Paper 180

To Be or Not to Be? That is the Dynamic Web Page Question.

Thomas Kunselman, P.A.I.R. Solutions, Inc.
Almira A. Lyst, Roche Diagnostics
Donald L. Penix, Jr., Pinnacle Solutions, Inc.

ABSTRACT

An important goal for Intranet applications development is quick response time. In the design stage of development, the issue of response time made us question whether to send dynamically generated Web pages directly back to the Web browser or store the web pages as temporary or permanent files. As we learned more about version 7 SAS® System components, including SAS/IntrNet® application dispatcher version 2.0, SAS/GRAPH®, SAS/QC® and SAS/STAT® and the Output Delivery System (ODS), we concluded that the Web pages we were dynamically generating would have to travel several paths to reach the browser. This paper describes four examples of how our Web application's dynamically generated HTML pages arrive at the browser and discusses the factors involved in our choice of methods.

INTRODUCTION

This paper explains several examples of dynamically generated Web pages using the SAS/IntrNet Application Dispatcher executing version 7 base SAS software, SAS/GRAPH, SAS/QC, SAS/STAT, and SAS Screen Component Language (SCL) code in a manufacturing application called **D**ata **I**ntelligence **V**erifies **A**dvantage (**DIVA**). The DIVA application provides process improvement and process monitoring information to managers, engineers, and analysts via an Intranet. Originally, the development team hoped to have all HTML reports sent directly to the client, without any need for intermediate storage of files on the Web server. However, as we became more familiar with the specifics of dynamic client/server applications, we realized that several different pathways for getting the dynamically generated HTML pages from the SAS Application Dispatcher to the Web browser were needed. In this paper, we give examples of different ways to generate Web pages:

1. dynamically generated pages that are not stored on Web server,
2. permanent pages that are created by a batch process and stored on the Web server with embedded links for drilldowns that dynamically generate Web pages that are not stored on the Web server,
3. dynamically generated pages that are not stored on the Web server, but have embedded links pointing to dynamically generated pages stored in temporary files on the Web server,
4. pages that are batch generated and stored on the Web server that make use of the JAVA applets provided by SAS Institute Inc.

DIVA APPLICATION OVERVIEW

Hardware and Software

The DIVA production system runs off of an AIX® server where the Apache Web server, SAS Application Broker and Server, and Oracle® database reside. A development environment was also created. In the development environment, the SAS Application Server resides on a PC running Windows® NT while the Application Broker is installed on an AIX server different from the production server. By placing the development environment on a PC instead of an AIX server, licensing costs were significantly reduced.

DIVA Application

The DIVA system is a manufacturing application that presents information to provide:

1. Production management with the necessary information to monitor and maintain the overall health of the manufacturing process and identify areas where improvement is necessary to meet strategic initiatives.
2. Production operators with the ability to monitor and understand the performance of their process relative to established expectations.
3. Troubleshooters (operators, maintenance, engineers) with the necessary information to drive improvement efforts and monitor the effects of improvement initiatives.

DIVA WEB PAGE GENERATION

The DIVA application has Web pages generated in three ways:

1. ***Static.*** Web pages are created using an HTML editor such as Dreamweaver®. These Web pages are stored on the Web server.
2. ***Pemanent.*** Web pages are created by a batch SAS program when the DIVA data is updated. The Web pages are stored on the Web server.
3. ***Dynamic.*** Web pages are generated whenever a request is made. The Web pages are returned to the Application Broker or stored in temporary directories on the Web server with the location being sent to the Application Broker. This is not the same as Dynamic HTML.

Batch Generated Permanent Web Page.

DIVA data is stored in an Oracle database with the tables updated in near real-time every 15 minutes. After the Oracle database has been updated, a set of SAS programs are executed in batch mode. These SAS programs pull the new information from the Oracle database tables and after processing the data, it is saved in SAS data sets. After the SAS data sets are created, the batch SAS programs generate and store a set of Web pages on the Web server.

These pages are made static because a) they are frequent requests that can place a bottleneck on the server, b) they involve summary data whose detail data may eventually be archived off the system, c) they take a long time to process, or d) they are "canned" reports.

