

Building a Customized Data Entry System with SAS/IntrNet™

Keith J. Brown
University of North Carolina – General Administration
Chapel Hill, NC

Introduction

The spread of the World Wide Web and access to the Internet has had a tremendous impact on the distribution of information. SAS Institute has aided this distribution by adding such features as %DS2HTM, %OUT2HTM, and %TAB2HTM to allow users to display information from their programs in a web-browser-friendly fashion. Still, we have been constrained at times in this process by a lack of data to feed into the programs that produce the web-friendly output.

While the capacity to manipulate data easily has always been a vital feature of the SAS System, this ability requires that data come from somewhere – an external file, a SAS dataset, in-stream following a CARDS statement, or some other source. While the information might be gathered from a wide variety of sources, some level of expertise was required to turn it into data that could be used, manipulated, and analyzed. Passing the information through someone with that expertise has often proved to be the bottleneck in the entire process. By using a relatively new component of the SAS System – SAS/IntrNet™ – and one of the oldest and most basic parts of SAS – the PUT statement – we can turn everyone on the Internet into our data supplier.

The Application Dispatcher

The first part of the SAS/IntrNet package we need to examine is the Application Dispatcher, which acts as an interface between your web browser and the SAS System. This interface is written using the Common Gateway Interface (CGI), and allows you to execute a program without having the program, the data, or even the SAS software installed on your machine. To use the Application Dispatcher across the web, you complete a form on a web page by selecting items and filling in fields. This form normally will be identical to those handled by other web applications, except that it invokes a SAS program rather than a script written in some language. The form can use any valid method of data entry – text input (up to 200 characters in Version 6.12),

radio buttons, check boxes, or selection bars. When you submit the form, the Dispatcher passes the information to the CGI program and on to a waiting SAS session. The specified SAS program executes, using the parameters you selected via the form on the web page. If the program you have executed produces viewable output, such as a table, report, graph, or map, this output returns through the CGI to display on your screen.

In its most basic form, the Application Dispatcher asks the user for one or two pieces of information, and uses those pieces of information to select the data needed to build the appropriate output. The output is normally passed through one of the formatting macros, which generates the HTML tags needed to display the output on the user's browser. By selecting the name of a school, for example, a user can generate a report for that particular institution from existing data. If the report is in the form of a PROC TABULATE report, the %TAB2HTM macro will create the tags needed to turn the output into an HTML table. It is this interplay between a form on a web page and a SAS program that generates HTML tags that makes it possible to do customized data entry via SAS/IntrNet. If a program can generate the HTML tags needed to display data, then a program can also generate the tags needed to collect data. The process revolves around one of the oldest and most useful SAS statements, the PUT statement.

Collecting the Initial Data

The initial web page collects the most basic information needed to generate the next web form, and then passes that information on to the program that will build the next form. The HTML tags for the initial page might look something like this:

```
<html>
<head>
<title>SESUG '99 Survey Form</title></head>
<body bgcolor="#ffffff">
<h1 align=center><font color="#0000ff">SESUG '99 Survey Form</font>
<br><font color="#ff0000">Welcome to Mobile!</font></h1>
<form method="post" action="http://localhost/scripts/broker.exe">
<input type=hidden name="_service" value="default">
<input type=hidden name="_program" value="kjb.sesug99a.sas">
<input type=hidden name="yr1" value="1999">
<pre>
<br>First Name:      <input type=text name="name1" size=40>
<br>Middle Initial(s): <input type=text name="name2" size=5>
<br>Last Name:        <input type=text name="name3" size=40>
                        Other HTML tags
<br><center>
<input type=submit value="Submit"> <input type=reset value="Reset">
</center></form></body></html>
```

If this form were being processed by a data handling mechanism other than SAS/IntrNet, such as a Perl script, the only change would be in the value of the "_program" parameter. When passing information from a form on a web page, you can use one of two methods – GET or POST. The GET method appends each parameter being passed to the URL; once the parameter names and values reach the maximum URL length for your browser, no more information can be passed to the waiting program. While this method can be used for forms passing a small amount of data, it's better to use the POST method, which does not have any inherent limit on the size of the parameters it can pass. Since POST does not show the parameters as it is passing them, respondents cannot change them by typing over them on the URL line, providing some small measure of security.

