

LEVERAGING YOUR MAINFRAME INVESTMENT: WEB Enable your Mainframe Applications using New Features of Versions 7 and 8.

Kevin Bestelmeyer, David Crow, and Denise Poll,
SAS Institute Inc., Cary, NC

ABSTRACT

The Web is here. Mainframe applications that were designed as batch applications, or for display on 3270 terminals may seem a long way from being Web enabled, but in many cases, a small effort can pay huge dividends in both the appearance and the utility of these mission-critical applications. This paper looks at some ways to use new features of SAS® Versions 7 and 8 to achieve this goal.

INTRODUCTION

The PC arrived, and close behind was the Web. When Internet obsession becomes a frequent topic in Ann Landers, you know the world has changed! But what can you do if you have mainframe applications that were invented before the Internet? The answer, surprisingly, is quite a lot. Unlike pure client/server architecture, where it was difficult for mainframe systems to participate, OS/390 is tailor-made for Web based applications: OS/390's role in life is to handle large volumes of requests with 24x7 reliability. The applications are already there, all we have to do is hook them up.

The Nashville project (SAS Versions 7 and 8) aims to do just that. The software products webEIST™, webAF™, Enterprise Guide™, SAS/IntrNet™, and the Output Delivery System (ODS) are all components of the Nashville project that are aimed at providing web access to SAS applications.

This paper will focus on two of these pieces:

- ODS (Output Delivery System)
- SAS/IntrNet

ODS provides powerful facilities to tailor the output of standard SAS procs and render it into any of several formats. When coupled with ordinary batch jobs, it makes Web publishing using static HTML a snap. Version 8 can also be used to produce GIF, JPEG, Postscript and PCL output. Example 1, discussed below, shows the use of HTML and GIF, including HTML links from a GIF graph into PROC REPORT output.

However, you don't always know in advance what reports a user or customer needs to see, and a **static** Web page just won't do the job. Sometimes you need to create Web pages on the fly to satisfy a request. To distinguish this from static Web pages, we call this **dynamic** Web processing.

In *dynamic* processing, a person at a Web browser selects processing options from various choices that are presented on a Web page. This is sent to a host server where a program produces a customized report, which is then returned to the browser for display. Example 2, discussed below, shows an example of how the SAS/IntrNet software product can couple a fairly ordinary looking SAS program with a Web browser to produce some dramatic results.

EXAMPLE 1: STATIC WEB PAGES PRODUCED BY ODS.

Take a little time to skim through Example 1 in Appendix A, and review the notes. Then once you've read this section, you should be ready to go through it in great detail. The example is fairly long, but that allows us to demonstrate a non-trivial example of a Web page.

In this example, the output is directed to 3 separate OS/390 HFS files (The OS/390 UNIX file system). We could have used a PDSE in the classic MVS file system, but we're trying to learn new stuff, HFS is a standard part of OS/390, and it really is easy to work with. The 3 files are:

- report_body_shoesg1b.html – a nice looking graph with HTML links into shoesg1b_regsales.html
- shoesg1b.gif -- the graph without the links
- shoesg1b_regsales.html – A PROC REPORT output with HTML tags inserted

We create these files by adding a few strategically placed statements in an otherwise ordinary looking SAS program. These are highlighted in bold, with a reference to a brief explanation of what each statement does follows the example. But for discussion, look at the statement:

```
ods html path=odsout
      body= 'report_body_shoesg1b.html'
      nogtitle;
```

This tells SAS to produce the output in **html** format. The **path** option references a previous filename statement and tells ODS which HFS directory to use. The **body** option tells what file name to use when the PROC runs and produces output. Finally, **nogtitle** suppresses titles so the output is a little cleaner. This is all you need to produce SAS output in HTML! Everything else is much the same as you would find in any SAS program. Of course, not every program uses the same PROCs this one uses, but ODS can handle the output from other PROCs in the same way.

If you have a Web server that can access HFS files, this is all you need to do to serve up some nice looking pages. If your Web server is running on another platform and cannot access the HFS files, you can move these files to the other platform in any number of ways. The simplest is probably to add an FTP step to the end of your job, or you could use the SAS FTP access method to write it to the other platform directly from SAS. But that's an example for another paper. If you find you need to do this, keep things simple until you figure out the EBCDIC->ASCII translation, and watch out for end-of-line characters.