Example 2 describes a batch generated permanent Web page.

Dynamically Generated Web Page

Two types of *dynamically generated* web pages are described in this paper. One type of dynamically generated Web page is sent directly to the Application Broker and is never stored on disk. The other type of dynamically generated Web page is stored in a temporary directory on the Web server. Requests for these Web pages are made by sending the appropriate Universal Resource Locator (URL) to the Web server. The request is usually made via a link on a Web page.

Example 1 describes a dynamically generated Web page that is never stored on disk. Example 3 describes a dynamically generated Web page that is stored in a temporary directory on the Web server.

EXAMPLE 1: HTML SENT TO APPLICATION BROKER

The selection window shown in Figure 1 is an example of a dynamically generated Web page. The selection window is an HTML form generated by an SCL program.
The select boxes are populated by unique variable values from SAS data sets. The selection window has has some JAVASCRIPT code which will make a request for other dynamically generated Web pages.

The HREF to request the selection window be generated is:
http://diva/cgi-bin/broker
?_program=**diva.FIG11.selectbox.scl**&_service=default
&machine=MACH&dateorlot=DATE

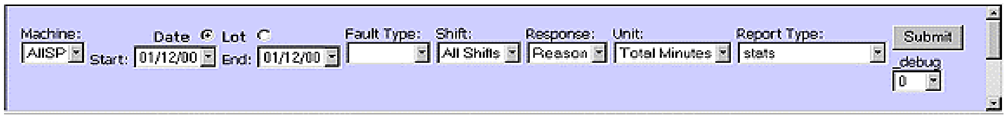

Machine: AllSP
Date ● Lot ○
Start: 01/12/00 End: 01/12/00
Fault Type:
Shift: All Shifts
Response: Reason
Unit: Total Minutes
Report Type: stats
Submit
_debug 0

Figure 1.1 - Dynamic Selection Window

The Machine select box and the Date/Lot radio button have an OnChange option specified. If the value of the Machine field (**&machine**) or the Date/Lot radio button (**&dateorlot**) changes, a JAVASCRIPT program is executed which sends a request to the Web server to

regenerate the selection window. The Application Broker then asks the Application Server to execute the SCL program to generate the selection window and repopulate the rest of the select boxes with values appropriate to the new machine selected.

Box 1 contains a fragment of SCL code used to generate the HTML for the selection window. In the first statement, the fopen function uses the SAS defined **_WEBOUT** reference to open output to the Application Broker. The fput function is used to tell the Broker and Web server what type of output is being received.

SCL Code for Selection Window - Box 1

```
fid=fopen('_WEBOUT','w');
rc=fput(fid,"Content-type: text/html");
rc=fput(fid,"");
***JAVA SCRIPT Code***
rc=fput(fid,='<script language="JavaScript"><!--');
rc=fput(fid,='    function modifySelect(formit)');
rc=fput(fid,='
{formit._program.value="diva.FIG11.selectbox.scl";');
rc=fput(fid,='formit.target="Selectframe";');
rc=fput(fid,='formit.submit(); } //--></script>');

***LIST BOX Generation Code***
rc=fput(fid,='<td align="top"><font size="2">Machine: ');
call display('diva.FIG11.newselbox.scl', fid, 'machine',
               'onChange="modifySelect(this.form)"',
               "divalib.machinedata", "machine", "$5.", "",
                quote(getnitemc(list,"machine")));
...more SCL statements...
rc=fput(fid,='<td align="top"><font size="2">Fault Type: ');
call display('diva.FIG11.newselbox.scl', fid, 'faulttype',
"divalib.machinedata", "faulttype", "$10.", "machine",
getnitemc(list,"machine"));
...more SCL statements...
```

The ***JAVASCRIPT Code*** section creates a **modifySelect** function that, when activated, will have the SAS application server run the SCL program defined by the variable **formit._program.value=**. In this case, the function is activated whenever the end-user changes the value of the machine or the date/lot radio button. The ***LIST BOX Generation Code*** section, calls a section of SCL code that creates the HTML select box. These parameters passed from the HTML request, including machine name, which is used to subset values for the select boxes. Also, any options for the select boxes are passed at this time, such as **onChange="modifySelect(this.form)"**.
The selection window HTML has to be dynamically generated because it requires unique values to populate

the select boxes. To reduce the processing time of the selection window Web page, PROC SUMMARY with NWAY is used to create a SAS data set with the unique values of each selection parameter.