The First Program

When you press the SUBMIT button, all the parameters you have defined will be passed to the appropriate SAS program (SESUG99A.SAS in this case). These parameters are introduced into the program as macro variables; all the parameters from a form are treated as if they were variables in a single observation in a SAS dataset.

Since your program accepts input from a wide variety of users, you need to ensure that this input is treated carefully by the SAS macro language; one way to do this is by using the %SUPERQ macro function. This function masks all items that may require quoting at macro execution, and is recommended for macro variables that contain user-supplied text. The beginning of the program SESUG99A.SAS should contain lines similar to the following for each parameter being passed into the program:

<pre>%global name1; %let name1=%superq(name1); run;</pre>

For forms that pass large numbers of parameters, you may want to put these statements into a separate file, which can be brought into programs via an %INCLUDE statement.

To build a web form for data entry, you need to write the HTML tags needed to define various methods of inputting data back to the user's web browser; in SAS/IntrNet, the browser is accessed by the special fileref _WEBOUT. Your form may use any valid form of data entry over the web -- text entry, radio buttons, checkboxes, or even hidden

input. The last is especially valuable when you need to pass along information entered on a previous form or derived from within the program.

Customizing this form is simply a matter of writing out the appropriate tags and text based on data values. As SAS programmers, one of the first and most valuable data-handling techniques we learn is the use of the IF-THEN-ELSE statements to deal with conditional values. Using this technique, either alone or in conjunction with macro language, makes building a data entry form no different from producing a printed report using a series of SAS PUT statements.

In SESUG99A.SAS, we take the information gleaned from the initial web page and build a SAS dataset from it by using the RESOLVE function to convert macro variables into dataset variables. We then enrich that dataset by merging it with an existing SAS dataset, using formats to expand the information in it, or by any other method you would use in SAS programming. In this particular example, we create one observation in our initial dataset for each type of activity the respondent checked, and then expand our dataset by combining it with a list of local attractions that match our respondent's interests. The resulting dataset will have one observation for each attraction that matches the respondent's list of interests; if the respondent doesn't like any type of attraction, we will write out a single observation.

```
data one;
drop other1;
length name1 name2 name3 title other1 category $40 gender $1 state $2;
name1=trim(resolve("&name1"));
name2=trim(resolve("&name2"));
name3=trim(resolve("&name3"));
title=trim(resolve("&title"));
other1=trim(resolve("&other1"));
if title="PB" then title="Exalted Grand High Poohbah";
  else if title="Other" and other1 ne "Other (specify)" then
title=other1;
gender=trim(resolve("&gender"));
state=trim(resolve("&state"));
years=&years;
yr1=&yr1;
category=" ";
if trim(resolve("&art"))="on" then do; category="Art"; output; end;
if trim(resolve("&sights"))="on" then do; category="Sightseeing";
  output; end;
if trim(resolve("&shop"))="on" then do; category="Shopping";
  output; end;
if trim(resolve("&history"))="on" then do; category="History";
  output; end;
if trim(resolve("&nature"))="on" then do; category="Nature";
  output; end;
if category=" " then output;
run;
```

```

proc sql;
create table one as
  select distinct *
  from work.one as one,
       work.mobile as mob
  where one.category=mob.category
  order by category, site;
run;

