Arial"><a href="DayTexas.htm"><b>Texas
Market Reporting</b></a></font></p>
</body>
</html>
```

Essentially, this HTML code created a web page with several links, in which each market name referenced a different location on the retail web site. The resulting output in the browser is shown below.



Retail Banking Reporting

Arizona Market Reporting

Ohio Market Reporting

Texas Market Reporting

As you can see, we used an image for our title and background.

This menu page worked well so long as only market level reporting was needed; however, our users soon requested banking center level reporting. We had to re-examine our retail web site and make several changes to accommodate this new level of reporting.

If we added links for all our banking centers to the menu page our users would have to scroll through a long list of links in order to make their selection. To make it easier for our users we chose to rebuild our menu page and use drop-down lists instead of links. We needed two drop-down lists to make banking center level selections quickly--one for market selection and one for banking center selection. The banking center drop-down lists would be limited to the banking centers available based on the market selection made in the market drop-down list. This type of selection limitation required additional programming not available in HTML; therefore, we needed to add programming capabilities to the code. Our option was to use either VBScript or JavaScript. We selected JavaScript due to its capabilities across browsers.

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.MLevel.length = 0;

Method syntax:
   Object.method()
   Example: document.MyForm.MLevel.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="MLevel"
   onChange="MktLevel()"</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 RptRetail.js we would include the following in our HTML code, generally at the beginning of the HTML file.

```
<script language="javascript"
        src="RptRetail.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

After learning some basic JavaScript we moved forward to rebuild our retail menu page using drop-down lists.

First, we built a new HTML file creating two drop-down lists--one for market selection and one for banking center selection. We analyzed the events that we wanted to take place based on the user's actions and decided that we needed functions 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

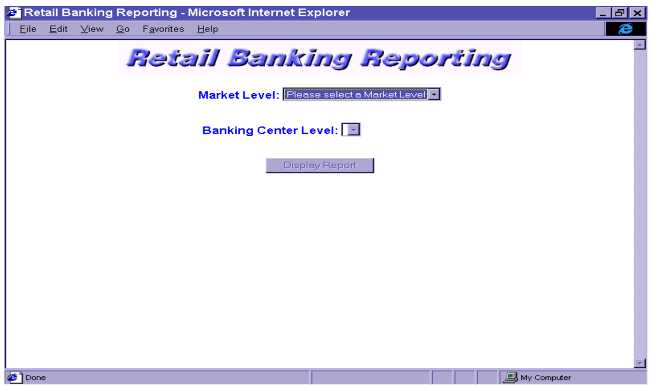
We named these functions Setup(), MktLevel() and DisplayReport().

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="retailrpt.js"></script>

(References the JavaScript program in which
the SetUp(), MktLevel(), 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 Reporting"></p>
<form name="MyForm">
<p align="center"><b>
<font face="Arial" color="#0000FF"size="5">
Market Level:</font></b><select name="MLevel"
```

```
onChange="MktLevel()"</select></p>

(This code defines the event handler to perform
the function MktLevel() 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="BCLevel"</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 “MLevel” and “BCLevel”. 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***=”MktLevel()”) instructs the browser to execute the function MktLevel() when the user changes or selects an entry in the drop-down list “Mlevel”. 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.


Retail Banking Reporting - Microsoft Internet Explorer
File Edit View Go Favorites Help
Retail Banking Reporting
Market Level: Please select a Market Level
Banking Center Level:
Display Report
Done
My Computer

BUILDING THE JAVASCRIPT PROGRAM

The JavaScript program referenced in the ***<script>*** tags defines the three functions listed in the HTML: Setup(), MktLevel() 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 market level drop-down list, such as “Arizona”, “Texas”, or “Ohio”. It also clears out any previous selections and automatically shows the first selection available in the market 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.MLevel.disabled = true;
 document.MyForm.MLevel.length = 0;
 document.MyForm.BCLevel.disabled = true;
 document.MyForm.BCLevel.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 OptElem1=document.createElement("OPTION");
OptElem1.text = "ARIZONA";
OptElem1.value = "AZ";
document.MyForm.MLevel.options.add(OptElem1);

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

The MktLevel() function creates elements or choices to show in the banking center level drop-down list. A sample of this code is shown below.

```
function MktLevel()
{var ML = document.MyForm.MLevel.value;

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

if (ML == "AZ")
{document.MyForm.BCLevel.disabled = true;
 document.MyForm.BCLevel.length = 0;

var OptElem1=document.createElement("OPTION");
OptElem1.text = "GLENDALE MAIN";
OptElem1.value = "00002.HTM";
document.MyForm.BCLevel.options.add(OptElem1;

(Basically this code creates option elements
if the “Mlevel” selection option equals “AZ”
and adds them to the selection object
“BClevel”. It also defines the text and value
properties of these options. Similar code will
generate several options in the “BCLevel”
drop-down list.)
```

This code shows that by defining the banking center elements, *if ML = “AZ”* , we can control the options added to the “BCLevel” selection list. Additional code would use an ***else if*** statement. For example, after the statement *else if ML=”OH”* we would define the options added to “BCLevel” for this “Mlevel” 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.MLevel.value +
           document.MyForm.BCLevel.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 market selection value of “AZ” and the banking center 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.htm.” 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 upon request.

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 have been to build a permanent JavaScript program for a market hierarchy that has over 2000 banking centers!

We decided to let SAS do the work for us. In order to achieve this we wrote three SAS programs. The first program created distinct market and banking center datasets using ***proc SQL.*** The next program actually built the JavaScript program described above from these distinct datasets using ***put*** statements. The final program produced all of the banking center level reports based on the Direct Lending data. The first program consisted of two simple ***proc SQL***s and a copy of this program is available upon request. The second program consisted of two data steps. The first data step built the Setup() function, whereas, the second data step built the MktLevel() and DisplayReport() functions. A sample of this SAS program is shown below.

