Paper 5-26

The Basics of Dynamic SAS/IntrNet ® Applications

Roderick A. Rose, Jordan Institute for Families, School of Social Work, UNC-Chapel Hill

ABSTRACT

The purpose of this tutorial is to introduce SAS® programmers to the components and processes required in order to create dynamic SAS/IntrNet applications. Dynamic SAS/IntrNet is a robust topic, covering everything from ODS and drill-down capabilities to htmSQL. This tutorial ignores many advanced topics and most optional components available in SAS/IntrNet. Instead, the instruction focuses on the basic skills needed to generate a simple dynamic application that utilizes the SAS Application Dispatcher, a web-based SAS processor.

Dynamic applications go beyond static Internet applications by allowing users from remote locations to process SAS programs and data sets online. The user does not need SAS installed on his or her machine, and the analysis is provided through the web browser rather than a SAS interface. This has limitations - the user can choose only from those options that have been prepared by the programmer, and its drawbacks – for instance, dynamic applications require significant server resources for processing. But it is a versatile means of implementing interactive SAS applications in a cost-effective manner.

Programming dynamic SAS/IntrNet applications is best left to the experienced SAS user. Web authoring experience is not required. I use only a minimal amount of HTML in this tutorial, and much of the required HTML can be produced using an HTML editor. The goal of this paper, and the reason for presenting it in this section of the conference, is to develop experienced SAS programmers into beginner-level dynamic application programmers. Suggestions and hints that are not essential to the tutorial are in italicized text.

This paper was adapted from a web site prepared for instructing users in dynamic SAS/IntrNet applications (referred to in this paper as the "companion web site"). It has links, programs and data sets that can be used as an instructional supplement to this paper. It can be found at http://ssw.unc.edu/workfirst/sasint/.

DYNAMIC INTERACTION

A dynamic application is a combination of web pages, SAS programs and a web server-based SAS component called the Application Dispatcher. The output – the page that your audience retrieves over the Internet – is created "on-the-fly" by the interaction of the web form with the application dispatcher, data and program:

- A user will select some information on a web form and submit it to the Dispatcher.
- The Dispatcher will pass this information, along with the SAS program, to an open SAS session.
- The SAS program will query the data set for this information, and a subset of the data set will be produced. This subset will meet the criteria established by the user on the form.
- A web page that contains an analysis of this subset will then be produced and displayed on the user's browser.

The above description of how SAS/IntrNet processes requests submitted by a form is simplified, and sufficient for this tutorial. The SAS Institute provides a more comprehensive description, including some illustrations (please refer to the URL in the reference section). For the most part, these can be ignored if you are not involved in the maintenance of the Dispatcher in your organization.

THE DATA STEP AS A FOUNDATION

Data steps are the foundation of SAS/IntrNet, just like in Base SAS. You normally use data steps to manipulate data and produce output data sets. Now you will add a multi-faceted web component. You will (1) use web information to manipulate data and then (2) instruct SAS to produce HTML output using a data step.

With dynamic applications there are 2 sources of information - a data set and a web form - that are required to make everything work.

- A Data Set – This is probably the concept with which experienced SAS programmers are most familiar. For each observation in the data set, the program will perform the actions specified in the data step or procedure.
- A Web Form - Even experienced SAS programmers may not have encountered web forms before. In addition to using a data set as a data source, the data step in SAS/IntrNet will use, as an additional source of information, a macro variable passed from a web form to the SAS program.

This constitutes most of SAS/IntrNet programming and will be covered in the next part of this tutorial.

DYNAMIC VS. STATIC

There are actually three sources of data. The third is "none". If you have no intention of using a form or outputting data elements to the web pages, but instead simply want to enhance your Web pages with web elements such as pictures, headers, footers, and other distractions, you'll use a data _null_ with no data set or web form. This is simply static web publishing, whereby permanent web pages are created and stored on a server for web access.

Most requirements for using the HTML Formatting Macros or ODS in dynamic applications carries over from static publishing. The little that must be done to modify them for dynamic processing will be explained further on, but the modifications are minor and the instruction is brief. Those of you with static SAS web publishing experience will undoubtedly be relieved about this! Those of you without any experience in static SAS web publishing will not find it difficult to follow, but you are encouraged to review static ODS HTML for further development.

