Paper 193

Tools for Dynamic Web Publishing

Sarah A. Calhoun
Emily O. Kistner
Sally S. Muller
University of North Carolina, Chapel Hill, NC

Abstract

With the implementation of the SAS/IntrNet™ product on the University of North Carolina's central computing server, faculty, staff, and students can write SAS® programs that create customized, dynamic Web pages. As members of the UNC Statistical Support Group, we teach monthly classes on developing "static" and "dynamic" Web applications using the Application Dispatcher®.

This paper describes the tools our students have found most useful in creating dynamic web pages. Examples will be presented that make use of these tools; and changes and enhancements with the Version 7 SAS/IntrNet product will be discussed.

Introduction

In April 1998 a new Central Computing service was approved for helping individuals and departments at UNC put their SAS applications on the Web. As the providers of that service, our job was to put together a framework to guide our users.

This paper defines the concepts and terms that form the basis of the Application Dispatcher technology. We describe the components of the HTML interface and the SAS Web Publishing tools. Examples are presented that make use of these tools; and the changes and enhancements with the Version 7 SAS/IntrNet product are discussed.

Concepts and Terminology

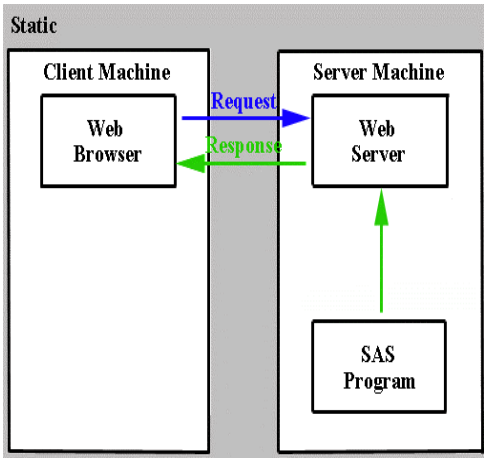
In our "dynamic" class, we teach two basic skills: (1) how to create an HTML form which will invoke a SAS program on a server via the Application Dispatcher and (2) how to construct a SAS program which will produce Web pages on demand from such an HTML form.

It is helpful to understand a few of the more common terms used in describing the Application Dispatcher:

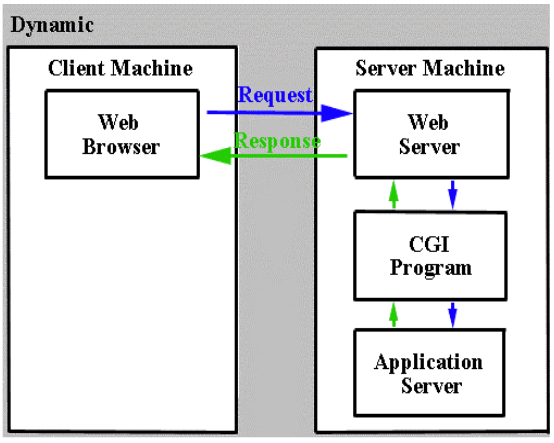
"Dynamic Web Application " is an application that includes HTML interfaces, SAS programs, and resulting Web-ready output. The HTML interface is the vehicle for user interaction with the SAS program. The SAS program generates the HTML output returned to the user's browser. Currently most dynamic web applications are created using either a CGI (Common Gateway Interface) program or a JAVA program.

When a Web browser requests an HTML page, it does so by submitting that page's URL (Uniform Resource

Locator) to a Web server. Upon receipt of this request, the Web server returns the page to the browser. This is known as a "static" Web application.



In contrast, when the URL submitted by the Web browser corresponds to a CGI program instead of an HTML page, the Web browser executes the CGI program. Typically, the CGI program invokes a session with a database or application server, which in turn processes the request from the client and returns the results to the browser via the Web server. This is known as a "dynamic" Web application.



"CGI (Common Gateway Interface)" is a programming interface that allows a web server to communicate with an external program. CGI programs are typically small, written in a script or high level language and reside on the web server to act as an interface between a web browser and a database server or application server [1].

