Abstract

Web-based forms can be an substitute for paper forms used to collect data. For example, every time a contact was made through our consulting services at Cornell, a note was made of the person's affiliation, the type of contact, and the topic(s) discussed. Historically this information was recorded on paper and then tallied by hand at the end of each month, quarter and fiscal year: a painstaking, labor-intensive process begging for automation. This paper will illustrate how the WWW in conjunction with SAS became an elegant, useful and efficient alternative to the common tasks of collecting data and in the process streamlined reporting. The form-based Web pages and SAS programs designed for our consulting database will be used as an illustration.

Introduction

For those who embrace computer and communications technology, the emergence and development of Internet tools have been an exciting period for exploration, development and deployment of new applications for collecting and disseminating information. During the infancy of Web browsers, one of the greatest challenges facing application developers was how to connect the user-friendly graphical interface of Web browsers to the power of applications "behind the scenes." The Common Gateway Interface (CGI) became the drafting board for connectivity development.

What did the CGI drafting board look like before products such as SAS/IntrNet were developed? This paper will demonstrate how the power of the SAS System can be connected to the versatility and universality of the HTML form-based Web pages. It will demonstrate how to set up SAS programs so that they can be used with either a CGI written in Perl or with the new SAS/IntrNet product.

Project Background

When I first joined the Cornell Institute for Social and Economic Research a number of years ago, I was dismayed that some of our most important statistics were still gathered and reported by hand. Our Help Desk serviced numerous people throughout the campus and we needed a record of each and every contact with our members. Every time a member contacted us for assistance, an entry was made on a paper log to record the type of contact (e.g. phone, e-mail), affiliation of the member (e.g. student, faculty), college, and the nature of the question (e.g. SAS, UNIX). Each month, the log sheets were tallied by hand to determine how many contacts were made of each type and how many from each college and affiliation. In addition to the labor-intensity of this task, I was concerned about two issues.

Firstly, the logs were often incomplete, incorrect, or illegible, which increased the propensity for errors. Secondly, we weren't able to exploit the data to discover where we could enhance our services.

As a SAS programmer, I was eager to have the data in an electronic format so that I could use SAS to exploit this valuable information and generate reports easily on a regular basis. I needed to develop an application that would collect and analyze the data. The requirements of a successful application were:

- A simple to use, graphically-based interface that could be deployed easily on a multitude of platforms (PC, Macintosh and UNIX).
- A straightforward means for controlling the quality and integrity of the data being entered.
- A means for storing the data so that it would be available to SAS.
- Data-driven, standardized reports available on demand.
- A means for empowering users to generate the reports on demand.
- Broad availability of the application (e.g. on-campus and off-campus).
- Minimal investment for development and maintenance of the application.

The solution proved to be a Web-based SAS application. The project development fell into three categories: (1) development of HTML-based Web pages, (2) the development of macro-driven SAS programs for reading data and generating reports, and (3) the development of a Perl-driven CGI to connect the web pages to the SAS programs. The order of the three categories was never distinct – each provided input as to how the other elements should be structured. For example, it was the SAS programs and specific data needs therein that helped me to decide which HTML form input-types to use. For purposes of presentation, the above order seems to be the most illustrative of the process. When developing your own applications, bear in mind that there is considerable overlap between the categories.

The application discussed in this paper was developed to run on an IBM RS/6000 41T running AIX 4.1.5. SAS versions 6.1x were also installed on the same machine. The CGI scripts were written in Perl 5.0.

Development of Web Pages

Hyper-Text Markup Language (HTML) is a simple language which serves information from a Web server to a Web client (normally a browser). Although HTML is still fairly primitive and somewhat limiting, it does contain some features useful for data collection. Most of these
tools are familiar to Macintosh and Windows operating system users: radio buttons, checkboxes, pop-up boxes and text fields.

A web page can be designed using these tools so as to facilitate simple and standardized data entry. For example,

- use pop-up menus for a limited number of discrete choices
- use scroll boxes when numerous choices are offered
- use radio buttons for Yes/No type questions
- use checkboxes for On/Off type questions.