As mentioned above, you could create these output files in the classic MVS file system with a PDSE. But please don't try an old-style PDS. A PDS can only have one output member open at a time, and ODS needs to have them all open at once. Unfortunately, there is no good way to detect this error, and you will simply create 3 garbage members if you do it by mistake.

One final note on ODS: With Version 6, SAS Institute delivered a set of free Web publishing tools. The ODS developers did their best to make it do everything you can do with those tools, plus a lot more. We think you'll be better off using ODS, but if you put a lot of effort into using the Web publishing tools, don't worry: They will continue to work, and you already have a head start on using the Web with your mainframe applications.

EXAMPLE 2: DYNAMIC WEB PROCESSING WITH SAS/INTRNET.

The second example looks more complex at first blush, but the individual pieces are really simpler. And, like most programming, you can usually find an example that's close to what you need and hack away at it. That's certainly how we created this one.

In this example, a person can go to a Web page that provides choices for how a report should be customized. After the options are selected, the Web page sends a URL back to the host. This URL has all the information that the host needs to run an application program and produce output to be displayed at the browser. This is the same kind of URL you see when you use a Web search engine.

When the Web server receives this URL from the browser, it calls the SAS/IntrNet Application Broker CGI, which in turn passes the request to a SAS/SHARE® server. The server processes the request, creates the customized HTML output, and

sends it back to the browser via the Web server.

This has a lot of moving parts, but each one is fairly simple. Appendix B1 shows the raw HTML that creates the Web page that you would initially see from a browser. Appendix B2 is what it looks like in the browser. To use it, you simply click on the radio buttons to specify how you want the report customized, and then click on DISPLAY to send the URL to your host's Web server. The resulting URL will look something like this:

```
//sasmvs.mvs.sas.com/BRKR-bin/broker?  
&sumvar=returns  
&salesnum=500000  
&_service=default  
&_debug=2  
&_PROGRAM=pgmlib.db2demo.sas
```

To break this down, the browser is communicating with a Web server at sasmvs.sas.com that has a CGI broker named BRKR-bin. (Hidden from view here is the fact that the broker is started with some configuration data that describes the TCP/IP socket it is to use to communicate with the SAS/SHARE application server). When the broker receives this URL, it passes it along to the application server. (Again, hidden from view here is the fact that the application server has some configuration data that tells it the name of the MVS PDS that is associated with pgmlib). The application server reads the program from the library, and runs the program using the values in sumvar, salesnum, in its processing. The program produces its output in HTML format, which is then returned to the browser by the Web Server.

The db2demo program is shown in appendix B3, and you will see that it is simpler than the program in the first example. That's not surprising because the first program was a general report aimed at a large number of users, and this one is tailored for a specific request. This program also shows the simplicity of accessing DB2 data using SAS. And Version 7 made this even easier because it supports the full range of DB2 names, and accessing DB2 databases no longer requires the use of a view.

If the request you are making will require a ton of processing, and take a long time to complete, you may not want to wait around while all this happens. You could use the same kind of processing to submit a batch job. Checking to see when a batch job has finished is always troublesome, and we all have better things to do. So Version 8 has an experimental email capability. As you might expect, you can send ordinary text to announce completion, or error conditions, but you can also include attachments, such as a graph in GIF format. The experimental label on email is our way of warning you that we are still thinking about exactly how it should work, so when the production release of the email interface comes out, you may have to make changes to get it to work right. So please experiment with it, but think twice before including this in a production job.

CONCLUSION

The topics covered in this paper are just the beginning of what you can do with the new features in SAS. These examples have barely scratched the surface of what you can do with ODS and SAS/IntrNet, and we didn't even mention the other products. We believe we have provided a set of tools that will not only allow you to take advantage of your investment in existing applications, but also to extend these applications and to develop new ones. As developers, we are really looking forward to seeing our customers use these tools for some heavy duty work.

APPENDIX A: EXAMPLE 1.

