

Web-Enabling SAS® Applications

Bradley W. Klenz, SAS Institute, Inc., Cary, NC

Introduction

SAS Institute now offers a number of tools for Web enablement of SAS software applications. In the field, these tools must be used to build applications. This paper will guide in the selection of the appropriate tools for particular applications. A few common applications will be outlined and development strategies presented. The advantages and disadvantages for different tools are discussed in the context they are presented. The tools and features discussed are available now in Release 6.12 of the SAS System. Applications presented are:

- Applications for Report Distribution
- Remote Data Entry
- Interactive user interface

Specific examples are presented to illustrate implementation details. Where design choices have been made, selection criteria are discussed.

The Basics

A number of tools are available for Web enablement. Many fall under the umbrella of SAS/IntrNet™ software. Others are available with base SAS software. In addition, new interfaces like the Jazz IDE development environment run outside the SAS System. The tools discussed include:

- HTML generation from a SAS DATA step
- Web Publishing Tools
- Application Dispatcher
- Jazz IDE Java development environment

'Web-enabling' an application can mean several different things, but in most SAS applications you are using the Web as an interface to a larger SAS application. In some cases it is creating a simple browser interface with HTML. More sophisticated interfaces may use Dynamic HTML. Interfaces that require a high degree of interactivity will use Java.

Applications for Report Distribution

Probably the most basic 'Web-enabling' of SAS applications is using a browser interface for report distribution. In the simplest form, existing paper reports are made available in HTML pages. The interactivity of the Web allows reports to be designed to provide better information through drill down and cross-referencing. In this instance, the 'report' is a set of HTML pages that are linked together to form a Web site. Two examples are presented, one using static Web pages, the other using dynamically generated Web pages.

Static Web Pages

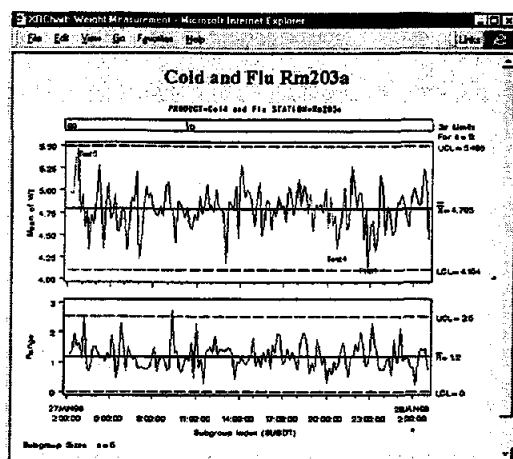
Static Web pages are HTML files that have been created at some time in the past and are stored on the Web server. After the pages are created, there is no further processing done when the user is viewing the pages. The report typically starts with the user viewing an index or summary page.

Outside Control Limits			
Headache	Headache Extra Severe	Cold and Flu	Cold and Flu Nighttime
Rm001a	Rm001	Rm001a	Rm001
Rm002	Rm001a	Rm001b	Rm001a
Rm003	Rm001b	Rm001c	Rm001b

Special Test Flagged			
Headache	Headache Extra Severe	Cold and Flu	Cold and Flu Nighttime
Rm001	Rm001	Rm001a	Rm001
Rm001	Rm001a	Rm001b	Rm001a
Rm001	Rm001b	Rm001c	Rm001b

In this example, a SAS program runs on a regular basis (e.g. nightly) to do quality control checks on a tablet production line. The program categorizes the results based on product and whether the production line is out of control or not. The program then generates the HTML file that is the index page. Three HTML tables are used. The first lists the production lines in which an out of control subgroup has been detected. The second table lists production lines where a test for special causes has been detected. Finally, the third table lists production lines that are in control. Each cell in a table is a link to a

page that displays a control chart of the selected process. Selecting a link displays a page as follows:



This page is an HTML file that was created by the previously mentioned SAS program. The page also displays a chart that is stored as a GIF file.

The SAS program that created the pages uses techniques similar to those found in many SAS reporting programs. The program does the initial processing using SAS DATA steps and SAS procedures. Then a series of DATA steps write the final report output. The unique portions that make the program Web-enabled are the following:

- SAS graphic output is directed to individual GIF files.
- The DATA steps that write the report also generate the HTML code for the HTML files.