THE WEB FORM

Your first task is to design a web form from which a user can select query criteria. Specifically, you need to (1) specify several required hidden form fields, (2) create a pool of query criteria from which the user can make selections, and (3) create a macro variable that corresponds to a field in your data set, with values corresponding to values of that field.

REQUIRED HIDDEN FIELDS

The following is a very simple web page that contains a form. The form consists of hidden input fields, radio button input fields and a submit button.

The following is required for making a web page:

```
<html>
<body>
```

The following FORM element and hidden input fields, which have three attributes - type, name, and value - are required for processing:

```
<FORM NAME="form"
ACTION="http://statweb.unc.edu/cgi-
bin/broker8">
<INPUT TYPE="hidden" NAME="_service"
VALUE="default">
```

```
<INPUT TYPE="hidden" NAME="_program"
VALUE="rarose.proj2.sas">
```

The code follows HTML standards for forms. The portions in **bold text** are subject to revision based on how your organization sets up the Dispatcher and libraries and what you choose to name the program. The remaining code can be copied *verbatim*.

QUERY CRITERIA
A pool of criteria from which the user can choose, and a submit button are required. This part is variable, and based on your needs. The criteria are created in HTML using *input* fields, much like the hidden ones above.

```
<input type = "radio" name = "vara" value
= "1"> 1 <br>
<input type = "radio" name = "vara" value
= "2"> 2 <br>
<input type = "submit">
```

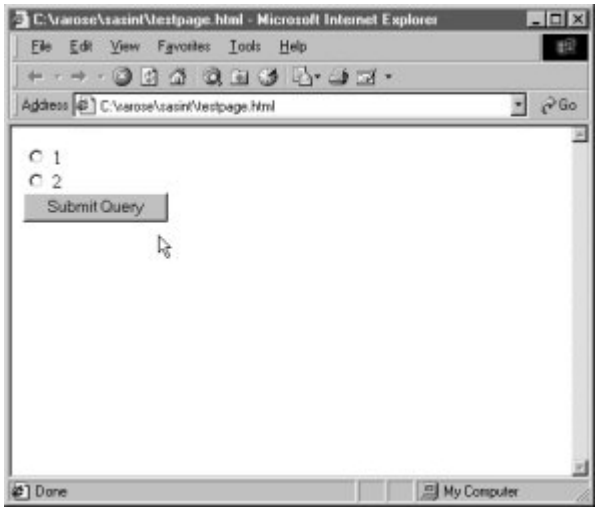
The input type employed here is the radio button (*type = "radio"*), from which the user can select only one option at a time. The query criteria also include specifications for what will become a macro variable.

THE MACRO VARIABLE
The name "vara" (*name = "vara"*) in the input fields will be invoked in the program as the macro variable. A field in your data set – one that is to act as a filter of observations - plays an important part in determining the values of each input field. The values given by the radio buttons (above, *value = "1"* and *value = "2"*) should, ideally, match the values of this field.

Required elements for closing the form and the page:

```
</form>
</body>
</html>
```

EXAMPLE
The HTML code above produces the web page pictured below.



When the user presses the button labeled "submit query", the information in the form is submitted, and the Dispatcher will look for a program called proj2.sas in the library given by my userID, *rarose*. You need to write this program yourself, and that is the subject of the next part of this lesson.

Other elements, such as images, other links, and text can be on the Form page as well. The page given above is the fully functional minimum and can be copied, with some exceptions, verbatim. Other methods of submitting information involve multiple choices (checkboxes) and variable fields (text boxes, whereby the user types in exactly what they want to see). Additional information about these options is available at http://ssw.unc.edu/workfirst/sasint/altinput.html.

THE SAS PROGRAM

As mentioned previously, the data step is the foundation of SAS/IntrNet programming. I assume that you know about data sets. You'll need at least one of those for processing data. In order to process the query submitted by the user on the form and produce output, the program requires the three items that are described in this section: (1) queries using the macro variable, (2) the content-type declaration, and (3) a method of analysis.