"The Application Dispatcher" is a CGI gateway between a Web browser and the SAS System. With the Application Dispatcher, you can build dynamic web applications that allow your users to access the SAS System from their web browsers (whether they have SAS installed or not). You only need to the write the SAS program and the HTML interface. The details of the CGI script and the Application Dispatcher are handled by your administrator.

It is helpful to examine the dynamic web application process in terms of the role that the Application Dispatcher plays:

(1) One of the two components of the Application Dispatcher is the Application Broker, a CGI program. When your Web server receives a request that points to a CGI program, your Web server starts the Broker.
(2) The Application Broker processes the request and then sends the request to the Application Server, the second of the two Application Dispatcher components.
(3) The Application Server, either launches a new SAS session or wakes up an ongoing SAS session waiting for action.

(4) In either case, the Application Server processes the input from the Broker and creates a fileref named _webout, which points back through the Broker to the user's browser. In addition the server converts the user's request to a set of macro variables.
(5) Then the Application Server executes the specified SAS program, using those predefined macro variables. Thus, the macro variables are what convey information between your user's browser and your SAS program.
(6) The output from your SAS program is written to the _webout fileref which streams the output to the Application Broker. The Broker passes the output back to the Web server that sends the output to your user's browser.

Web Publishing Tools

Three Web publishing tools are provided by SAS Institute for converting SAS output into HTML output: the **Data Set Formatter, Output Formatter,** and **Tabulate Formatter**. These formatting tools are macros that pass parameters, which create and customize your HTML page. Each macro generates different HTML code, such as table and preformatted tags. For this reason, the macro you use is a function of the SAS procedure you wish to format.

The **Data Set Formatter** outputs your SAS data in the form of an HTML table. The information displayed is the same as the content generated in a PROC PRINT. You can create this table with only one call to the macro, **%ds2htm**. This macro requires certain parameters. The following simple example illustrates the Data Set Formatter. The bold text indicates the required parameters.

```
libname create 'afs/isis.unc.edu/eok';

%ds2htm(data=create.perm,
        htmlfref=_webout,
        runmode=s);
```

The first required parameter is **data=libref.dataset** that points to where your data is located. Next, the required parameters **htmlfref=_webout** and **runmode=s** indicate that the results will be passed to the Web browser via the Application Dispatcher.

The second macro formatting tool, the **Output Formatter,** was designed for use with all SAS procedures except PROC TABULATE and graphics procedures. This macro, **%out2htm**, creates an HTML page that looks identical to the SAS procedure output. With the Output Formatter, there must be two calls to the macro, as shown in the following example. Again note that bold text indicates required parameters.

```
Libname create ‘afs/isis.unc.edu/eok';
options formchar='|----|+|---+=|-/\*';

%out2htm(capture=on);

proc print data=create.perm;
run;

%out2htm(capture=off,
         htmlfref=_webout,
         runmode=s);
```

As the above example shows, the macro calls precede and follow the SAS procedure(s) you wish to display. The macro requires two of the same parameters as the Data Set Formatter: **htmlfref=_webout** and **runmode=s.** However, you must include an additional parameter to the first and second calls to the macro**, capture=value**.

The third and last macro is the **Tabulate Formatter, %tab2htm**. This formatter was designed to display the PROC TABULATE output. The procedure output is produced in an HTML table. As shown in the following example there are two calls to the macro, one before the procedure and one after the procedure. Bold text indicates required parameters.

```
libname create '/afs/isis.unc.edu/eok/';

%tab2htm(capture=on);

proc tabulate data=create.perm
  formchar='8283485868788898a8b8c'x;
  class SEX;
  var YEARS;
  table SEX, YEARS*(min mean max);
  title 'Tabulate Formatter Example';
run;

%tab2htm(capture=off,
         htmlfref=_webout,
           runmode=s);
```

Components of an HTML Interface

When creating the HTML interface, you will want to remember that the HTML interface calls the SAS program and the SAS program in turn generates the HTML output. One common HTML interface is an HTML form. The HTML form allows your user to send and request information from your Web page and consequently from your SAS program.