To save the graphs as GIF files, the GSASFILE GOPTION and GIF device driver are used.

```

/*****
/* Assign gsasfile and driver to
/* create GIF
*****/
filename gsasfile &gifdir\&gifname";
goptions gaccess=gsasfile device=gif570;

```

One important point is that you must generate a separate GIF file for each graph. If you have a number of graphs that you are producing, you must either run the graph procedure for each file

or store the graphs in a catalog and use PROC GREPLAY to save each graph individually.

The DATA step that generates the report produces output that is written to HTML files. Similar to the graphs, a separate HTML file is created for each Web page. The following DATA step was used to create the index page:

```

/*****
/* Build index html page
*****/
data _null_;
  merge h1 hx caf cafn
    end=lastobs;
  by descending exflg;
  file "&pagedir\index.htm";

/*****
/* Start HTML page
*****/
if _n_ = 1 then do;
  put '<html>' / '<head>' /
    '<title>Tablet Production: Index'
    ' of graphs(weight)</title>' /
    '</head>' / '<body'
    background="greenwhite_paper.gif">' /
    '<h1 align="center">Tablet '
    'Production: Weight Measurements</h1>'
    ;
end;

/*****
/* Start HTML table
*****/
if first.exflg then do;
  put '<table border align="center"
    bgcolor="white">';
  if exflg=2 then do;
    put '<caption><b>Outside Control
      Limits</b></caption>';
  end;
  else do;
    if exflg=1 then do;
      put '<caption><b>Special Test
        Flagged</b></caption>';
    end;
    else do;
      put '<caption><b>In
        Control</b></caption>';
    end;
  end;
end;

```

```

*****/
/* Write column headers */
*****/
put '<tr><th
    rowspan=2>Headache<br></th>' /
    '<th
    rowspan=2>Headache<br>Extra
        Strength</th>' /
    '<th rowspan=2>Cold and
        Flu<br></th>' /
    '<th rowspan=2>Cold and
        Flu<br>Nighttime</th>
    </tr><tr></tr>';

end;
put '<tr>';

/*****/
/* Write values for table cells */
/*****/
if h1_st NE ' ' then do;
    htmname=scan(h1_hm,1) || '.htm';
    put '<td><a href="' htmname +(-1)
        '">' h1_st '</a></td>';
end;
else put '<td></td>';

if hx_st NE ' ' then do;
    htmname=scan(hx_hm,1) || '.htm';
    put '<td><a href="' htmname +(-1)
        '">' hx_st '</a></td>';
end;
else put '<td></td>';

if caf_st NE ' ' then do;
    htmname=scan(caf_hm,1) || '.htm';
    put '<td><a href="' htmname +(-1)
        '">' caf_st '</a></td>';
end;
else put '<td></td>';

if cafn_st NE ' ' then do;
    htmname=scan(cafn_hm,1) || '.htm';
    put '<td><a href="' htmname +(-1)
        '">' cafn_st '</a></td>';
end;
else put '<td></td>';

put '</tr>';

/*****/
/* Finish HTML table */
/*****/
if last.exflg then
    put '</table><br>';

```

```

*****/
/* Finish HTML page */
*****/
if lastobs then do;
    put '</body>' / '</html>';
    ;
end;

h1_st=' '; hx_st=' '; caf_st=' ';
cafn_st=' ';

run;

```

SAS programmers may have experience using a DATA step to generate additional DATA step code or macro code. The previous program works in a similar manner. As the file is written, the HTML tags are embedded within the values being written. In this manner, the information in the report is dynamic, but the end result is a static Web page that is stored.

Static Web pages have some real advantages for Web enabling applications. They provide a good starter project for making a Web-enabled application. Since the applications are straightforward and use programming techniques that many SAS programmers are familiar with, static Web page applications provide a lot of 'bang for the buck'. For the report users, a more interactive report interface is used and the pre-generated Web pages provide a quick response time. In addition, Web server resource use is low when browsing static Web pages.

Some of the drawbacks of static Web pages are that the generation program must generate all possible paths up front. Since each page (and graph) must be stored, there is potential for a large number of files to be generated. Also, the static Web pages have no ad-hoc capability and the data is current only through the time that the page was generated.

Another additional tip is that the HTML pages do not have to be created from scratch in the DATA step. An HTML editor can be used to create a template for the Web page. A DATA step could then read in the templates and create the actual HTML files.