Note: Security constraints on macro variables in SAS/IntrNet have changed since SAS implemented version 8. Since you are not precluded from implementing these security precautions in version 8 (they are not necessary, but it does not hurt to add them), and some of you may have to use version 2.0, I have chosen to include them as part of the instruction. The SAS Institute web site explains this better if you feel you really must know. See http://www.sas.com/rnd/web/intrnet/dispatch20/progsec.html#amp.

QUERIES USING THE MACRO VARIABLE
If you've ever written a SAS program where you have specified an if-then clause such as

```
if var1 = 1 then do;
```

where var1 is a fieldname in your data set, then you know most of what you need to accomplish this task. The difference in SAS/IntrNet is to replace the digit 1 on the right side of the equality with the name of the macro variable (*vara*, in our example above). The Application Dispatcher will "retrieve" the macro variable from the form, and pass it, with the program, to the open SAS session. The data set will be queried for the specified value.

The importance of this concept cannot be understated. *var1* is a fieldname in our data set, and our macro variable is a character variable named *vara*, passed to the SAS system by submitting the web form. If the user selected "Choice 1", which has a value of "1", on the example form from the previous page, then the program will look in the data set for all values of var1 that have the matching value "1" and produce a corresponding subset.

To begin, construct a libname where the data set is stored, then the data statement that performs the query:

```
libname rose '~rarose/public_html/';
data one;
set rose.test;
if var1 = symget('vara');
run;
```

symget('vara') can be replaced with *%superq(vara)*, or if you are using the Application Dispatcher version 8, just *"&vara"*.

Please remember, as with all SAS applications, the variables on each side of the if-then must be the same type (you need to compare character with character and compare numeric with numeric, etc.). The macro variable retrieved from the form is, by default, a character variable. If the field in the data set does not agree, then simply change one of them.

THE CONTENT-TYPE DECLARATION
The standard destination for dynamic SAS/IntrNet output is *_webout*. You will use it whenever you need to send anything to the

web destination. The Dispatcher requires the following code, called the content-type declaration, in your program before any analysis is sent to _webout (for instance, it can follow the code above in the order presented in this paper):

```
data _NULL_;
file _webout;
put 'Content-type: text/html';
put;
run;
```

Do not overlook the last (blank) put statement in the data step.

A METHOD OF ANALYSIS

There are several options for performing an analysis: the data step, ODS HTML, the HTML Formatting Macros, and a combination of SAS procedures. If I could choose one word to describe your analysis options, it would be "versatile." I'll look at ODS HTML further on, and leave the rest to you.

You can also add static content to your output pages. The companion web site contains information about using HTML and DATA _NULL_ to add static content, such as links, images and header/footer text, to your dynamic web pages.

UPLOADING, TESTING & OTHER HINTS

For writing the program, several different applications are available. Of course, you can use SAS, but it is not necessary (although the enhanced editor, with its color-coded interface, can be useful). Any simple text processor will do. I use Notepad, a simple Windows95/98/NT text utility with which you are undoubtedly familiar.

After you write both the form and the program, you'll need to put them in their proper places on the server. When I teach the static lesson, I encourage students to use the SAS FTP Access Method. That will not work for dynamic SAS/IntrNet. I suggest you use WS-FTP or another FTP utility to upload both the web form and the program. Information on WS-FTP is available on the companion web site. Alternatively, you may be able to write the program in its place on the server if you can obtain that kind of access and you are familiar with the necessary text processors.

The form, which is a Web pages with a .html extension, is stored in your public_html directory for public access via the web browser. The program goes in your sascode directory, for access only by the application dispatcher. The data can be in either directory. However, if it goes in your public_html directory, then it can be accessed by anyone with web access. If it goes in your sascode directory, it can only be accessed by the Application Dispatcher. *DISCLAIMER: Much of this is subject to how your organization has set up the Dispatcher, so check with the responsible parties in your organization.*

To test your web page/program combo, go to the web form you've created, interact with it by selecting options, and then wait for the magic to occur. If your magic is anything like mine, it won't work the first time and you'll get incorrect or missing output. This is ok. Please refer to the next section.

DEBUG

Normally, when something goes wrong, you would access the SAS log to see what you've done wrong, fix it and then re-run the program. However, because you are using the application dispatcher to run the program, there technically isn't a SAS log to review.