During the design phase of a Web-based data input and reporting project, many things need to be taken into consideration. If a suite of web pages is intended to replace and mimic paper forms, it may be advisable for the first edition of the Web pages to closely resemble the paper forms in appearance and structure for purposes of familiarity, although not at the expense of the added value of online functionality.

One great advantage of online data collection is the ability to reduce data entry errors. This can be achieved by:

- data validation techniques in JavaScript (client-side validation);
- providing pre-determined selections for open-ended questions directly in the HTML; and
- the use of data validation in a Common Gateway Interface (server-side validation).

It has been observed that the accuracy of data collection tends to increase the more closed-ended the survey instrument. To this end, the Web page developer should select data-input methods with pre-determined choices. For example, if a question can be answered in the affirmative or the negative, written forms may yield responses such as “Yes”, “yes”, “Y”, and “y” for the affirmative. To eliminate the need to recode these responses to a consistent response, the web page developer could make use of a radio button or a selection box which could pass either a one/zero or a Yes/No response respectively. Similarly, the number of non-responses to essential fields can be reduced by using JavaScript to ensure that certain fields are filled in before the data are sent to the Web server.

HTML forms are encapsulated by the <FORM ... > and </FORM> tags; all HTML between these tags is part of the online form. Input is achieved through the use of the various input types, which are marked by the <INPUT ... > tag, which specifies the type of data input, such as radio button, checkbox, single- or multiple-selection boxes.

Here is an example for asking a Yes/No Question, “Do you have previous SAS programming experience?”:

```html
<html>
<head>
<title>Form Example #1</title>
<h3>Form Example #1</h3>
<form action="/cgi-bin/cgi_name method=get">
<b>Do you have previous SAS programming experience?</b>
<input type=radio name=exp value=1> Yes
<input type=radio name=exp value=0> No
</form>
</html>
```

The <form> tag includes a reference to the CGI (Action=) and the method for sending the data from the Web client to the Web server (Method=). Data can be sent using one of two methods: POST or GET. (The GET method is the less secure method because it places the name-value pairs directly in the requested URL after the CGI reference. This can be extremely helpful during the development phases of a project, but the POST method is recommended for production.)

The responses to the question are contained in two <input> tags using the radio button type. Note that radio buttons with the same name are grouped together. The third <input> tag contains a special input type, “submit”. This creates a button which the user can click to send the input to the Web browser. The Web client will send a URL request to the Web browser (action= from the <form> tag) and the parameters from the form as name-value pairs (when METHOD=GET). If the user checks “Yes” in this example, the requested URL will be:

http://www.company.com/cgi-bin/cgi_name?exp=1

Forms can be as simple as the one in the previous example or as complex as the imagination can create. They can contain any combination of the input types, which allows a developer the power and flexibility to create customized applications.

**Developing SAS Programs for Web Applications**

Many of us are accustomed to using SAS for data management and reporting. It has the power and flexibility to allow us to create customized applications. As mentioned above, the driving force for this project was my desire to have our consulting data in a format that allowed me to use SAS to exploit it. Not only would it be able to save someone countless hours of tallying handwritten forms each month, but I would be able to ask questions about our services and use SAS tools to obtain the answers. For example:

- How do people contact us most often?
- What topics do we deal with most?
- Do we need to develop more expertise in certain areas?
- Are certain topics coming up often enough to justify a workshop?
- Are any of our consultants being utilized too much?
- Is a particular group using our services more than others?

I considered these to be my “standard” questions and I developed SAS programs to generate reports to answer them. The tools did not need to be sophisticated – a simple PROC FREQ could provide me with a wealth of information.
In the earliest phases of this project, I imagined what my “ideal” data set would be. I thought about what variables I would like to have and how I would like them to be measured. This in turn helped me to decide which HTML form elements to use to collect the data in the most usable format.

I then made a “dummy” data set, which contained made-up values of my ideal variables. I used this data set to test my SAS programs. When the SAS programs produced the reports I needed, I then set about the task of using the SAS Macro Language to transform my programs such that each report became a “module” or self-contained macro. Within each macro, I further transformed my SAS code so that it could accept parameters, such as which variable to use in a PROC FREQ. Lastly, I added a means for obtaining the external parameters and running, or not running, a particular macro accordingly. The external parameters in this case would be obtained from the user via the HTML-based web form.