Dynamic Page Generation

There are some situations where report distribution through static Web pages is not desirable. In some cases there are numerous static pages created, even though only a few are referenced on a regular basis. In other cases, the data presented on the Web page must be the most currently available. It is also possible that the selections available to the user need to be based on the most current data as well. Application requirements such as these can be handled using dynamic page generation. With dynamic page generation the Web pages are not created until the user actually requests the page to be viewed. Instead of a static HTML file that resides on the Web server, the HTML is written from the program that creates the page directly to the user's browser. No intervening HTML file is stored.

The ability to write SAS program output directly to the user's browser is available through the SAS/IntrNet Application Dispatcher. The Application Dispatcher uses a CGI broker program (SAS Application Broker provided with SAS/IntrNet software) to route the page request to a SAS Application Server that is running. The Application Server executes the requested SAS program, which then writes HTML back to the user.

In the example application, the page displayed to the user looks the same, whether it is a static page or a dynamically generated page. The difference is the HTML link used to call the page. A link to a static page is as follows:

```
<a href="g14.htm">Rm201a </a>
```

A link that references a page generated dynamically looks like the following:

```
<a href="/cgi-bin/broker.exe?
_service=default&
_program=sug.sugispc1.sas&
product=Headache&
station=Rm201a">Rm201a </a>
```

This link contains a few pieces of information. First, the broker.exe program is referenced instead of an HTML file. After the broker.exe reference are some variables expected by the Application Dispatcher. One variable (_program) indicates the SAS program to run in the SAS Application Server. Additionally, other

variables (e.g. product and station) can be specified that are passed to the requested program as macro variables.

When the requested SAS program runs in the SAS Application Server, there are facilities available to write the results back to the user's browser. A special file reference is created by the Application Dispatcher called _WEBOUT. The SAS program writes the resulting HTML to this file reference and the Application Dispatcher then handles the details of getting the results to the user. Note that the output written to this fileref has the HTML tags embedded in it. An example of how this is used is shown below:

```
data _null_;
  file _webout;

  product="&product";
  station="&station";

  put 'Content-type: text/html';
  put ;

  put '<html>' / '<head>' /
    '<title>XRChart: Weight
      Measurement</title>' /
    '</head>' / '<body>' /
    '<h2 align="center">' product
    station '</h2>'
    ;
  put '</body>' / '</html>';
run;
```

Through the Application Dispatcher, Web pages can be dynamically generated using the most current data available.

As shown, the SAS programs used to generate the output are a little more complex than generating static Web pages. Another issue that must be addressed is the installation and administration of a SAS Application Server. Applications can be setup to use a 'socket service' or 'launch service'. Socket service requests go through a single SAS session and are useful for page requests that can be handled quickly. Launch service requests can spawn a separate SAS session. Launch service is helpful when a program that requires a longer time to run is requested so that it will not hold up other page requests. The SAS CGI broker configuration file has the ability to set up multiple services and balance requests between

then. Please see the SAS/IntrNet documentation for a more thorough discussion of this topic.

Drill Down

One other feature of the report distribution example is a drill down feature. This allows the user to see the data that went into the control chart. The data are displayed using an HTML table with values from a SAS data set.

SUBDT	PRODCODE	OPERATOR	SHIFT	LOT	WT	TH	HD
27JAN1998 10:30:00	CAF	BRG	1	77	4.54	0.75	12.58
27JAN1998 10:30:00	CAF	BRG	1	77	4.71	0.91	10.5
27JAN1998 10:30:00	CAF	BRG	1	77	4.71	0.87	9.98
27JAN1998 10:30:00	CAF	BRG	1	77	4.18	0.96	10.44
27JAN1998 10:30:00	CAF	BRG	1	77	5.65	0.96	11.5
27JAN1998 10:40:00	CAF	BRG	1	77	5.43	0.85	11.71
27JAN1998 10:40:00	CAF	BRG	1	77	5.47	0.92	11.01
27JAN1998 10:40:00	CAF	BRG	1	77	4.76	1.05	10.73
27JAN1998 10:40:00	CAF	BRG	1	77	4.69	0.79	11.21
27JAN1998 10:40:00	CAF	BRG	1	77	5.01	0.9	10.12
27JAN1998 10:50:00	CAF	BRG	1	77	4.35	0.9	9.9
27JAN1998 10:50:00	CAF	BRG	1	77	4.51	0.74	9.27