EXAMPLE 2: PEMANENT HTML W/ IMAGE MAP DRILLDOWN

Figure 2.1 is an example of a Web page that has been generated by the batch SAS process and stored on the Web server. The page includes an image map which defines each bar on the chart as a link to dynamically generated detail reports. Each bar on the chart will tell the Web server to call the broker. The broker then executes an SCL program which to dynamically generate the detail data.

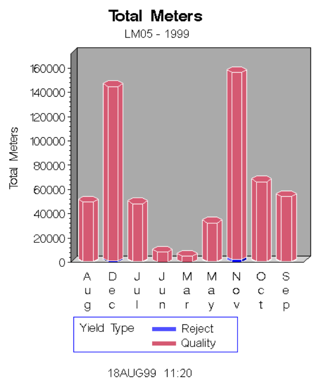


Figure 2.1 - Pemanent Summary

The SAS code used to generate the permanent web page, shown in Box 2.1, first creates the drilldown URL (***rpthtml***) in a macro, mkhtml. Next, proc gchart specifies the drilldown URL variable, ***rpthtml***, which is used to create the imagemap href's.

The PROC GCHART code in Box 2.1 generates the following HTML image map when the HTML= option is used.

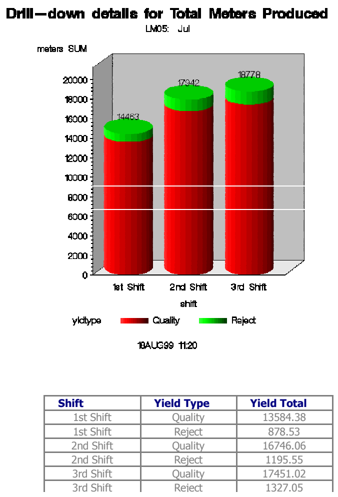
```
<MAP  NAME="gr9kc13g_map">
<AREA SHAPE="POLY" href="/cgi-bin/broker?
_PROGRAM=diva.FIG21.summary.scl&_SERVICE=d
efault&_MONTH=Jul"
COORDS="327,156,327,107,334,103,349,103,355,107,
355,156,349,160,334,160 " >
```

SAS Code for Batch Web Page – Box 2.1

```
%macro mkhtml;
  data work;  set work;
    %do i = 1 %to &nmon;
      if month="&&month&i" then rpthtml =
                  'href="/cgi-
           bin/broker?_PROGRAM=diva.FIG21.summary.s
           cl&_SERVICE=default
                              &_MONTH='||"&&month&i"||'"';
     %end;
%mend;
%mkhtml

proc gchart data=workl(where=(year='1999'));
vbar3d month / html =  rpthtml;
```

When a bar is clicked on the chart in Figure 2.1, the web browser sends a URL to the Web server to execute the broker. The broker passes the request to the SAS application server run the program **diva.FIG21.summary.scl**. This program generates the dynamic web page displayed in Figure 2.2, which uses an embedded image tag that makes another broker call.

Figure 2.2 - Dynamic Summary HTML Drilldown



Shift	Yield Type	Yield Total
1st Shift	Quality	13584.38
1st Shift	Reject	878.53
2nd Shift	Quality	16746.06
2nd Shift	Reject	1195.55
3rd Shift	Quality	17451.02
3rd Shift	Reject	1327.05

In Box 2.2, notice the image tag ***<IMG*** generated for Figure 2.2. When the Web browser receives the above HTML, the browsers then passes the location of the image back to the Web server. In this case, the Web server runs the program ***diva.FIG21.summary.scl*** passing along the parameters for month and machine**.** Next, the SAS application server runs the program ***diva.FIG22.summary.scl*** and sends the output back to the broker and the Web server sends it to the Web browser where the browser displays the image in the appropriate place.

