Interfacing the SAS® System with the World Wide Web

Larry Hoyle
Institute for Public Policy and Business Research, Univ. of Kansas

Participants in this workshop will learn:

- how to read data across the Internet using the FTP and URL access methods,
- some HTML basics,
- how to write HTML from a data step,
- how to create World Wide Web (WWW) compatible bitmap graphics (GIF files) using the SAS system,
- and how to use the SAS system and CGI scripts to create WWW pages on demand.

We will also discuss other methods of using the SAS system to serve WWW information, including JDBC and htmSql.

The starting point for information on using SAS with the Internet is:

SAS Institute WebTools
http://www.sas.com/rnd/web/intro.html

Reading or writing Internet sources

FTP

The SAS system has access methods for reading and writing from FTP sites. To point to the file "readme" in the directory "/pub/ippbr/fdiv" at "ftp2.cc.ukans.edu" logging in as user "anonymous" with the password "guest" use the following filename statement:

```
filename rm ftp 'readme' cd='/pub/ippbr/fdiv'
        user='anonymous' pass='guest'
        host='ftp2.cc.ukans.edu';
```

The ftp file can be read or written as its access permissions permit.

URL

The URL access method can be used to read WWW pages. Suppose you need current weather for the ocean just off San Diego. The National Weather Service in Tallahassee, Florida maintains a page with current weather buoy data from the National Data Buoy Center. Buoy number 46045 is off the coast from San Diego. To access data from it, use the following SAS statement (note the ":80").

```
filename buoy URL
  'http://www.met.fsu.edu:80/nws/cgi-bin/buoy.cgi?46045';
```

The SAS code below reads the page and strips out the recent observations.

```
data b;
infile buoy;
keep dt tempair winddir windsp pressure
     tempsea waveht waveprd;
length slash dec $ 1;
input @3 slash char$1. @10 dec char$1.@;
if (slash eq '/' and dec eq '.') then do;
    input @1 utcday 2.
      @4 utchour 2.
      @7 tempair 5.
      @24 winddir 3.
      @29 windsp 3.
      @46 pressure 6.
      @58 tempsea 5.
      @64 waveht 4.
      @69 waveprd 3. ;

   curmonth=month(date());
   curyear=year(date());
   dt=mdy(curmonth,utcday,curyear)*24*3600
     +utchour*3600;
   format dt datetime12. ;
   output;
end;
run;
```

Some HTML basics

Here are a few places to start to learn to create World Wide Web (WWW) pages. There are, of course many other sources - both on the Web and in print.

A Beginner's Guide to HTML
http://www.ncsa.uiuc.edu/General/
Internet/WWW/HTMLPrimer.html
A Beginner's Guide to URLs
http://www.ie.cuhk.hk/mirror/
url-primer.html
An Instantaneous Introduction to CGI Scripts and HTML Forms
http://www.cc.ukans.edu/info/forms/
forms-intro.html

WWW pages are written using *Hypertext Markup Language* (HTML). HTML files are simply text with embedded formatting codes called *tags*. The tags for a minimal HTML file are shown below.

```
<html>
<head>
<title>A minimal HTML file </title>
</head>
<body>
<h1>An optional heading</h1>
This is a paragraph.
It will be justified
until the following tag.<p>
This is the second paragraph.
</body>
</html>
```

Depending on the browser, this file would display as:

TITLE:A minimal HTML file

An optional heading

This is a paragraph. It will be justified until the following tag.

This is the second paragraph.

Some tags like “<title>“ are paired with the ending tag containing a “/”, like “</title>“. Other tags may not need a corresponding ending tag - like the paragraph tag “<p>“, or the line break “
“.

One useful, although not elegant, pair of tags is the “<pre>...</pre>“ construct. All text between these two tags will not be justified. These tags are useful for wrapping tabular material which cannot be recoded with HTML tags such as the output of a SAS Proc.

SAS Institute is developing several HTML formatting tools, including an HTML Output Formatter, an HTML Data set Formatter, and an HTML Tabulate Formatter.