The solution is to create a SAS log in the browser window by appending the diagnostic variable *_debug*, with the value 131, to the URL in the address bar (as highlighted in the illustration below, at the end of the string is probably easiest):

```
&_debug=131
```

This will append the contents of the log to the web-based output in the browser window, below what output is produced, if any. Then it will be simple to identify the problem, fix it, and then run it again.

It is not always necessary to go back to the form to verify that the modifications have worked. If you do not have to revise your query information, then just click the reload button on the browser. This will re-submit the information to the server using the same information, re-run the program, and reproduce the output. If the macro variable value needs to be changed then it can either be changed it in the URL of the output page (there is more on this option further on) or re-submitted on the form.

131 is not the only valid debug value. Please refer to http://www.sas.com/rnd/web/intrnet/dispatch/debuginp.html for more information.

DYNAMIC INTERACTION WITH ODS HTML

I encourage you to use ODS HTML to produce your dynamic analysis, as opposed to the HTML Formatting Macros (a brief lesson in the HTML Formatting Macros is available on the companion web site). ODS is more versatile in its output and easier to manipulate. *Also: There are economies to be had by learning and using ODS, as it has many other applications besides HTML output.*

MODIFY THE DESTINATION

Using ODS for dynamic applications is a matter of a simple modification in the output destination of the static version. This *must* come after the content-type declaration.

Take the following ODS HTML call (you can use *file* in place of *body*):

```
ODS HTML
body = ods2.html;
```

To modify this for dynamic output, simply replace the filename *ods2.html* with *_webout*.

```
ODS HTML
body = _webout;
```

The rest of what you need to know about ODS HTML is the same: after opening the HTML destination, you add some data manipulation and PROCs and then you close the destination. Closing the destination in dynamic applications is the same as it is in static applications:

```
ODS HTML CLOSE;
```

ODS HTML WITH FRAMES

A useful enhancement that ODS HTML has over the HTML Formatting Macros is the built-in ability to generating pages consisting of multiple windows or frames, as pictured below.

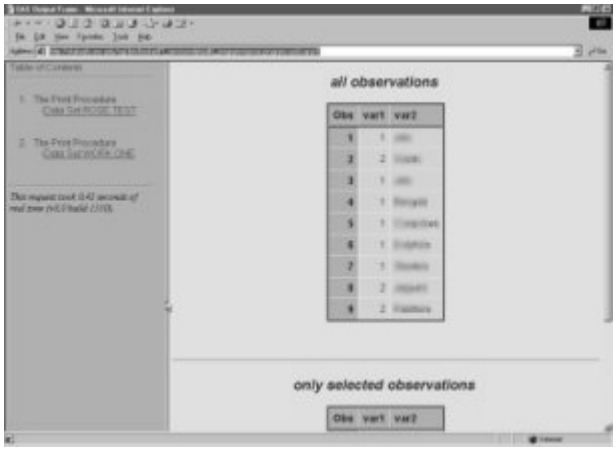
A browser window with two frames in it actually consists of three pages. The two pages shown are the body page, containing the

analysis, and a contents page, with links that allow the user to jump from section to section of the body page when selected. The third page holds the two of them together, side-by-side as they are.

Due to the temporary nature of the three files created in a dynamic application, modifying Dynamic ODS HTML to include frames requires a little more than simply modifying the *body* option. Instead, leave the *body* and *contents* options as they are in a static application, specify _webout as the *frame* option, and add a path option, exactly as it is represented below (you cannot use *file* in place of *body*):

```
ODS HTML
body = bods2.html
contents= cods2.html
frame=_webout
path=&_tmpcat(url=&_replay)rs = none;
```

For creating frames, the path option and all of its components are required. The illustration below shows the contents section, on the left with a gray background, and the body section with the lightly colored background taking up most of the window.



The SAS Institute has more information on using ODS with dynamic applications. See the URL in the reference section.