For example, the “plain” SAS code to generate a frequency table of TOPIC by MONTH might look like this:

```sas
proc freq data=ssd.consult;
tables topic*month;
```

The same code driven by a macro might look like this:

```sas
%macro report1
proc freq data=ssd.consult;
tables topic*month;
%mend report1;
```

Perhaps MONTH is a variable that we'd like to specify when we decide to run the report. We might want to look at TOPIC*CONSULT instead. By implementing a macro variable, the module becomes even more flexible:

```sas
%macro report1 (byvar=);
proc freq data=ssd.consult;
tables topic*&byvar;
%mend report1;
```

Lastly, we may not always want to run this particular report, so we can ask in the HTML-form using an on/off radio button called “report1” which will have a value of 1 or 0 respectively.

Obtaining these values from the HTML form assumes that the CGI mechanism is able to pass them as system variables. The values can then be obtained using the SYMGET() function and stored as macro variables using %LET.

```sas
%LET report1 = symget(report1);
%LET byvar = symget(byvar);
```

IF/THEN logic can then be used to determine whether a report macro is to be run or not:

```sas
%IF &report1 = 1 %THEN %report1;
```

The last step in the process is finding a way to return the SAS output to the Web client (browser). The browser is expecting HTML-compliant output, so the SAS output essentially needs to be put in an HTML “envelope”. I have discovered two ways of achieving this:

1) Use a DATA _NULL_ at the beginning and end of the SAS program to place the expected HTML in the output; or
2) Use the SAS publishing macros provided on the SAS WWW site for (free!) download.

Both methods work very well and are described briefly below.

Enveloping SAS Output in HTML: DATA _NULL_

I decided upon a standard appearance for my Web pages for this application and hard-coded the desired HTML code into two SAS macros, %TOP and %BOTTOM. %TOP is called at the beginning of the SAS program and %BOTTOM is called at the end. A simplified example of what might appear in each macro is provided below. In this example, the output is written to a physical file whose name is specified in the CGI as a parameter called “OUTFL”. Upon the completion of the SAS job, the external file is then sent to the browser as part of the CGI.

```sas
* macro to put HTML at beginning of output file;
%macro TOP;
options nodate nonumber ls=70 mprint;
data _null_; 
file "&outfl";
put 'Content-type: text/HTML';
put ;
put '<HTML>';
put '<BODY bgcolor=#ffffff>';
put '<B>';
put '<PRE>';
run;
proc printto print="&outfl";
run;
%mend TOP;
```

```sas
* macro to put HTML at end of output file;
%macro BOTTOM;
/* close the filehandle to output file so we can output data _null_ below */
proc printto;
run;
data _null_; 
file "&outfl" mod;
put '</B>';
put '</PRE>';
run;
%mend BOTTOM;
```

Enveloping SAS Output in HTML: Publishing Macros
The SAS Institute provides SAS Publishing Tools\(^1\) which allow users with any standard browser to view SAS output as text, tables or graphic files. The tools are distributed via the SAS web page and take the form of macros for:
- HTML encoding of standard SAS output;
- creation of HTML tables from SAS data sets, and
- HTML formatting of complex tabular reports produced by PROC TABULATE;

and graphics drivers for GIF, GIF animation and JPEG graphics.

In the above example, the equivalent SAS Publishing Tool would be the HTML Output Formatter macro, %OUT2HTM.

```
option nodate nonumber ps=999;
%let report1 = symget(report1);
%let byvar = symget(byvar);
%out2htm (capture=on);
%macro report1 (byvar=);
  proc freq data=ssd.consult;
  tables topic*&byvar;
  %mend report1;
%if &report1 = 1 %then %report1;
%out2htm (htmlfile=outfile.html,
brtitle=Frequency Table,
tface=arial, tsize=6,
capture=off);
```

Macro-driven SAS programs like the ones presented above will not only work with the “traditional” CGIs, but can be used without modification if/when SAS/IntrNet is installed. The SAS/IntrNet CGI (“Broker”) performs the tasks of parsing the name-value pairs, setting environment variables using those values, and either launching a SAS session or sending the job to a SAS session listening on a certain port. The only modification necessary to use the SAS Broker as opposed to another CGI is the ACTION= tag in the HTML form, which should point to the location of the Broker executable\(^2\).