For information on these new tools see:

SAS Institute Web Tools - Html Formatting Tools
http://www.sas.com/rnd/web/htmlgen.html

Writing HTML from a SAS program
The example below outputs HTML tags from DATA steps and wraps the output of a Proc in “<pre>“ tags. SAS macros could make this less tedious.

```
filename myhtml
'c:\ddrive\sugi\sugi22\handson\means.htm';
options ls=64 nodate pageno=1;

data _null_;
file myhtml ;
put '<html>';
put '<head>';
put '<title>HTML from SAS</title>';
put '</head>';
put '<body>';
put '<pre>';
                /* redirect PROC output to the HTML file */
 proc printto print=myhtml;
title 'Means from the sasuser.class dataset';

proc means data=sasuser.class mean;
var weight height;
class sex;

 proc printto;

data _null_;
file myhtml mod;
put '</pre>';
put '</body>';
put '</html>';
run;
```

When viewed in a browser, the output would look like the table at the bottom of this page. The SAS Institute Web Tools page points to a number of tools for making nicer looking HTML output. Some of these tools are from SAS Institute and some from SAS users.

```
Means from the sasuser.class dataset            1

      SEX  N Obs  Variable  Label                      Mean
      ----------------------------------------------------
      F        9  WEIGHT    Weight in pounds    90.1111111
                  HEIGHT    Height in inches    60.5888889

      M       10  WEIGHT    Weight in pounds   108.9500000
                  HEIGHT    Height in inches    63.9100000
      ----------------------------------------------------
```

HTML tables
The HTML for a simple table might look like:

```
<table border>
<tr>
<th>my table<td align=right>col 2<td align=right>col 3
<tr>
<th>row heading<td align=right>1.1<td align=right>3.3
<tr>
<th>another row<td align=right>4.4<td align=right>5.5
</table>
```

and render as:

my table	col 2	col 3
row heading	1.1	3.3
another row	4.4	5.5

The “<tr> tag starts a row. The “<th>“ and :<td> tags start cells.

HTML forms
HTML has the capability of defining a form which will be used by the browser to enter data. The form tag has an *action* parameter which specifies the WWW address of the *Common Gateway Interface* (CGI) script which will process the form. The form tag also has a *method* parameter which specifies the protocol by which the CGI script expects the data. *POST* is the preferred method in most cases.

The HTML below sets up a form which has a scrolling select box showing the words “one” and “two”. The former is selected by default. The form also has an input box with the default value of 3 entered. (See the 8th page of this paper for a picture of a form displayed in a browser).

```
<FORM ACTION=“http://foo.bar/foo.pl” METHOD=“POST”>
<SELECT NAME=“x” MULTIPLE>
  <OPTION VALUE=“1” SELECTED>one
  <OPTION VALUE=“2”>two
</SELECT>
<INPUT TYPE="HIDDEN" NAME="P" VALUE="hello">
<INPUT TYPE=“TEXT” NAME=“y” VALUE=“3”>
<INPUT TYPE=“SUBMIT” VALUE=“send x,y to foo”>
</FORM>
```

When “two” is chosen from the select box the browser will prepare “x=2” to be sent to the CGI script. Entering a 4 in the text box will prepare the message “y=4”. The parameter “p=hello” will always be sent but not appear on the screen since it appears in a hidden field.

Hidden fields are useful for specifying values unique to the form like which SAS program the CGI script should run. They can also be used to save the state of a sequence of interactions - as in a sequence of drill down selections.

The stream of parameter data is sent to the CGI script when the button created by the submit field is selected.

Creating a GIF or JPEG file from SAS

The SAS system has a graphics driver “IMGGIF” which will write GIF format files, and a driver IMGJPEG which creates JPEG files. PROC GDEVICE can be used to create copies of these drivers which produce files of different sizes. The code below will create a new driver named “gif320” which will produce images at 320 by 240 pixels.

```
libname gdevice0 "c:\apps\sas\gdevice0";
goptions reset=all;
proc gdevice noprompt nofs
    c=gdevice0.devices;
  copy imggif from=sashelp.devices
    newname=gif320;
  modify gif320 xpixels=320 ypixels=240
    xmax=3.2in ymax=2.4in;
run;
```

The sample below produces a GIF file of the first GTESTIT screen.

```
filename gifout 'handson\test.gif';
goptions device= gif320
        gsfmode=replace
        gsfname=gifout ;
proc gtestit pic=1;
run;
```

If this GIF file was named “test.gif” and located in the directory “mypics” in the http server directory of the machine “fake.foo”, the HTML tag:
<IMG SRC=“http://fake.foo/mypics/test.gif”>
would cause it to be displayed inside the WWW page.