Box 2.3 PART A shows the SAS code for generating this dynamic HTML drilldown using a data _null_ step for writing out the <IMG tag, which tells the server which program to execute to generate the image.

The SAS code in Box 2.3 PART B that generates the dynamic GIF image is pretty straightforward as well. In this case, however, ***Content-type: image/gif*** must be sent to _webout prior to the image being transmitted.

The top-level summary web page was developed as a “canned” report that would only be updated on a monthly basis. Hence, we chose to create it as a permanent file on the web server. However, with the unknown drilldown selection of the end users, it is practical to have the detail web pages generated dynamically. By doing so, we also relieve a lot of space requirements on our web server because we don’t need the large number of permanent detail files necessary to store multiple years of data.

SAS Code for Dynamic Summary HTML drilldown – Box 2.3 Part A

```
data _null_;
  file _webout;
  put '<IMG SRC=';
  put
'"http://diva/cgibin/broker?_PROGRAM=diva.FIG22.summary.scl'
;
  put '&_SERVICE=default&_MONTH=';
  put "&_month" '&_MACH=' "&mach" '">';
run;

ods html body=_webout;
  proc print data=work;
  run;
ods html close;
```

SAS Code for Dynamic Image – Box 2.3 Part B

```
data _null_;
  file _webout;
  put 'Content-type: image/gif';
  put ;
run;
goptions device=gif transparency noborder gsfname=_webout;
proc gchart data=work;
  where yldmonth="&_month";
```

EXAMPLE 3: DYNAMICALLY GENERATED WEB PAGE STORED TO TEMPORARY FILES

The thumbnail pie chart shown in Figure 3.1 is generated dynamically using the html=*variable* option in the SAS/QC pareto procedure where the value of *variable* references the SAS application server to run a specified SAS program

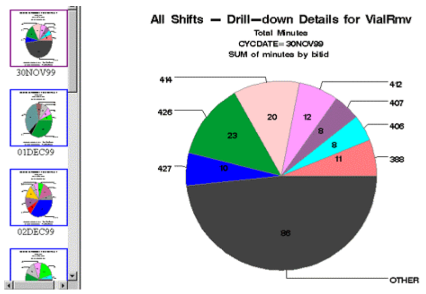
SAS Code for Temporary HTML drilldown – Box 3.1

```
data _null_;
  now=datetime();
  call symput('tmpext',compress(now));
run;

x 'cd /opt/sas/tmp';
x "mkdir &tmpext";

filename webdoc "/opt/sas/tmp/&tmpext";
goptions device=webframe gsfname=webdoc;
options nobyline;
proc gchart data=work;
  by date;
  pie var1;
run;
…more SAS statements…
%do i=1 %to &num;
  filename thumb "gchart&i.html" mod;
    ods html body=thumb (notop) gpath=webdoc;
      proc summary data=work;
        where date = &date;
        var var1 var2;
      run;
    ods html close;
%end;
```

when the end-user selects a bar on the pareto chart. The SAS code used to produce this web page is very similar to that shown in Box 2.1, which uses the same programming logic except with the gplot procedure.

Links defined by the image map of the pareto chart execute the broker. The application server runs an SCL program which sends info back to Web server and also writes out temporary HTML files accessed via anchor tags **<A**.

During the dynamic creation of this web page, the SAS program also creates GIF HTML web pages that correspond to each thumbnail GIF image. These files are stored in a temporary directory on the server. In DIVA, all of the temporary directories that we create on the server are located under one main directory (**/opt/sas/tmp**)



which is cleared out at the end of each day.

BitID	Frequency	Total Minutes
131	559	196.59
008	506	79.50
112	126	83.12
362	95	16.45
559	27	54.12
076	16	2.90

Figure 3.1 – Dynamic Pie Chart

In order to create a temporary file that is unique for each on-line DIVA user, we generate a temporary directory based on the value of ***now*** in the SAS datetime() function (see Box 3.1). Because the probability of different users requesting the same report and drilldown at the same time is very small, this was to be our best option. The value ***now*** is held in the macro variable **&tmpext**.