### Development of Common Gateway Interfaces (CGI)

Common Gateway Interfaces (CGI) provide the means for collecting data passed from a Web client (browser) via an HTML form to the Web server. CGIs are gateways to a number of other tools, such as programs or shell scripts which reside on the Web Server. CGIs themselves are executable programs or scripts designed specifically for the Web environment. They can be binary executables, shell scripts, Perl scripts or a number of other scripting tools. The SAS/IntrNet Broker is a compiled C executable.

A CGI can perform a number of tasks:
- data collection
- data validation
- data manipulation
- writing data to external files for use by other programs/scripts
- call to another program or script
- call to a URL
- HTML generation

For the application presented in this paper, I chose to use Perl\(^3\) as the language for developing my CGIs. Perl is short for Practical Extraction and Report Language and was developed by Larry Wall to give programmers a language for navigating among various files, scanning large amounts of text efficiently and printing easily-formatted reports. Perl is highly functional and can be viewed as a language that picks up where C programs and UNIX shell scripts leave off. It contains the functionality of both and complements where each is lacking. With the growth explosion of the Internet, Perl has become increasingly popular because it is straightforward and simple to learn. Many of the earliest CGIs were written in Perl.

The structure of a Perl CGI may look very familiar to a C programmer, since the language is structured similarly. As a SAS programmer and UNIX shell scripter, I found Perl to be easy to pick up.

Another advantage to using Perl for CGIs is that a lot of the difficult work has already been done. There are a number of Perl libraries readily available that contain subroutines for handling data arriving from an HTML form. I obtained a library\(^4\) that contains a routine called &ReadParse to obtain the name-value pairs from a form. This cut down the development time considerably.

The most basic CGI for obtaining data from an HTML form and launching a SAS program would resemble the following.

```
# Include CGI routines by Steven Brenner
require 'cgi-lib.pl';

#obtain name-value pairs
&ReadParse;

# set environment variable for each
# name-value pair
if (&ReadParse(*input)) {
  while ( ($name,$value) = each %input) {
    $ENV{$name} = "$value";
  }
}

# Point to work space
$WORK_DIR="/tmp2";
# Point to SAS System executables
$SAS="/usr/local/bin/sas";
```

---

\(^1\) See http://www.sas.com/md/web/products.html

\(^2\) Example: /cgi-bin/broker?_service=testserv&_program=sample.test.sas

\(^3\) Some excellent Perl references are Learning Perl and Programming Perl by Larry Wall and Randal L. Schwartz, published by O'Reilly & Associates, Inc.

\(^4\) The cgi-lib.pl routine is written by and copyrighted by Steven E. Brenner (S.E.Brenner@bioc.cam.ac.uk). You can find the latest copy of this library at: http://www.bio.cam.ac.uk/web/form.html
# Point to where SAS programs reside
$SASCODE="/tmp2";

# Use p$$ to get the unique process id number which is used to name output
$name="p$$";

# name of SAS program
$SASPROG = Frequent;
$PROG="$SASCODE/$SASPROG.sas";
# Other environment variables to be set
$MONTH = $ENV{fmonth};
$YEAR = $ENV{fyear};

# Launch SAS
system("SAS -fsdevice ascii.vt100
-sasuser /tmp $PROG
-log $WORK_DIR/$pname.log
-print $WORK_DIR/$pname.lst
-set outfl $WORK_DIR/$pname.out
-set year $YEAR -set month $MONTH ");

# Print the SAS output file
$output = `cat $WORK_DIR/$pname.out`;
print "$output";

# Clean up the temporary files
system ("rm $WORK_DIR/$pname.* ");

A DETAILED EXAMPLE

What follows is a simplified example from managing Help Desk data. The hope is that these programs will serve as a starting point for people embarking on similar projects.

Collecting Data

The first phase of this application is data collection. Only and HTML form and a CGI are required to collect the data, make some cursory checks of the data, and write the data to a flat file.