This example makes use of a SAS Web Publishing Tools macro, called the Data Set Formatter macro. The macro takes a SAS data set and generates an HTML table with the data values. Macro parameters can control appearance, which variables are displayed, and subsetting using a WHERE clause. The code to create the above table is as follows:

```

/*****
/* Run Web Publishing macro to
/* generate the HTML table
*****/
%ds2htm(data=drilldat,
        runmode=s,
        openmode=replace,
        htmlfref=_WEBOUT,
        bgtype=image,
        bg=
        /mycustom/sug/greenwhite_paper.gif,
        center=y,
        var=subdt prodcode operator
        shift lot wt th hd,
        round=2,
        tbbgcolor=white
);

```

Using the Web Publishing Tools, Web pages can be created without a DATA step to generate HTML. No knowledge of HTML is necessary and an interactive development interface is available. In addition to creating tables from

data sets, there are Web Publishing macros to capture SAS procedure output and create HTML tables from the TABULATE procedure output. Web Publishing macros can write the HTML to a file for a static Web page or to the Application Dispatcher for a dynamically generated Web page.

Another tool for getting SAS data set values is htmSQL. This allows SQL statements that reference SAS data sets to be embedded in HTML pages. Since the example application is already using the Application Dispatcher, the drill down portion was implemented as an Application Dispatcher program.

Remote Data Entry Applications

Another common application need is data entry from a remote location. This is a good candidate for a Web-enabled application because the user requirement is not complex and a simple interface is all that is needed. Additionally, there is no need to install the application on the user's machine. The example application demonstrates this application type by providing an interface to enter patient demographic information for a pharmaceutical clinical trial.

Study: ZMP-901
Patient Demographics

Supply values for the fields below. Then select 'Add Patient'.

Patient ID Number:

Patient Initials:

Center:

Investigator:

Patient Gender:

Patient Race:

Patient Date of Birth:

The application uses an HTML form to allow the user to input data values. When the form is submitted (by selecting the 'Add Patient' push button), an Application Dispatcher program runs. This program performs some data validation, updates a SAS data set, and provides feedback to the user as whether the update was successful. The user's data values are passed to the program through macro variables. There is a macro variable for each field in the HTML form and the Application Dispatcher transfers the value from the HTML form to the Application Server that is

running the program. The following shows that code to update the data set:

```

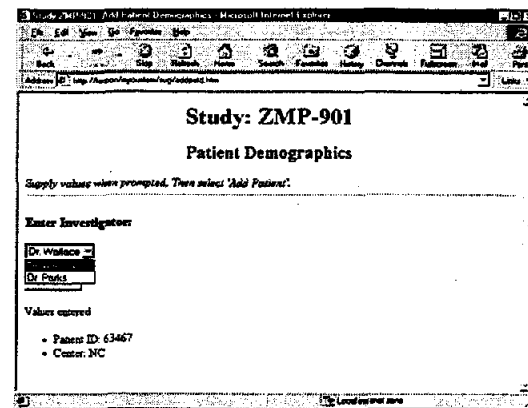
/*****
/* If HTML form values are OK then add*/
/* to SAS data set                      */
/* Macro variables are set by          */
/* Application Server from name/value */
/* pairs in HTML form.                  */
*****/
proc sql noprint;
    insert into patdata.patinfo
        (patid, center, inv, patinit,
         patsex, patrace, patdob)
        values ("&patid",
               "&center",
               "&inv",
               "&patinit",
               "&patsex",
               "&patrace",
               "&patdob");

```

This application is a straightforward design to allow simple data entry. This type of application would be a good candidate for a socket service because the transactions are small and quick response is necessary. If a more involved data entry system is required, then the application will need multiple Web pages to collect the data values. Also, the user does not get feedback on incorrect values until the form has been submitted.

Dynamic HTML

A more robust data entry application would need to be able to present selection lists to users based on previously collected values. Additionally, it would be useful to provide immediate feedback to the user for correct value selection. Dynamic HTML pages can perform these tasks. Dynamic HTML allows the Web page to dynamically alter itself without reconnecting to the Web server. In the example application, the selection of doctors is narrowed down based on which center the patient is at.



Dynamic HTML allows a more complex interface where the user can have many interactions with the Web page without requiring Web server hits.