```

At this point, our dataset now contains all the information we need to build our customized form. We have the demographic information from the original web form and information we've added about the various attractions that might interest our respondent; our next task is build the form that would allow him or her to supply us with data. Here we turn to one of the most basic features of SAS – the PUT statement. If you've ever created a customized report with PUT statements, this step will seem very familiar.

```

data _null_;
set one end=last;
by category;
line=_n_;
file _webout;
if _n_=1 then do;
put @1 "<html><head><title>SESUG '99</title></head>" /
      '<body bgcolor="#ffffff"><h1 align=center>' /
      '<font color="#0000ff">' /
      "Welcome to Mobile and SESUG '99!!</font></h1>";
if compress(name1 || name2 || name3) ne '' then put
  '<h1 align=center><font color="#0000ff"><br>Hi there<br>' /
  title name1 name2 name3 '</font></h1>';
if "&years"="26" then
put @1 '<h1 align=center><font color="#ff0000">' /
      'Wow!! Over 25 years? Really?' /
      "<br>Are you sure you're not Jim Goodnight in disguise?" /
      '</font></h1>';
put @1 '<form method="post" ' /
      'action="http://localhost/scripts/broker.exe">' /
      '<input type=hidden name="_service" value="default">' /
      '<input type=hidden name="_program" value="kjb.sesug99b.sas">' /
      '<input type=hidden name="name1" value="' name1 +(-1) '">' /
      '<input type=hidden name="name2" value="' name2 +(-1) '">' /
      '<input type=hidden name="name3" value="' name3 +(-1) '">' /
      '<input type=hidden name="title" value="' title +(-1) '">' /
      '<input type=hidden name="gender" value="' gender +(-1) '">' /
      '<input type=hidden name="state" value="' state +(-1) '">' /
      '<input type=hidden name="years" value="' years +(-1) '">' /
      '<input type=hidden name="yr1" value="' yr1 +(-1) '">' /
      '<table border="1" align=center>' /
      '<tr><th colspan=3>' /
      'How interested are you in this attraction?</th>' /
      '<th rowspan=2>Attraction/Activity</th>' /
      '<th colspan=4>If you have already tried this attraction, ' /
      'how would you rate it?</th></tr>' /
      '<tr><th>Very<br>Interested</th><th>Interested</th>' /
      '<th>Not<br>Interested</th>' /

```

```
'<th>Excellent</th><th>Good</th><th>Fair</th><th>Poor</th></tr>'
;
end;
```

This step begins by pointing the upcoming output to the special fileref _WEBOUT, SAS/IntrNet's name for the user's web browser. As we process the first observation, we write out HTML tags to define the subsequent output as a form for the browser. At this point, we begin customizing the form, first by addressing the user by name, and optionally making a small attempt at humor if the respondent has been using SAS for over 25 years. Notice that this customization is done with just IF-THEN logic and PUT statements; while the content may be HTML, the process is exactly the same as that for writing out a report. Using HTML tags, we enclose all the existing information that we want to pass as hidden input in our form, and then define the column headings for our data collection.

The next section of code simply prints the category ("Art", "History", "Nature", "Shopping", or "Sightseeing") in red at the beginning of the list of appropriate attractions. Since this line is purely for the respondent's information, there is no type of input associated with it.

```
if first.category then put
  @1 '<tr><th colspan=8><h2 align=center><font color="#ff0000">' /
    category +(-1) '</font></h2></td></tr>';
```

For each attraction that might interest our respondent, we would like to collect some information; specifically, how interested is he or she in this attraction and, if the respondent has already visited it, how highly would he or she rate it. Since for this particular survey the choices among interest levels and ratings are mutually exclusive, we will use two sets of radio buttons for each observation. To stimulate interest or jog a memory, we will link the attraction to its web site (if available), so the respondent can quickly obtain a little extra information about the attraction.

```
put @1 '<tr>' /
  @1 '<td><input type=radio name="int' line z2. '" value="3"></td>' /
  @1 '<td><input type=radio name="int' line z2. '" value="2"></td>' /
  @1 '<td><input type=radio name="int' line z2. '" value="1"></td>' /
  @1 '<td>' ;
if webaddr1 ne '' then put
  @1 '<a href="' webaddr1 +(-1) '">' site '</a>' ;
else put site ;
put @1 '<input type=hidden name="site' line z2.
  '" value="' site +(-1) '">' /
```

```

        '<input type=hidden name="cat" line z2.
        '" value="' category +(-1) '"></td>' /
    @1 '<td><input type=radio name="rate" line z2. '" value="4"></td>' /
    @1 '<td><input type=radio name="rate" line z2. '" value="3"></td>' /
    @1 '<td><input type=radio name="rate" line z2. '" value="2"></td>' /
    @1 '<td><input type=radio name="rate" line z2. '" value="1"></td>' /
    @1 '</tr>';