AN EXAMPLE PROGRAM

The following is a complete program incorporating all of the variables and procedures necessary to process a data set and produce dynamic output. For simplicity, the program will not generate frames. The web form from the first part of this paper is sufficient for processing this program. *At the companion web site, you can download the web form and the data set.*

First I need to include all of the necessary SAS code, as specified in the second part of the tutorial. This includes the libname referencing the data set and the DATA _NULL_ that opens the *_webout* destination and places the content-type declaration on the web page.

```
libname rose '~rarose/public_html/';

data _NULL_;
file _webout;
put 'Content-type: text/html';
put;
run;
```

Close the Listing destination (the Listing destination is, by default, open, and the SAS Institutes suggests that it should be closed before opening the HTML destination). Specify the HTML destination _webout.

```
ods listing close;

ods html file = _webout;
```

The following is a data step that manipulates data according to the form submitted by the user, and a set of PROC PRINTs that I use to illustrate that the macro variable accomplished its task by subsetting the data set. First I print the entire data set, then only the subset according to the query. The HTML destination is closed in the final line.

```
data one;
set rose.test;
if var1 = symget('vara');
run;

proc print data = rose.test;
title "all observations";
var _all_;
run;

proc print data = one;
title "only selected observations";
var _all_;
run;

ods html close;
```

AN ALTERNATIVE TO THE WEB FORM

I introduced in the third part of the dynamic tutorial the *required* code of a web form for processing the form with the Application Dispatcher and producing output. This section explains that there is some variability in how the web form is presented to the user.

You may have noticed, when testing your SAS/IntrNet application, that the URL of the output page contains what seems to be, at first glance, an unreadable string of letters, numbers and special characters. On closer inspection, you will find that the URL contains all of the information that was passed from the form to the Dispatcher, including the form action, the hidden and query input fields, and the value of the macro variable. Moreover, the string is easy to understand and an essential part of this part of the lesson. The motivation for introducing this alternative is thus twofold: to gain familiarity with another method for submitting information to the Dispatcher, and to better illustrate how the information is presented to the Dispatcher and how you can make quick adjustments to submitted forms.

THE "OLD" WAY: FORM
In the first section, you were instructed that the following HTML form code is necessary for processing an Application Dispatcher program:

```
<FORM NAME="form"
ACTION="http://statweb.unc.edu/cgi-
bin/broker8">
<INPUT TYPE="hidden" NAME="_service"
VALUE="default">
<INPUT TYPE="hidden" NAME="_program"
VALUE="rarose.proj2.sas">
```

The information contained in this form and the input tags can be alternatively included in a link (pictured below).

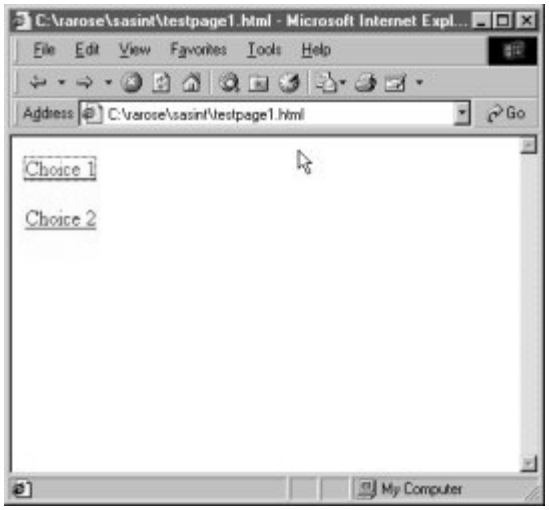
THE "NEW" WAY: LINK
For a link, all of the content, including variable content, must be included in the link as well. Instead of giving your users radio buttons to choose from, give them links.

It is very important that you understand that the links store all of the information required for processing the program: the form tag, the two hidden input tags (_service and _program), and the options available to the user - the macro variables that will determine how the data is processed.

Example (this should be an unbroken line of code with no carriage returns or spaces):

```
<a href="http://statweb.unc.edu/
cgi-bin/broker8?_service=default
&_program=rarose.proj2.sas&vara=1">
Choice 1</a>
```

The illustration below shows two links, reproducing the form from the first section of this tutorial, using the link method.



Let's study each component of this link, from left to right:

<a href= This is an HTML anchor (link) tag and reference attribute. This is essential for linking static pages or dynamic applications.

http://statweb.unc.edu/cgi-bin/broker8 This is the same as the value of the ACTION attribute of the FORM tag. It sends the request to the Application Dispatcher. Note that it is followed by a question mark: the "?" indicates that what follows it is a query string. The query string is composed of all input fields, including required input fields, and the query criteria input fields.