One major drawback of Dynamic HTML is that it is not standardized and requires the most current Web browsers. It also requires more complex programming than standard HTML.

```

<html>
<head>
<title>Study ZMP-901: Add Patient
    Demographics</title>
<script language="JScript">
<!-- addInv function saves Investigator
    value and-->
<!-- resets HTML to prompt for Patient
    Initials -->
function addInv () {
    document.all.inv.value =
        document.all.uservalue.value;
    document.all.vallist.innerHTML =
        document.all.vallist.innerHTML +
        "<li>Investigator: " +
        document.all.inv.value;
    document.all.prompt.innerText = "Enter
        Patient Initials:";
    document.all.editval.innerHTML =
        "<input type='text' size='20'
        id='uservalue'>"
    document.all.cbutton.outerHTML =
        "<input id='cbutton' type='button'
        value='Continue'
        onclick='addInit()'>";
}
... additional code omitted

```

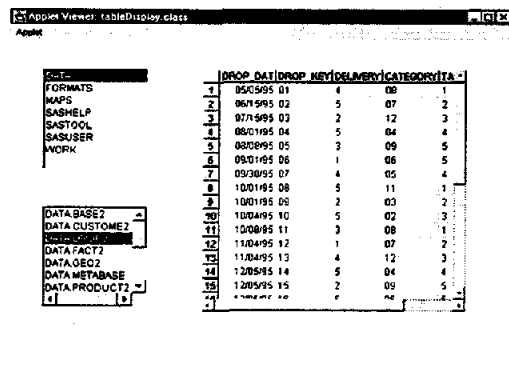
Dynamic HTML is a potential tool if you know that all the application users will have the same browser and have the most current version

(Netscape Navigator® 4.0 or Microsoft Internet Explorer® 4.0).

A SAS program could also generate dynamic HTML. This would allow output to be presented that had much interactivity.

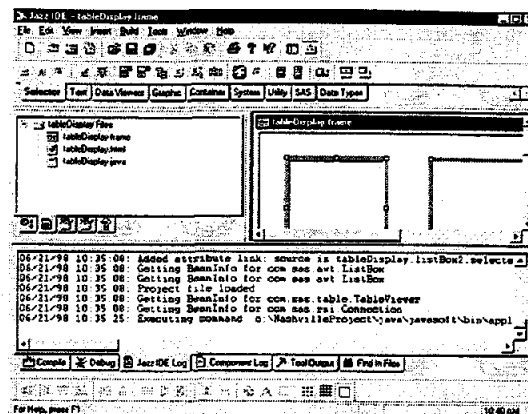
Interactive User Interface

Some applications require a high degree of interactivity in the interface with the user. Dynamic HTML allows more interactivity than standard HTML, but will only work on certain browsers. Java applets allow Web pages to have interactive user interfaces and are supported by most currently installed browsers. An example application allows a user to navigate a collection of SAS data libraries and select data sets for viewing. When viewing the data set, the user can scroll throughout the data set as desired.



Building such applications requires more advanced programming skills than HTML Web pages. Also, applet download times can make the application slow to come up.

To build a Java application, the programmer usually uses a packaged development environment. SAS Institute provides a integrated development environment currently called 'Jazz IDE'. This development environment provides normal Java programming support in addition to a set of SAS specific components that greatly enhance the ability to create applets that use the functionality of SAS software. Other Java development environments include Microsoft Visual J++® and Symantec Cafe®.



The SAS server support to Java applets is available through the SAS/CONNECT® driver for Java and the SAS/SHARE® driver for JDBC. The SAS/CONNECT facility allows the Java applet to submit code to a SAS session, similar to the way SAS/CONNECT works for SAS system client/server applications. The SAS JDBC classes provide access to SAS data sets through a JDBC interface.

Conclusion

Example applications have been presented to illustrate different manners in which SAS applications can be Web-enabled. Each example shows how various tools were used to meet the application requirements. Additional information was provided to explain why various design options were chosen.

Initial SAS Web projects can be implemented with a very reasonable investment of resources. As project requirements grow, a number of interface options are available and applications can be scaled up in a reasonable manner.

Feel free to contact the author at:

Bradley W. Klensz
Senior Quality Technology Specialist
Statistical Quality Improvement R&D
SAS Institute, Inc.
SAS Campus Drive
Cary, NC 27513
Email: brklen@wnt.sas.com

Additionally, you can request (by email) a ZIP file containing sample code used in this paper.

SAS, SAS/IntrNet, SAS/CONNECT, and SAS/SHARE are registered trademarks of SAS Institute, Inc.

Netscape Navigator is a registered trademark of Netscape, Inc.

Microsoft Internet Explorer and Visual J++ are registered trademarks of Microsoft, Inc.

Symantec Café is a registered trademark of Symantec, Inc.