HTML

The HTML form uses a variety of input types to collect the data. Wherever possible, pre-determined choices are provided to ensure the highest possible level of accuracy.

<html>
<head>
<title>Enter Consulting Stats</title>
</head>
<body background="#ffffff" bgcolor="#ffffff">
<h3>Consulting Statistics Data Entry</h3>
<form action="/cgi-bin/consult.pl" method=post>

<! Pop-up selection box >
Month: <select name="month" size="1">
<option>Jan</option>
<option>Feb</option>
<option>Mar</option>
<option>Apr</option>
<option>May</option>
<option>Jun</option>
<option>Jul</option>
<option>Aug</option>
<option>Sep</option>
<option>Oct</option>
<option>Nov</option>
<option>Dec</option>
</select>

Year: <select name="year" size="1">
<option>96</option>
<option>97</option>
</select>
</form>
</body>
</html>
The purpose of the CGI is to parse the name-value pairs from the HTML form, fill in blank date fields, do some minor data checking, and then write the resulting data to a flat file.

```bash
#!/usr/local/bin/perl
# This script parses matched pairs from an HTML form and processes the data. The data are then stored in a file for later use. This file specifically collects consulting statistics.
# Written by Mary-Elizabeth Eddlestone
push(@INC,"/usr/local/etc/httpd/cgi-bin/"); require 'cgi-lib.pl';
&ReadParse;

# get the system data and use it to fill in # blank values $date = `date +%d%b%y`;
chop($date);
if($in{'day'} eq ""){
   $in{'day'} = `date +%d`;
   chop($in{'day'});
}
if ($in{'month'} eq "Months") {
   $in{'month'} = `date +%b`;
   chop($in{'month'});
}
if($in{'year'} eq "Yr") {
   $in{'year'} = `date +%y`;
   chop($in{'year'});
}
# convert month to uppercase $in{'month'} =~ tr/a-z/A-Z/;
# clean new lines from comments field $in{'comments'} =~ s/
//go;
open(DATA, " >> /tmp2/consult.data") || die "cannot open data file";

# write data to output file print DATA "$date:";
print DATA "$in{'day'}:";
print DATA "$in{'month'}:";
print DATA "$in{'year'}:";
print DATA "$in{'consultant'}:";
print DATA "$in{'netid'}:";
print DATA "$in{'affiliation'}:";
print DATA "$in{'college'}:";
print DATA "$in{'contact'}:";
print DATA "$in{'topic1'}:";
print DATA "$in{'comments'}:";
print DATA "\n";

# if more than one topic is discussed, create a second record with 2nd topic if ($in{'topic2'} ne "Topics" && $in{'topic2'} ne "None"){
print DATA "$date:";
print DATA "$in{'day'}:";
print DATA "$in{'month'}:";
print DATA "$in{'year'}:";
print DATA "$in{'consultant'}:";
print DATA "$in{'netid'}:";
print DATA "$in{'affiliation'}:";
print DATA "$in{'college'}:";
print DATA "$in{'contact'}:";
print DATA "$in{'topic2'}:";
print DATA "$in{'comments'}:";
print DATA "\n";
}
close(DATA);

#generate html print &PrintHeader;
print "<HTML><HEAD><TITLE>Submission Results</TITLE></HEAD>
";
print "<H2> Item successfully added to database.</H2>
";
print "<HR>"
print "</HTML>";

GENERATING REPORTS
Once the data are stored electronically, SAS can be used to analyze and summarize the Help Desk activities. All of the reports presented here are standard in that there are no provisions in this example for ad hoc reporting. The advantages of serving the reports via an HTML form are:
- the data are the most current;
- anyone can run the reports without needing to know SAS;
- the reports can be generated from any “thin client” – only a web browser needs to be installed on the machine; and
- a user can view the reports on any platform with a web browser that supports forms.

HTML
The web page presented below offers the user a selection of standard reports. To opt for any particular report, the user needs only to select the appropriate radio button and submit the form to the web server.

<HTML>
<HEAD>
<TITLE>Generate Usage Reports</TITLE>
</HEAD>

<body background="" bgcolor="#ffffff">

<h3>Generate Consulting Report</h3>
<form action="/cgi-bin/sasreport.pl" method="post">

<b>Select the Month and Year for Report:</b><p>

<b>Month:</b> <select name="month" size="1">
<optionopenhid=Months>Months</option>
<option value="Jan">Jan</option>
<option value="Feb">Feb</option>
<option value="Mar">Mar</option>
<option value="Apr">Apr</option>
<option value="May">May</option>
<option value="Jun">Jun</option>
<option value="Jul">Jul</option>
<option value="Aug">Aug</option>
<option value="Sep">Sep</option>
<option value="Oct">Oct</option>
<option value="Nov">Nov</option>
<option value="Dec">Dec</option>
</select>  

<b>Year:</b> <select name="year" size="1">
<optionopenhid=Yr>Yr</option>
<option value="96">96</option>
<option value="97">97</option>
</select> </p></form>

<p>Select desired tables: </p>
<table border=1>
<tr><td>Total Consultations By:</td>
<td><input type="radio" name="table1" value="1" checked> Yes <input type="radio" name="table1" value="0" checked> No</td> </tr>
<tr><td>Month * College</td>
<td><input type="radio" name="table2" value="1" checked> Yes <input type="radio" name="table2" value="0" checked> No</td> </tr>
<tr><td>Month * Topic</td>
<td><input type="radio" name="table3" value="1" checked> Yes <input type="radio" name="table3" value="0" checked> No</td> </tr>
<tr><td>Month * College * Topic</td>
<td><input type="radio" name="table4" value="1" checked> Yes <input type="radio" name="table4" value="0" checked> No</td> </tr>
<tr><td>Month * Consultant * Topic</td>
<td><input type="radio" name="table5" value="1" checked> Yes <input type="radio" name="table5" value="0" checked> No</td> </tr>
<tr><td>Month * Contact-Type * College</td>
<td><input type="radio" name="table6" value="1" checked> Yes <input type="radio" name="table6" value="0" checked> No</td> </tr>
<tr><td>Month * Contact-Type</td>
<td><input type="radio" name="table7" value="1" checked> Yes <input type="radio" name="table7" value="0" checked> No</td> </tr>
<tr><td>Affiliation * Contact-Type</td>
<td><input type="radio" name="table8" value="1" checked> Yes <input type="radio" name="table8" value="0" checked> No</td> </tr>
<tr><td>Affiliation * Contact-Type * College</td>
<td><input type="radio" name="table8" value="1" checked> Yes <input type="radio" name="table8" value="0" checked> No</td> </tr>
</table>

<input type="submit" value="Generate Report">
<input type="reset" value="Reset">  

</form>

</body>
</html>

---

### CGI