This is an example of a SAS program that produces 3 output reports. These reports contain links among them, so that when you are looking at the summary graph, you can link to detailed information. The example is intended to be complex enough to show how a real life application report could be put out on the Web. **BOLD** text identifies a statement that is key to producing the Web output. Each of these references a note at the end of the example that describes a little more about the statement's purpose..

```
/* define destination for graphics and HTML output */
filename odsout '/u/mktDEM3/public_html'; /*note 1*/

/* stops sending output to GRAPH and OUTPUT windows */
ods listing close; /* note 2 */

/* set the graphics environment */
goptions reset=global gunit=pct
          colors=(black blue green red)
          hsize=8.42 in vsize=6.31 in ftitle=zapfb
          ftext=swiss htitle=4 htext=2.5
          device=gif transparency noborder; /*note 3*/

/* add the HTML variable to NEWSHOES */
/* this lets us have links between the web pages */
data newshoes;
  set sashelp.shoes;
  where Region in ('Canada' 'Central America/Caribbean'
                  'South America' 'United States');
  length regdrill $40;
  if Region='Canada' then /* note 4: */
    regdrill='HREF="shoesg1b_regsales.html#IDX1"';
  else if Region='Central America/Caribbean' then
    regdrill='HREF="shoesg1b_regsales.html#IDX2"';
  else if Region='South America' then
    regdrill='HREF="shoesg1b_regsales.html#IDX3"';
  else if Region='United States' then
    regdrill='HREF="shoesg1b_regsales.html#IDX4"';
run;

/* assign the destination for the ODS graphics output */
/* and ODS HTML files */
ods html path=odsout /*note 5*/
      body='report_body_shoesg1b.html'
      nogtitle;

/* define title and footnote for chart */
title1 'Total Sales for the Americas';
footnote1 h=3 j=1 'click on bars' j=r 'REPORT3D ';

/* assign a pattern color for all bars */
/* this is just ordinary graph stuff here */
```

```

pattern color=cyan;

/* define axis characteristics */
axis1 order=(0 to 7000000 by 1000000)
      minor=(number=1)
      label=none;
axis2 label=none offset=(4,4)
      value=('Canada' 'C. Amr./Car.'
            'S. America' 'USA');

/* generate vertical bar chart */
proc gchart data=newshoes;
  vbar3d Region / discrete
        width=6
        sumvar=sales
        html=regdrill /* note 6 */
        coutline=black
        cframe=blue
        maxis=axis2
        raxis=axis1
        name='shoesg1b '; /* note 7 */
run;
quit;

/* open the HTML destination for the PROC REPORT output */
ods html body='shoesg1b_regsales.html' path=odsout; /* note 8 */

/* sort data set NEWSHOES in order by region */
proc sort data=newshoes;
  by Region Subsidiary Product;
run;
quit;

/* clear the footnote */
goptions reset=footnote;

/* suppress the default BY-line */
option nobyline;

/* print a report of shoe sales for each Region */
title 'Sales Report for #byval(Region)';
proc report data=newshoes nowindows;
  by Region;
  column Region Subsidiary Product Sales;
  define Region / noprint group;
  define Subsidiary / display group;
  define Product / display group;
  define Sales / display sum format=dollar12.;
  compute after Region;

```

```

        Subsidiary='Total';
    endcomp;
    break after Region / summarize style=rowheader page;
run;

/* Close the HTML destination and open the listing output */
ods html close; /* note 9 */
ods listing;    /* note 10 */

```

NOTES ON EXAMPLE 1:

- 1: Set filename **odsout** to the HFS directory where we will write the data sets.
- 2: Close the listing output so only desired output goes to html.
- 3: Tell graph you will be producing graphs in GIF format.
- 4: HREF is a link from one html page to another. The link has the form "name#label", where:
Name – is the name of the file you are linking to.
Label – is a label where the browser will position
5. Assign the graphics output to the file specified by **body**, using the directory specified by **path**.
6. html=regdrill means that the variable regdrill is to be output as an html link.
7. The name of the output file.
8. Tells ODS to handle PROC REPORT output, and where to put it.
9. Close the HTML destination.
10. Open the listing output.

APPENDIX B1

This is the HTML for the Web page that kicks off the request to SAS/IntrNet. It almost fits nicely on a normal Web browser screen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
  <TITLE>Display Shoe Sales by Product</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000" LINK="#0000FF" VLINK="#800080"
ALINK="#FF0000">