```

When we've written out the last line of our data entry form, we need to do some web "housekeeping" by closing the table, the form, and the HTML page. Since the number of observations we're passing to the next SAS program can vary, we also need a way to tell that program how many observations we are sending. Since this is information that the user neither knows, needs to know, nor cares to know, we use one final bit of hidden input to send the maximum number of observations we have processed.

```

if last then put
    @1 '</table><input type=hidden name="max" value="' line z2. '">' /
    @1 '<br><center>' /
    @1 '<input type=submit value="Submit">' /
    @1 '<input type=reset value="Reset"></center>' /
    '</form></body></html>';
run;

```

The Final Program

The beginning of SESUG99B.SAS is similar to that of SESUG99A.SAS; in both programs, we convert the parameters being passed from the previous web page into macro variables, and then convert those into dataset variables. In addition to the demographic variables that we passed as hidden input, we also need to read convert the macro variables for the attraction about which we surveyed the respondent, the type of attraction, the respondent's interest in it, and his or her rating of the site. The code shown below accomplishes these tasks.

```

%global name1; %let name1=%superq(name1); run;
        Similar statements for each parameter being passed
run;

data one;
length name1 name2 name3 title category $40
       site $50 gender $1 state $2;
name1=trim("&name1");
name2=trim("&name2");

```

```

name3=trim("&name3");
title=trim("&title");
gender=trim("&gender");
state=trim("&state");
years=input(compress("&years"),4.);
yr1=input(compress("&yr1"),2.);

do line=1 to &max;
  k=put(line,z2.);
  interest=input(resolve('&int' || k),1.);
  rating=input(resolve('&rate' || k),1.);
  category=trim(resolve('&cat' || k));
  site=trim(resolve('&site' || k));
  output;
end;
run;

```

At this point, all the information your respondent has entered on the previous web forms is contained in the temporary SAS dataset WORK.ONE. Exactly what you do with the data from this point on depends on what sort of data you've collected and how you need to use it. You should write the data out in some fashion, either as a permanent SAS dataset or a flat file. You should also give the respondent some indication that the data collection process is over by writing a message back to _WEBOUT, or by displaying some sort of results from the data collection, perhaps by using %TAB2HTM, %OUT2HTM, or %DSHTM.

Beyond those two tasks, if your form has subtotals and totals for rows or columns, you should check to make sure that the respondent's calculations were correct, and inform the respondent of any errors. If information entered on this form needs to be compared with data collected by other means, your program should perform those checks, and alert the user to any problems. In addition, you should perform any other checks or verification that you would perform if the data came from any other source.

Conclusion

While many users have discovered the utility of SAS/IntrNet in moving information to the World Wide Web, using that product to collect data has received less attention, even though the process can be quite simple. If you need to obtain the same information from each user, such as demographic data, your data collection system might consist of nothing more than a web form and a single SAS program that wrote that data out in a permanent fashion. Similarly, you might collect only the data that the SAS program needed to generate output; for instance, your web form might consist of little more than a selection box of school names. When the user highlighted a school

and submitted the form, the SAS program would display information about that school in a browser-friendly fashion, eliminating the need for static web pages for each institution.

When you need to customize the data that you are collecting, the process requires only a slight modification – the insertion of a SAS program between the initial web page and the final SAS program. This intervening program takes the information from the web form and uses it to build a more detailed form; data from this second form is then passed to the final program, processed, and written out.

As you might imagine, this data chain is not limited to the three links we have illustrated; it can easily be expanded as necessary. For instance, we have focused on only one of the pieces of information from the initial web page, the respondent's interests in local attractions. We could just as easily have chosen to collect data based on the user's location, years of SAS experience, gender, or preferred title. When the SUBMIT button was pressed on that web form, the respondent might have been presented with a selection of surveys on different topics, each of which might have generated more forms, and all stemming from that one single web page. The entire Internet is ready to supply you with data; all you need to do is ask for it.

Author Information

Keith J. Brown is an applications analyst in Chapel Hill, NC. He has used SAS as a student, teacher, programmer, and consultant since 1981. He can be reached at:

UNC-General Administration
Program Assessment & Public Service
P.O. Box 2688
Chapel Hill, NC 27515-2688
Telephone: (919) 962-1000
Fax: (919) 962-4316
Email: kjb@ga.unc.edu