Animated GIF

SAS 6.12 includes a driver -GIFANIM - for creating animated GIF files. Documentation and examples can be found at:

SAS Institute Web Tools -- GIFANIM Device Driver
http://www.sas.com/rnd/web/GifAnim.html

Accessing SAS through CGI scripts

The figure to the right shows the sequence of events involved when a user opens an HTML form which uses the SAS system as a CGI server using a user-written intermediary program.

There are four programs shown in the figure:

- The browser (e.g. Netscape's) running on the client system.
- The http server program or daemon (also known as the Web server program) running continuously on the server.
- The CGI script, an intermediary program launched on the server system by the http server program. This is typically a UNIX shell script, a C program, or a Perl script.
- The SAS program.

These four programs are duplicated in the figure to simplify the drawing of the links among them. There is only one instance of each for a given transaction.

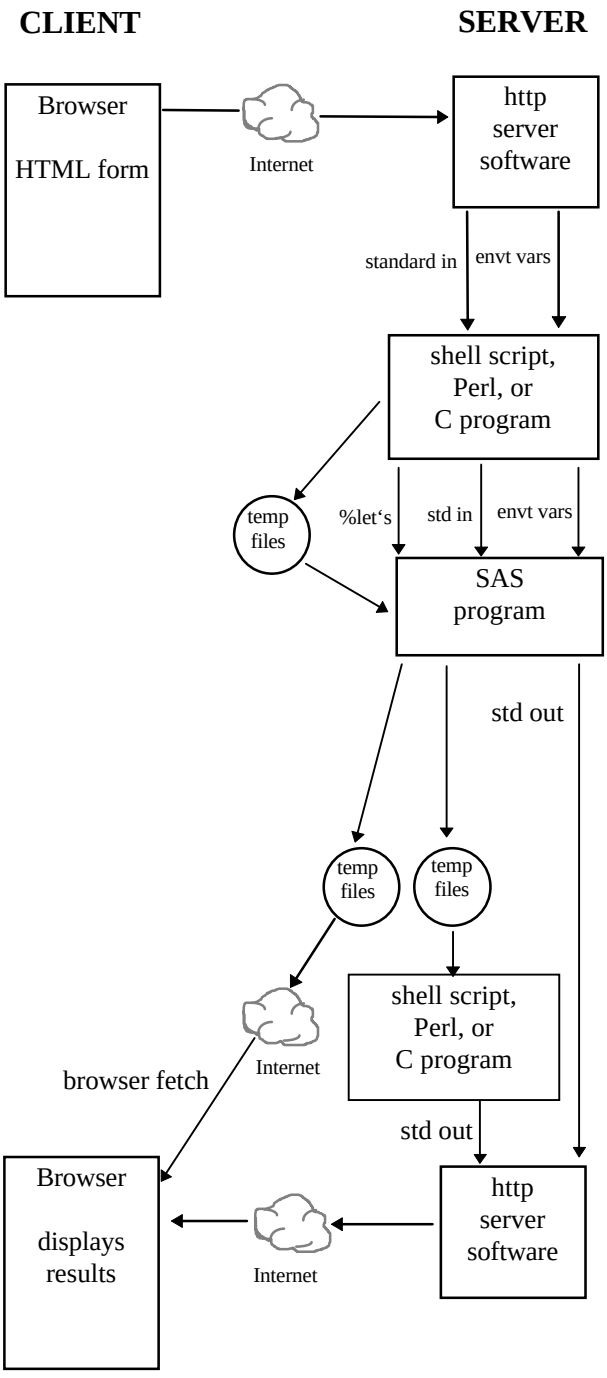
The browser initiates the interaction by sending the parameters selected in a form through the http server to the CGI script. The CGI script receives these parameters via a combination of environment variables and the *standard input file* (stdin).