The information the user requests can be as simple as running an unmodified SAS program or as complicated as specifying variables and values for statistical analysis.

The most basic element of an HTML form is the submit button. This element requires specific HTML tags as shown in bold in the following example.

```
<html>
<head>
<title>Submit Button</title>
</head>
<body bgcolor=white>
<h2>Display Staff Information</h2>
<p>
<form action="http://statweb.unc.edu/cgi-bin/broker">
<input type=hidden name="_service" value="default">
<input type=hidden name="_program" value="lporter.submit.sas">
<input type=submit value="Run the Report">
</form>
</body>
</html>
```

The first necessary tag is the **<form action=server>** tag. The **action** points to the location of the CGI program, for example our CGI program is located at "statweb.unc.edu/cgi-bin/broker", therefore action=statweb.unc.edu/cgi-bin/broker. The next two required HTML tags are **<input name=parameter value=parameter>** tags. There are three required input tags. The first tag designates the service being run. The second designates the program being run. The last input tag creates the submit button.

To allow your user to create more complicated requests there are other HTML form elements. One such form element is radio buttons. With radio buttons you can, for example, allow your user to select a variable's value from a list of mutually exclusive items. As seen below in bold text, the HTML code for radio buttons is an extension of the submit button code:

```
<html>
<!—radio-->
<head>
<title>Student Information</title>
</head>
<body bgcolor=white>
<h2>Display of ATN's Statistical Support Group Student Assistants</h2>
Select a student to learn more about
<p>
<form action="http://statweb.unc.edu/cgi-bin/broker">
<input type=hidden name="_service" value="default">
<input type=hidden name="_program" value="lporter.radio.sas">
<!-- radio buttons here -->
<input type=radio name="first" value="EMILY" checked> Emily
<input type=radio name="first" value="SARAH"> Sarah
<input type=radio name="first" value="JUDD"> Judd
<input type=radio name="first" value="BOB"> Bob
</p>
<input type=submit value="Display Information">
</form>
</body>
</html>
```

Notice that within the input tag the **type=radio** and the **name=** and **value=** fields correspond to the variable name and variable value represented by the radio button. Hence each additional radio button requires its own input tag.

Once your user clicks the submit button, the HTML form is sent to your Web server. As described above the user's request (the HTML form) collaborates with the SAS program that you have written. The following is an example SAS program that would work with our example HTML form:

```
%global first;

title "Statistical Support Group
Information";

%ds2htm(data=webdata.rsgstaff,
      where=fname eq symget('first'),
      var=fname lname sex phone
      years, htmlfref=_webout,
      runmode=s);
```

Another useful HTML form element, for your HTML interface, is the selection list. Selection lists are pull down menus. Like radio button code, selection list code is an extension of the original submit button code as illustrated below in bolded text:

```
<html>
<!—single choice -->
<head>
<title>Select An Employee</title>
</head>
<body bgcolor=white>
<h2>Display of ATN's Statistical Support
Group</h2>
Select an employee to learn more about
<p>
<form method="post"
action="http://statweb.unc.edu/cgi-
bin/broker">
<input type=hidden name="_service"
value="default">
<input type=hidden name="_program"
value="lporter.selection.sas">
<!—Selection list here -->
<select name="first">
<option value="SALLY" selected> Sally
<option value="SARAH"> Sarah
<option value="DAVID"> David
<input type=submit value="Display
Information">
</form>
</body>
</html>
```

When specifying the selection list, the **select name**= tag indicates the variable name. The **option value=** tags specify the variable values.

We are not including an example SAS program here since the SAS program that we gave as an example with radio buttons would work here as well.