The **x "mkdir &tmpext"** statement allows us to create and reference these temporary directories on the web server as the user requests. The HTML files that are generated in the procedures that follow will be created under this directory as referenced in webdoc.

In our example, we use the SAS frame web page developed from the goption **device=webframe** and the gchart procedure to create our frame HTML file GIF image and GIF HTML files in the temporary directory. Next, we modify the GIF HTML files (**gchart&i.html**) by appending a summary table at the bottom of the pie chart using the **mod** option in the filename statement in addition to the **notop** option in the ods html statement. The GIF html files are displayed in the main frame when the user selects its corresponding thumbnail image using an anchor tag **<A** around the thumbnail image source tag **<IMG SCR=**. A section of the resulting HTML code for Example 3.1 follows:

```
<HTML> <HEAD> <TITLE>sasthumb</TITLE>
<BASE
HREF='http://diva/tmp/tp1263821879.2/'>
<BASE TARGET=view_frame> </HEAD>
<P ALIGN=CENTER>
<A HREF="gchart.html"><IMG
SRC="fgchart.gif"></A>
</P>…
</HTML>
```

We were limited to creating temporary drilldown HTML files for this report because we used the SAS **webframe** driver that automatically creates these files for us. Although the processing time for these reports runs longer than others, it still falls within the required 60 seconds.

***Caveat*:**
In our development environment, we experienced a problem for our dynamic web pages that point to temporary HTML files containing text and graphic output. Since the graphs are generated on the PC, we needed to somehow get them up to the AIX server where the web server runs. Normally, you could use the FTP option in the filename statement, however the ODS statement does not support FTP and graphics. To solve this issue, we manually FTP'd our dynamic graphics up to the web server.

Filename Statement with FTP Option – Box 3.1c

```
filename HOSTTXT ftp '/opt/sas/tmp/host.html' options;
filename HOSTGRPH ftp '/opt/sas/tmp' options;
goptions device=gif gsfname=HOSTGRPH;
 ods html body=HOSTTXT gpath=HOSTGRPH;
   proc print data=work;
   run;
   proc gchart data=work ;
     vbar x ;
   run;
   quit;
 ods html close;
-------------------------------------------------------------------------------
SAS LOG:
ERROR: Unable to allocate graphics device I/O.
ERROR: Unable to initialize graphics device.
```

Filename Statement with Manual FTP – Box 3.2c

```
filename PCGRPH 'c:\temp';
filename HOSTTXT 'c:\temp\mypage.html';
ods html body=HOSTTXT gpath=PCGRPH;
   …SAS print and graph procedures
 ods html close;

filename ftp1 'c:\temp\ftpit.bat' ;
filename ftp2 'c:\temp\ftpit.scr' ;

data _null_;
    file ftp2;
    put "open ozaki <username><password>";
    put "cd /opt/sas/tmp"; put "lcd c:\temp"; put "prompt";
    put "type image"; put "mput *.gif";
    put "type ascii"; put "mput *.html";
    put "quit";
  run;

data _null_;
   file ftp1;
   put 'ftp -s:c:\temp\ftpit.scr'; put 'exit';
run;
```

The FTP option in the filename statement lets you access information on other machines using TCP/IP. You must have TCP/IP software and a WINSOCK.DLL installed on your local machine and be able to connect to a machine that can function as an FTP server.

Box 3.1c shows that merely referencing the filename **HOSTTXT** in the **body=** option of the ds statement will work well with non-graphic output. However, using the **gpath=** option in the ods statement to reference the filename **HOSTGRPH** will give you an error because there is no support for graphics and FTP files.

As a result, we were forced to take a different approach to FTP our files. We decided to create the output locally and then write a script to manually FTP the files up to the host.