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

 put 'function setup()';
 put '{';
 put 'document.MyForm.MLevel.disabled=true;';
 put 'document.MyForm.MLevel.length=0;';
 put 'document.MyForm.BCLevel.disabled=true;';
 put 'document.MyForm.BCLevel.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' _n_ + (-1)'=
        document.createElement("OPTION");';
put 'OptElem' _n_ + (-1) '.text = "'
               MARKET + (-1) '";';
put 'OptElem' _n_ + (-1) '.value = "'
                MKTABBR + (-1) '";';
put
'document.MyForm.MLevel.options.add(OptElem'
          _n_ + (-1) ');';
put ' ';
```

(This SAS code produced the JavaScript element options code for every record in the distinct market dataset; thereby, populating the market selection list with all of the markets in the dataset.)

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

```
data _NULL_;
file "c:\RptRetail.js" mod;
set MARKET end=lastrec;
…
put '{';
put 'else if (ML == "' MKTABBR + (-1) '")';
put '  {';
put 'document.MyForm.BCLevel.disabled=true;';
put 'document.MyForm.BCLevel.length = 0;';
put ' ';

do I = 1 to nobs;
 set BANKCNTR point=I nobs=nobs;
 if (MKTABBR EQ MKTABBR2) then do;
 put 'var OptElem' I + (-1) ' =
      document.createElement("OPTION");';
 put 'OptElem' I + (-1) '.text = "'
                    BCTRNAME + (-1) '";';
 put 'OptElem' I +(-1) '.value = "'
            BANKCNTR + (-1) '.HTM";';
 put
'document.MyForm.BCLevel.options.add(OptElem'
       I + (-1)');';
 put ' ';
 end;
end;
```

(This SAS code produced the JavaScript element options for every record in the distinct market/banking center dataset; thereby, populating the banking center 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 upon request.

For the third program we simply modified the above ***proc report***, by implementing macros to produce a report for every banking center within each market.

After we rolled out these three SAS programs in batch we had reports and JavaScript that updated daily. If a new branch was added or changed markets all three programs would automatically update from the Direct Lending data.

USING SAS/INTRNET^{TM} TO CREATE DYNAMIC HTML

When we learned of the SAS/IntrNet^{TM} application dispatcher 2.0, we were generating over a thousand banking center level retail reports in batch every night. This product allows us to have these reports generate dynamically from the web. Basically, we were able to discontinue running the program to produce the banking center level reports in batch, and only run it when the user clicks the “Display Report” button on the web site. We rewrote our SAS program that created the banking center level reports in batch and changed it to only produce the report needed based on macros passed to the program from the web site. A sample of this program is shown below:

```
%out2htm(capture=on);
or ODS HTML FILE = "c:\&Mlevel.&BCLevel";

 proc report data=RetailData headskip nowd;
  where market="&MLevel" and bcenter =
"&BCLevel";
  column DAY NAPP;
  define DAY/group 'Day of Application';
  define NAPP/sum '# of Apps';
  title "&BCLevel Applications";
  footnote '<p style="page-break-after:
            always">Support Services</p>';
  run;

  %out2htm(capture=off,
    htmlfile="c:\&Mlevel.&BCLevel",
       openmode=REPLACE, runmode=B, encode=N,
       tcolor=BLUE, bgtype=COLOR,
       bg=WHITE);
or ODS HTML CLOSE;
```

(Please remember these programs have been simplified for this paper.)

In order to implement the SAS/IntrNet^{TM} product we had to make certain changes per the requirements of the application. Some of the changes included modifying our configuration on our SAS server, as well as defining SAS/IntrNet^{TM} program and data libraries.

We also had to add specific code to our HTML file in order for the values of the selections from our drop-down lists to pass to the indicated SAS program as macros. A sample of the enhanced HTML code from above with the additional SAS/IntrNet^{TM} code required is shown below.

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

<form name="MyForm" (we removed the >)

action="http://cspm.bankone.net/cgi-
bin/broker.exe" method="POST"
OnSubmit="returnValidate(document.TheForm)">
<input tupe=hidden name=_service value=default>
<input type=hidden name=_program
value="dynamic.CreateRetailReports.sas">
 (the above code was added)
…
```

By adding this code our select objects named "MLevel" and "BCLevel" become macros that pass to the SAS program "CreateRetailReports.sas". SAS uses these macros in the ***proc report*** program from above to generate the HTML of the specific banking center level requested. All of the required changes to implement SAS/IntrNet^{TM} have been invisible to our users. Their reports are updated daily and easy to find using the drop-down lists on the web site.

CONCLUSION

Since 1998, we have greatly enhanced our retail reporting and increased report production. Our users are very happy with the visually attractive reports and our reporting capabilities. Using SAS to generate JavaScript and HTML has reduced errors, decreased maintenance and increased the "friendliness" of our web site. By combining Base/SAS and SAS/IntrNet^{TM} 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

I would like to acknowledge the following people for assisting with the editing process of this paper: Stacy Ries, Peggy Fredricks, Patricia Davis, Janice Koger, Sam Radobenko, Jarrod Zaremba, Jeanne Corello, Tina Dunning, Rowel Fulinara, Richard Reigelsberger and John Clark. Thank you for helping out. I would also like to thank Mark Moran for teaching me something new every day! Lastly, but definitely not least - I would like to thank EJ Harank for taking a chance on an administrative assistant with a new degree in Information Systems. Thanks to Bank One for their support and resources.

CONTACT INFORMATION

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

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