The third HTML form element that we will describe, is the text field. As with previous form elements, the text field code is an extension of the submit button code. The code for the text field is in bold in our example:

```
<html>
<!-- formtext.html -->
<head>
<title>Select a Student - Text Entry
Version</title>
</head>
<body bgcolor=white>
<h2>Display Staff List By Name</h2>
<p>
Please choose a name from the following
list:
<ul>
<li>Emily</li>
<li>Sarah</li>
<li>Judd</li>
<li>Bob</li>
</ul>
<form action="http://statweb.unc.edu/cgi-
bin/broker">
<input type=hidden name="_service"
value="default">
<input type=hidden name="_program"
value="lporter.text.sas">
Employee name
<input type=text name="first" size=4
maxlength=5>
</p>
<input type=submit value="Display
Information">
</form>
</body>
</html>
```

As you can see the text field input tag begins with **input type=text.** The **name="first"** field specifies the variable name, the **size=4** field determines the size of the text field on the HTML form, and the **maxlength=5** field limits the number of characters that can be typed into the text field. Again the same SAS program that we used in our earlier examples works here as well.

The final HTML form element that we will describe is the checkbox. Although the checkbox is quite similar to radio buttons, the items listed are not mutually exclusive and thus you may choose more than one item. As with the other HTML form elements we have discussed, the checkbox code builds upon the submit button code.

```
<html>
<head>
<title>Select Employee(s) - Multiple
Checkbox Version</title>
</head>
<body bgcolor=white>
<h2>Display of ATN's Statistical Support
Group Information</h2>
Select an employee(s) that you would like
more information on.
<P>
<form method="post"
action="http://statweb.unc.edu/cgi-
bin/broker">
<input type=hidden name="_service"
value="default">
<input type=hidden name="_program"
value="scalhoun.newcheck.sas">
<!-- Checkboxes here. -->
<input type=checkbox name="fname"
value="DAVE">Dave
<input type=checkbox name="fname"
value="LESLIE">Leslie
<input type=checkbox name="fname"
value="BOB">Bob
</P>
<input type="submit" value="Display
List(s)">
</form>
</body>
</html>
```

Note that each checkbox requires its own input tag. Within each input tag you need to specify **type=checkbox** for the name of the form element, **name=variable name** for the name of the variable, and **value=variable_value** for the variables value.

As with earlier HTML forms when your user clicks on the submit button, the HTML form is sent to the Web server and collaborates with the SAS program that you have written.

For this example a new SAS program is used because the coding required for checkbox is quite

different that the SAS code required for radio buttons, selection lists, or text fields. The checkbox code is shown in bold text in the example[2].

```
%macro newcheck;/* Variables that
should always exist */
%global fname fname0 fname1;
%if %superq(fname)= %then    /* No
selcetions*/
 %do; %local fname0; /* Dispatcher did
not create it */
%let fname0=0;
 %end;
%else
%if %superq(fname0)= %then
    %do; %local fname1 fname0;
    %let fname1=%superq(fname);
    %let fname0=1; %end;
data _null_;
 file _webout;
put 'Content-type: text/html';
put;
run;
%local i; %do i=1 %to %superq(fname0);
%ds2htm(data=webdata.rsgstaff,
            where=fname eq
"%superq(fname&i)",
            caption=Employee Information
for  %superq(fname&i), htmlfref=_webout,
runmode=s,
          openmode=append);
 %end;
%mend newcheck;
%newcheck
```

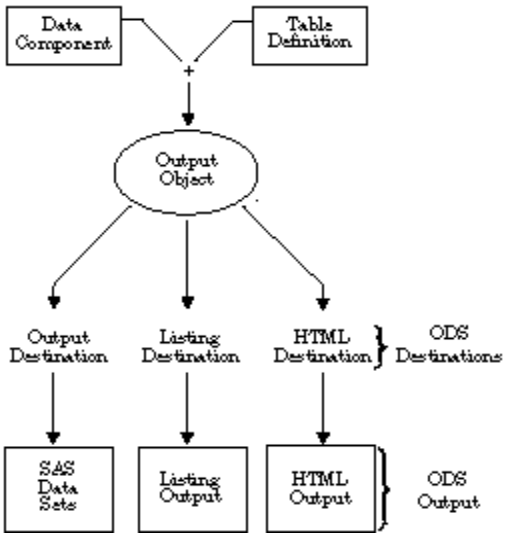
Dynamic Applications and Version 7

We began using Version 7 of the SAS System on our central statistical computing server in January 1999. However, SAS V7 requires SAS/IntrNet Version 2.0 and at the time of this writing we had not yet upgraded to Version 2.0. Nevertheless, we are able to make a few observations.