In Box 3.2c, the output html files are stored on the local C:\ drive. Then, two temporary files are created via data _null_. The **FTP1** file (ftpit.scr) actually does all the FTP work; whereas, the **FTP2** file (ftpit.bat) is simply a batch file that executes FTP1. The newly created FTP script can be executed in SAS using the following statements:

```
options noxwait;
x 'c:\temp\ftpit.bat';
```

EXAMPLE 4: CREATING PEMANENT AND DYNAMIC HTML USING SAS PROVIDED JAVA APPLETS

The batch generated Web page in Figure 4.1 uses the CSF (Critical Success Factor) JAVA applet provided by SAS Institute Inc. This applet is generated in a permanent web page that links to permanent drilldown URL's for detailed information. In Figure 4.1, we use the applet several times across different machines and performance measures. A section of the HTML code the for CSF applet follows:

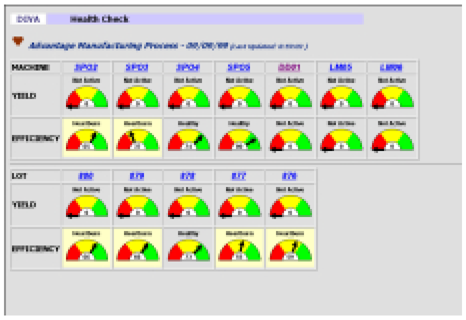


Figure 4.1 - Batch Web page with the SAS CSF Applet

```
<APPLET
CODE="com.sas.net.sharenet.samples.CSF2.class
"
NAME="CSF2TEST" CODEBASE="/sasweb/graph"
ARCHIVE="/sasweb/graph/js11csf.jar" >
…<param name= options…
<PARAM NAME='url'
value='http://diva/machinedata.txt'>
</APPLET>
```

The applet obtains the data it needs to generate the CSF regions and value through HTTP URL access to **machinedata.txt** referenced in the **<param** tag above. Although this applet also supports data access via SAS/SHARE ShareNet JDBC driver with SAS data sets, we were limited to using text files because our site does not have a SAS/SHARE software license. Regardless, the data **must** have the following format:

> csf *startValue endReg1 endReg2…endRegLast csfValue*

For example, if our file has the information, csf 0 25 50 75 100 89, our applet would have a start value of 0 that ends the first region at 25, the second region at 50, the third region at 75, and the last region at 100. The actual CSF value displayed by the arrow would be 89.

SAS Code for permanent HTML with CSF applet – Box 4.1

```
%macro csfapp (csfval,filein);
   put '<APPLET CODE="com.sas.net.sharenet.samples.CSF2.class"
';
   put 'NAME="CSF2TEST" CODEBASE="/sasweb/graph"';
   put 'ARCHIVE="/sasweb/graph/js11csf.jar" WIDTH="100"
HEIGHT="80">';
   if &csfval < 25 then
     do;  put '<param name="label" value="Region1">';
          put '<param name="backColor" value="CC99CC">';
     end;
   ...other conditional statements...
   put "<param name='url' value='http://diva";
   put "&filein";
   put "'></applet>";
%mend;
%mkhtml
```

A series of put statements within a data _null_ step were used to generate the HTML for Figure 4.1. Due to the frequency that the CSF applet is used for this report, a macro **%csfapp** was created to include the code within the **<APPLET** tags shown in Box 4.1. The macro variables passed to **%csfapp** include the actual CSF value, used for conditional processing, and the text file name.

This report checks the current health of the manufacturing process and is a "canned" report that the users would generate frequently. Therefore, we decided to create it as a permanent web page that gets updated as often as we refresh the DIVA data.

COMPARISONS

The four examples described in this paper explain different ways of delivering information that are as dynamic as the data behind the reports. The choices you make as a developer depend on the SAS procedures used, the power of the hardware, number of users, disk space availability, size of data sets, etc. Each of these considerations will contribute to how you route your HTML and images from the SAS Application dispatcher to the client Web browser. The ideal assumption is that there are enough hardware resources to enable all Web pages to be generated dynamically at each request. In the real world this is not the case. With limited hardware and the way the Web server and client communicate, this ideal is not always possible.

Batch Generated Permanent Web Page.

Batch generated permanent Web pages are used when these pages will have a large number of requests and/or take up a large amount of time and computing resources to create.