<H1>Display Shoe Sales by Product</H1>
<HR>
Use this Dispatcher application to display a specified
data set in an HTML-formatted table. You can
select to display sales information for specified
products.
<p>Make your selections and click on the <b>Display Table</b> button.</p>
<hr>
<b>Display Options</b>
<p><i>Select Sum column and Sales range.</i> </p>
<p>
<FORM ACTION="/BRKR-bin/broker"> <!-- Change this line for your site -->
<TABLE BGCOLOR="#COCOCO" BORDER CELLPADDING="2">
  <TR BGCOLOR="339999">
    <TD>
<B>Select display option:</B>
      </TD>
      </TR>
      <TR>
        <TD>
          <INPUT TYPE=radio NAME="sumvar" VALUE="sales" CHECKED>Total by Sales
        </TD>
      </TR>
      <TR>
        <TD>
          <INPUT TYPE=radio NAME="sumvar" VALUE="returns">Total by Returns
        </TD>
      </TR>
    </TABLE>
  </p>
  <hr>
  <TABLE BGCOLOR="#COCOCO" BORDER CELLPADDING="2">
    <TR BGCOLOR="339999">
```

Select Sales range:
<input checked="checked" name="salesnum" type="radio" value="0"/> Sales > 0
<input name="salesnum" type="radio" value="50000"/> Sales > 50000
<input name="salesnum" type="radio" value="100000"/> Sales > 100000
<input name="salesnum" type="radio" value="500000"/> Sales > 500000
<input name="salesnum" type="radio" value="1000000"/> Sales > 1000000

Service and Debug Options

Select the service as defined by your configuration file. This service identifies how the Application Broker communicates with the Application Server.

Select service:

☐ Default Service (Socket)

☐ Launch Service

☐ Pool Service

Debugging level:

☐ None

☒ Show time

```
<OPTION VALUE="18"> Get a hex dump of the output
<OPTION VALUE="258"> Show a connection trace
<OPTION VALUE="14"> Just list all services
<OPTION VALUE="131"> Show fields, time, SAS Log
</SELECT>
</NOBR>
<HR>
<INPUT TYPE="HIDDEN" NAME="_PROGRAM" VALUE="pgmlib.db2demo.sas">
<INPUT TYPE="SUBMIT" VALUE="Display Table">
</FORM>
</BODY>
</HTML>
```

APPENDIX B2

This is what the page looks like. You click on the radio buttons, and then click **DISPLAY TABLE**:

Use this Dispatcher application to display a specified data set in an HTML-formatted table. You can select to display sales information for specified products.

Make your selections and click on the **Display Table** button.

Display Options

Select Sum column and Sales range.

Select display option:

☐	Total by Sales
☐	Total by Returns

Select Sales range:

☐	Sales > 0
☐	Sales > 50000
☐	Sales > 100000
☐	Sales > 500000
☐	Sales > 1000000

Service and Debug Options

Select the service as defined by your configuration file. This service identifies how the Application Broker communicates with the Application Server.

Select service: Debugging level:

APPENDIX B3

This is the program that is run after you click on the DISPLAY Button at the bottom of the page in Appendix B2:

```
%global sumvar salesnum;

libname db2demo db2 authid=mktDEM3 in='database public1';
ods listing close;
ods html body=_webout (dynamic)
      rs=none;

data newshoes;
  set db2demo.shoes2;
  where Region in ('Canada' 'Central America/Caribbean'
                  'South America' 'United States');
run;

proc sort data=newshoes;
  by Product;
run;

title 'Shoe Sales';
proc print data=newshoes;
  by Product;
  where (Sales > &salesnum);
  sum &sumvar;
  format Sales comma10.;
run;

ods html close;
```

REFERENCES

For more information on these topics, and other stuff from SAS Institute, check out our Web site at [//www.sas.com](http://www.sas.com) and especially check out the link on Web enablement at [//www.sas.com/web](http://www.sas.com/web).

ACKNOWLEDGMENTS

We gratefully acknowledge the help of John Leveille, Ellen Kunkel, Nancy Wills, and the entire OS/390 Host Development and testing staff. Without their help this paper would not have been possible.

AUTHORS

Dave Crow
SAS Institute Inc.
SAS Campus Drive
Cary NC, 27513
(919) 677-8000
dave.crow@sas.com

Kevin Bestelmeyer
SAS Institute Inc.
SAS Campus Drive
Cary NC, 27513
(919) 677-8000
kebest@wnt.sas.com

Denise Poll
SAS Institute Inc.
SAS Campus Drive
Cary NC, 27513
(919) 677-8000
Denise.Poll@sas.com

SAS, SAS/SHARE, SAS/IntrNet, webAF, webEIS, and Enterprise Guide are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA Registration.

Other brand and product names are registered trademarks of their respective companies.