```perl
#!/usr/local/bin/perl

# This script is launched by an HTML form.
# It in turn launches a SAS program which
# returns a consulting statistics report as HTML
# Written by Mary-Elizabeth Eddlestone

push @INC, "/usr/local/etc/httpd/cgi-bin/";
require 'cgi-lib.pl';

# The cgi-lib.pl routine is written by and
# copyrighted by Steven E. Brenner
# http://www.bio.cam.ac.uk/web/form.html

# set environment variables for name-value pairs
if (&ReadParse(*input)) {
  while ( ($name,$value) = each %input) {
    $ENV{$name} = "$value";
  }
}

# Point to work space
$WORK_DIR="/tmp";
# Point to SAS System executables
$SAS="/usr/local/bin/sas";
# point to location of SAS program(s)
$SASCODE="/tmp2";

# Use p$$ to get unique PID number to name
# temporary output
$pname="p$$";

$SASPROG = Monthly;
$PROG="$SASCODE/$SASPROG.sas";
$MONTH = $ENV{month};
$YEAR = $ENV{year};
$table1 = $ENV{table1};
$table2 = $ENV{table2};
$table3 = $ENV{table3};
$table4 = $ENV{table4};
$table5 = $ENV{table5};
$table6 = $ENV{table6};
$table7 = $ENV{table7};
$table8 = $ENV{table8};