_service=default This is equivalent to the hidden tag named "_service". Note that it is followed by an ampersand (see below).

_program=rarose.proj2.sas This is similar to the hidden tag named "_program". This processes the program proj2.sas, which is located in my SASCODE directory (which has the libname rarose). Note that it is followed by an ampersand (see below).

vara=1> This constitutes what was formerly submitted by using a radio button. The variable "vara" is the macro variable whose value (in this case, "1") will be used to query a data set to produce an analysis subset. The > closes the tag.

Choice 1 This is the click-able text that the user will click to make the selection. It appears on the page in the web browser (see the illustration).

**** This closes the anchor tag.

Note that there is an ampersand character after each query value. In a query string (the part of the URL following the question mark) variables are separated by the ampersand character. The only variable without an ampersand behind it is the last one. If you've ever used a search engine, you may have noticed in the address bar at the top of the browser that your search terms follow a "?" and are separated by ampersands. This is the same principle.

This should illustrate how easy it is to reference a dynamic application from nearly anywhere on the web. The disadvantage to this is that if you have more than one choice for the user to make, you'll have to program a separate link for every *set* of choices. For instance, if your original form has a selection box with 3 selections, and a set of 3 radio buttons, you will need 9 links: one for each pair of selections. This makes it unreasonable for loads of more than a handful of choices.

A good example of a dynamic web site that uses multiple and overlapping methods of submitting information to the Dispatcher can be found at http://ssw.unc.edu/workfirst/demos/.

SUMMARY

The following might be thought of as a checklist.

BEFOREHAND (Do this only once)
- Obtain the URL of the dispatcher in your organization
- Set up a "sascode" directory and assign it a permanent libname
- Set permissions on the sascode directory for the Application Dispatcher
- Set permissions on public_html directory for global read access

WEB FORM
- Create a web form with the appropriate action and hidden input fields
- Add to the form a way for your users to select query criteria and submit it
- Put the form in the appropriate place on the server (the public_html directory)

DATA SET
- Create or obtain a data set and store it in either your sascode or public_html directories
- You should already be familiar with methods of manipulating data sets

SAS PROGRAM
- Retrieve the query criteria from the form
- Manipulate the data appropriately
- Output the content-type declaration to _webout
- Output analysis to _webout after the content type declaration

- Put the program in the appropriate place on your server
- Test, debug, retest, etc.

CONCLUSION

This presentation provides experienced SAS programmers with the foundation for creating basic dynamic SAS/IntrNet applications. The focus was on the requirements for creating a form and a program sufficient for performing a web-based analysis, placing aside most optional enhancements. The requirements were presented as components which could be copied directly from this tutorial and, with only a few modifications, applied to a working analysis. The method by which the Application Dispatcher processes a user request was reduced to the essential components that the programmer is required to know.

REFERENCES

The following references represent selected material from the SAS Institute web site and are useful supplements to this tutorial:

- HTML and Forms - http://www.sas.com/rnd/web/intrnet/dispatch/refhtml.html
- SAS/IntrNet Application Dispatcher 8.2 - http://www.sas.com/rnd/web/intrnet/dispatch/index.html
- SAS/IntrNet Application Dispatcher 2 - http://www.sas.com/rnd/web/intrnet/dispatch20/index.html
- The Output Delivery System in Dynamic SAS/IntrNet - http://www.sas.com/rnd/web/intrnet/dispatch/ods.html

The companion web site to this tutorial is available at

- http://ssw.unc.edu/workfirst/sasint/
- http://www.unc.edu/~rarose/sasint/

CONTACT INFORMATOIN

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

Roderick A. Rose
Jordan Institute for Families
CB 3550
301 Pittsboro St.
Chapel Hill, NC 27599
919-962-8826
rarose@email.unc.edu

The author gratefully acknowledges the assistance and support of Dennis K. Orthner, Ph.D., Kimberly Flair, Sally Muller, Rosemary Hallberg, Dean Duncan, Ph.D., and John Painter, Ph.D.