Our first observation is that the SAS Web publishing tools, the formatting macros, are included in V7.

The new way that you create HTML procedure output with V7 is through the "Output Delivery System" (ODS). Touted by many as the most significant enhancement in V7, ODS gives you control over the output from SAS procedures.

With earlier versions of SAS, each time you run a SAS procedure it writes the procedure output directly to a listing file. With V7 the procedure creates an "output object" for each piece of output to be displayed. The figure below shows some of the destinations that you can request.



This diagram comes directly from the SAS Institute's web page, http://www.sas.com/rnd/base.

As indicated in the figure you can request any or all of the following formats: (1) Standard SAS listing (what you are accustomed to and what you get by default) (2) HTML

for the Web (3) output data sets.

Output Destinations have default templates associated with them. In order to customize your ODS HTML output, you have to modify its template. The templates have default settings for such things as color and fonts. Our example in this paper uses the default template.

```
/* Create HTML files. */
ods html body = _webout (dynamic);

proc univariate data=testing.statepop;
   var citypop_90 noncitypop_90;
  run;
/* Close the HTML destination.  */
/* You must close this destination
before  */
/* you can browse the HTML files*/
ods html close;
```

This material uses information from the SAS Institute ODS Documentation. The document is currently located at the following URL:

http://www.sas.com/rnd/base/early-access/odsdoc/tw4566/index.htm

Conclusion

This paper described how to create an HTML interface for invoking a SAS program on a server via the Application Dispatcher. This paper also explained how to construct a SAS program, for producing Web pages on demand from such an HTML interface. Although the future of SAS Dynamic Web Publishing will change, these concepts will remain constant.

Acknowledgements

Don Henderson and Aaron Hill, for doing just a dynamite job at initially introducing us to the SAS/IntrNet product at our local SAS Users Group (RTSUG) meeting.

Warren Repole and Dan Horinek, SAS Institute "on site" Instructors. Their classes on "Using the Web Publishing Tools" and "Running SAS Applications on the Web" gave us the inspiration
to develop the campus service and write this paper.

Jude Lynn Redman, Chevell Parker, and David Shinn, SAS Institute Technical Support Group, for their many hours of help.

Chris Olinger and Paul Kent, SAS Institute Base SAS Development Group for their help with the Output Delivery System and Version
7 of the SAS System.

References

SAS Institute Inc. (1998). SAS/Intrnet Software: Delivering Web
Solutions, Cary, NC: SAS Institute Inc.

Running SAS Applications on the Web Course Notes, Cary, NC: SAS Institute Inc.

http://www.sas.com/rnd/web - this is the home page for the
SAS Web Tools

http://www.sas.com/service/techsup/

whatsnew_v7.html - this is the home page for Version 7 of the SAS System

http://www.sas.com/rnd/base - this is the BASE SAS home page and includes very extensive ODS documentation and examples

Mickey Waxman and Larry Hoyle (1998), "A Hands on Introduction to Creating Dynamic Web Pages," Proceedings of the Twenty Third Annual SAS Users Group International, 697-705 .

Donald J. Henderson (1998), "SAS/IntrNet Software: A Roadmap," Proceedings of the Twenty Third Annual SAS Users Group International, 23, 861-870.

Chris R. Olinger and Randall D. Tobias (1998), "ODS for Data Analysis: Output As-You-Like-It in Version 7,: Proceedings of the Twenty Third Annual SAS Users Group International, 23, 1271-1291.

Authors

Sarah A. Calhoun, Emily O. Kistner, and Sally S. Muller can be contacted at:

Room 28 Phillips Hall
CB#3455
University of North Carolina
Chapel Hill, N.C.

They can be reached at (919) (962-9803) or via e-mail at:

sarah_calhoun@unc.edu
emily_kistner@unc.edu
sally_muller@unc.edu