# Return a page if no Month or Year provided
# (JavaScript could be used in the HTML form
# instead)
if ($MONTH eq "Months" || $YEAR eq "Yr") {  
  print &PrintHeader;
  print "<HTML><HEAD><TITLE>Data Entry Error</TITLE></HEAD>

  <body bgcolor="#ffffff">

  <h2>Data Entry Error: You forgot to select a month or year for your report. Please return to the previous page and make your selection.</h2>

}<h2> Data Entry Error:

</body>
</HTML>
```
```
SAS

The SAS program obtains the name-value pairs from the environment variables set in the CGI when SAS is launched. The program creates macro variables for these values and determines which reports should be run. The output is enveloped in HTML by using the %TOP and %BOTTOM macros.

* SAS program to read consulting data and print out summaries;
* Mary-Elizabeth Eddlestone;

* This program is invoked via the WWW using a CGI called sassearch.pl;

* get environment variable set through HTML input;
%let outfl = %sysget(outfl);
%let month = %sysget(month);
%let year = %sysget(year);
%let table1 = %sysget(table1);
%let table2 = %sysget(table2);
%let table3 = %sysget(table3);
%let table4 = %sysget(table4);
%let table5 = %sysget(table5);
%let table6 = %sysget(table6);
%let table7 = %sysget(table7);
%let table8 = %sysget(table8);

options nodate nonumber ls=70 mprint;

**** macro definitions *******;

* macro to put HTML at beginning of output file;
%macro top;
data _null_;
file "&outfl";
  put 'Content-type:text/HTML';
  put;
  put '</HTML>';
%mend top;

* macro to put HTML at end of output file;
%macro bottom;
/* close the filehandle to outputfile so we can output data _null_ below */
data _null_; file "&outfl" mod;
  put '</B>';
  put '</PRE>';
  put '</BODY>';
  put '</HTML>';
%mend bottom;

***** table macros *******;
%macro table1;
* total consultations by month;
  proc freq data=&month;
    tables month/ nocol norow nopercent nocum;
    title "Total Consultations &month &year";
  %mend table1;

%macro table2;
* total consultations by month by college;
  proc sort data=&month;
    by topic college;
  proc freq data=&month;
    tables college / nocol norow nocum;
    title "Total Consultations by College for &month &year";
  %mend table2;

%macro table3;
* total consultations by month by topic;
  proc sort data=&month;
    by topic;
  proc freq data=&month;
    tables topic/ nocol norow nocum;
    title "Total Consultations by Topic for &month &year";
  %mend table3;

%macro table4;
* total consultations by month by college by topic;
  proc sort data=&month;
    by topic college;
  proc freq data=&month;
    tables college*topic/ nocol norow nocum;
    title "Total Consultations by College and Topic for &month &year";
  %mend table4;

%macro table5;
* total consultations by month by consultant by topic;
  proc sort data=&month;
    by cons topic;
  proc freq data=&month;
    tables consul*topic/ nocol norow nocum;
    title "Total Consultations by Consultant and Topic for &month &year";
  %mend table5;
CONCLUSION

Web-based forms can be an ideal substitute for paper forms used to collect data. When the flexibility, widespread availability and ease-of-use of web browsers can be combined with the power, flexibility, familiarity and reliability of the SAS System, the only boundaries become those of the imagination. By developing macro-controlled SAS programs, an application developer is able to use the web as a front end to the application. HTML forms can be used to obtain user input which is then sent to the SAS System via a Common Gateway Interface (CGI). CGIs need not be elaborate to bridge the channel between a thin client and the SAS System. Perl is often the language of choice for CGIs, particularly on UNIX platforms, because it is a simple yet powerful language providing all the necessary functionality. Output from the SAS System needs to be enveloped in HTML before being served to the browser. This is easily accomplished using either DATA step programming or by making use of the readily available SAS Publishing Tools. The SAS Publishing Tools can simplify the process.
greatly and can enliven web pages with color and graphics.

The author can be reached for comment at sasmee@unx.sas.com. Although the author is currently employed by the SAS Institute Inc., the application presented in this paper was developed while the author was an employee of Cornell University. All opinions expressed in this paper are those of the author from the perspective of a SAS user and are not necessarily those of the SAS Institute.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ©indicates USA registration.