When the information you want to present is of common interest to many members of your organization, batch generated permanent Web pages are a good choice. The DIVA project presented status information for each of the machines on the production line in this manner (Example 4). These reports were of interest to all people using the DIVA application. By generating these reports and saving them on the Web server the need to process the same information over and over again was eliminated. This also allowed these reports to automatically refresh as new data was loaded into the DIVA databases, with the Web browser display automatically refreshed as well.

Example 2, the historical summary data, is also an example of permanent web pages. The DIVA application did not have the resources to keep data for more than a year. To provide comparisons of summary performance information, reports for historical data were created and permanently stored on the Web server before the data was archived and deleted from DIVA.

If processing the report takes a lot of time and the information is useful to a number of people, then having your reports stored on the Web server definitely makes sense here. The tradeoff is the disk space required, plus the additional resources, software and personnel, to automatically generate and monitor any process to create these reports.

Dynamically Generated Web Page

Dynamically generated web pages are great for when you have a lot of data but people want to be able to slice it in ways that are meaningful to them. Instead of generating every possible report, or every possible combination of a single report, using dynamically generated Web pages allows you to spend resources on just those reports people want to see.

When only a small percentage of all combinations of the reports are going to be used, or certain reports are used infrequently, then it makes more sense not to create these reports unless there is a request for them. This also helps to keep your batch processing time to a minimum by not generating unnecessary reports. Also, if all of the reports were created there may not be the disk space necessary to store these reports. This is especially true for any historical data that may be available. In most situations it is better to allow access to historical detailed data through dynamically generated reports as requested. DIVA does allow access to detail data through dynamically generated reports, but because historical detail data is archived and deleted from the system after 13 months, summary historical information was provided as permanent Web pages.

The dynamic examples of the Selection Window (Example 1) and the Pie chart (Example 3) use two different ways of dynamically generating the HTML. Example 1 does not use any temporary file storage and Example 3 does use temporary files to store the dynamically generated Web pages. To understand why all dynamically generated web pages can not be created without intermediate storage as temporary files, it is necessary to understand how Web pages are delivered from the Web server to the browser.

An example of delivering a Web page from the Web server at the simplest level is a Web page containing text and a graphic. When the browser requests the Web page, the Web server sends to the browser the HTML for the Web page. However, the image on the Web page is not sent at this time. The HTML includes the text to be displayed. The HTML also describes where on the Web page the image is to be displayed and where the image is located on the Intranet. The browser then has to make another request to the Web server for the image, and the image is then sent to the browser and displayed on the Web page in the location described by the HTML.

If you have a simple request where SAS is generating tabular information along with a graph, then the broker can request the SAS Application Server to generate the HTML for the tabular data. The SAS program generating the tabular data would also output an HTML link to call the SAS Application Server again, once the HTML has been delivered to the browser. Tthis time the broker would have the SAS Application Server run the SAS code to generate the graph. This graph would then get passed along the line to the browser.

This process worked fine in the DIVA application because these two SAS programs were able to execute and return a complete Web page in less than five seconds. However, this process means that the data had to be subset twice by the given parameters required for the report. If an application has to subset large amounts of data or has other time consuming processing then it is a waste of resources to have to do this twice, and can take a long time. In these cases, it is better to write out both the HTML for the tabular data and the graph image to temporary files and then have the Web Server tell the browser to point to the location of these temporary files.

The reason for the double processing is because the SAS Application Server resets between requests and the request for the tabular data is separate from the request for the graph. It would be possible to create a SAS library to store intermediate data sets but the DIVA project did not take this approach. Instead, DIVA uses temporary directories on the Web Server to store these reports.

Another reason for using temporary directories is illustrated in Example 3. Some of the capabilities of the Output Delivery System capabilities allow HTML framesets to be created with an index or thumbnail index along the left hand side of the Web page. SAS creates all of the required HTML and images at one time and the HTML links inside of these documents point to the appropriate image or report HTML file. Remembering how Web pages with images are sent to the browser it is impossible to send all of this to the web browser at one time. After receiving the initial HTML, the browser then requests each image and these have to be available.

Using temporary files requires some thought as to application requirements. Temporary files need to be unique. If there are multiple requests at the same time, and the SAS Application Server is writing to the same temporary files then it is impossible to know which request will be returned to the browser. DIVA creates temporary directories based on a SAS DATETIME value. This provides a unique temporary directory name to store the files for each request.

Also, how long do are the temporary files kept? If the BACK button on the browser is clicked, do you want to have the Web pages in these temporary files redisplayed? Or has the data changed so the temporary Web pages need to be expired and any attempt to redisplay them will require a new request to be generated by the SAS Broker? How much space do you have for temporary files on the Web Server? These temporary files will obviously need to be deleted and DIVA uses a small PERL script to delete temporary directories after a specified time.

If your Web Server and SAS Application Server are not on the same machine, then how can the Web Server access temporary files? One way DIVA does this in the development environment is to FTP the files from the machine where the SAS Application Server resides to the machine where the Web Server resides. But NFS mounted directories might be a better alternative, depending on your hardware setup.

SAS Provided JAVA Applets

DIVA uses two of the JAVA applets provided by SAS institute. The CSF applet (Example 4) is used to provide some neat traffic lighting capabilities with update. The graph applet is not described as an example in this paper due to page limits. The applet could be an entire ten pages in itself. The graph applet is an excellent way to provide a set of data to the browser that allows it to be subset and displayed in different ways without having to make additional requests to Web Server and SAS Application Server for additional processing. A link to the SAS web site describing the applet is in the Reference section.

CONCLUSION

The capabilities of SAS Version 7 with the ODS and the SAS/QC and SAS/GRAPH procedures adds immense functionality to an Intranet application. These four examples described can give you a feel for some of these capabilities and some of the questions you need to consider when planning the development of your own Intranet application.

The DIVA application was developed using the following

REFERENCES

http://www.sas.com/rnd/web/dispatch20/index.html (Application Dispatcher 2.0)
http://www.sas.com/rnd/web/dispatch20/oview.html (Overview of Application Dispatcher)
http://www.sas.com/rnd/web/dispatch20/how.html (How the Application Dispatcher Works)
http://www.sas.com/rnd/web/dispatch20/require.html (Requirements for Application Dispatcher)
http://www.sas.com/rnd/web/dispatch20/security.html (Application Dispatcher Security)
http://www.sas.com/rnd/web/dispatch20/tshoot.html (Application Dispatcher Troubleshooting)
http://www.sas.com/rnd/web/dispatch20/glossary.html (Glossary of Terms)
http://www.sas.com/rnd/web/graphapp/index.html (GraphApplet)

ACKNOWLEDGMENTS

Thanks to everyone who contributed to the success of the DIVA project.

Dave Glemaker, SAS Institute Inc.
Julie Licht, Roche Diagnostics
Kami Parker, Technical Writer
David Roy, SAS Institute Inc.
Bill Walters, Roche Diagnostics

CONTACT INFORMATION

Thomas Kunselman has over 12 years of experience analyzing data and developing applications using many of the SAS system products. He has experience working with varied types of data, including institutional research and assessment, health care and phramaceutical, and manufacturing process data.

Thomas Kunselman
P.A.I.R. Solutions, Inc.
Tek@pair-solutions.com
1.877.226.8135
http://www.pair-solutions.com/sas

Almira A. Lyst has been a statistician at Roche Diagnostics (formerly Boehringer Mannheim) for over 7 years in both the R&D and Manufacturing environments. She is currently a process analyst in the Process Improvement group and provides statistical support to managers and engineers in the production area. Almira has obtained an M.S. degree in Applied Statistics and has been working with SAS for approximately 10 years.

Almira A. Lyst
Applied Statistician
Roche Diagnostics
1.317.576.3310
mia.lyst@roche.com

Donald L. Penix, Jr. (D.J.) has over 7 years of experience analyzing data and developing applications using many of the SAS software products. Although he worked mainly with clinical trial data, he has most recently been involved with process manufacturing data. He is currently an independent consultant with his company Pinnacle Solutions, Inc.

D.J. Penix
Pinnacle Solutions, Inc.
dj@psiconsultants.com
1.317.590.4695
http://www.psiconsultants.com