The CGI script launches a SAS program. The script can send information to the SAS program through a number of mechanisms. It can:

- Pass on the standard input file (except in the Windows environment).
- Define environment variables.
- Write a sequence of *%let* statements at the beginning of the SAS program.
- Write a named file.
- Open a socket to a running SAS Program (not discussed here).
- Communicate via OLE with a running SAS program in the Windows environment (not discussed here).
- Send a SQL query to a SAS/SHARE*NET™ server using the SAS SQL library for C (not discussed here).

The SAS program can, in turn, write a named file or write to *standard output.* The latter is passed on by the http server program to the browser across the Internet. The former can be passed on by the CGI script program, with or without further processing.

Finally, the browser can fetch a file directly from the server if it has been passed a reference to it in the HTML sent back through the http server program.



Additional information about using SAS with CGI scripts is available on the SAS Institute page at:

Running the SAS System From a CGI Script
http://www.sas.com/rnd/web/sascgi.html

CGI script to SAS and back - an example

Minimal.pl is an example of a Perl script used as an intermediary between an http server and a SAS program. It has several design constraints:

- brevity,
- ease of modification,
- compatibility with UNIX and Windows,
- integrating the HTML form with the script which serves it,
- compatibility with the CGI component of the SAS/IntrNet product (more on this later).

Only the first and last sections (part 1 and part 4) of minimal.pl need to be modified to adapt it to a different form

The first section of minimal.pl defines variables which point to the files and directories needed by the program. The variables which need definitions are:

- *$tempdir* - the directory containing the temporary files used by minimal.pl
- *$formtitle* - the title used in the form that minimal.pl returns when there are no input data
- *$scriptloc* - the URL (web address) of minimal.pl
- *$sasprogloc* - the location of the SAS program file to include in the SAS program to be run by minimal.pl
- *$sasexe* - the location of the SAS system executable file
- *$sasconfig* - the location of config.sas
- *$cgilib_loc* - the location of the program cgi-lib.pl

Minimal.pl calls freeware subroutines from the package cgi-lib.pl by Steven E. Brenner. This package has routines which parse the input parameters from the http server, split apart multiple selection parameters, send back error messages to the browser and so on. The last line of the part 1 listing links in that package.

minimal.pl - (part 1)

```
#!/usr/local/bin/perl
#
# minimal cgi program Larry Hoyle - December 1996
#

srand();
$rnum = int(rand(99999));

#-----------------------------------------------------------------
#     The variables in this box point to necessary files, etc.
#     Set them to the appropriate values.
#-------------------------------------------------------------------------#
                                # temp directory                        #
$tempdir = "c:\\p158\\";                                                #
                                                                        #
                                # Title of returned HTML                #
$formtitle = "CGI - SAS";                                               #
                                                                        #
                                # This script                           #
$scriptloc = "http://lhoyle.ippbr.ukans.edu/cgi/minimal.pl";            #
                                                                        #
                                # The SAS program to run                #
$sasprogloc = "c:\\fnord\\cgi\\minimal.sas";                            #
                                                                        #
                                # Location of SAS itself                #
$sasexe = 'X:\\APPS\\SAS612\\SAS\\SAS.EXE';                             #
                                                                        #
                                # Location of config.sas                #
$sasconfig = 'c:\\apps\\sas612\\CONFIG.SAS';                            #
                                                                        #
                                #location of cgi-lib.pl                 #
$cgilib_loc = "/apps/perl5/lib/cgi-lib.pl";                             #
#-------------------------------------------------------------------------#

                        # don't change these
$sasfile = "$tempdir"."cgi$rnum.sas";
$saslog = "$tempdir".".cgi$rnum.log";
$saslist = "$tempdir"."cgi$rnum.lst";
$webout = "$tempdir"."cgi$rnum.web";

        # uses cgi-lib.pl to parse the form parameters
        # cgi-lib.pl Copyright (c) 1996 Steven E. Brenner
        # For more information, see:
        #     http://www.bio.cam.ac.uk/cgi-lib/

require ($cgilib_loc);
```

The next two sections of minimal.pl should not need to be changed to handle different forms. Part 2 uses *&ReadParse* from cgi-lib.pl to parse input from the http server and put it into an associative array called *%in*. If there is no input *&ReadParse* returns *false* and subroutine *&sendform* is called to send back the HTML form.

minimal.pl (part 2)

```
# ---------------------------------------------------------------------
#               Try to parse input, if none - send a form
# ---------------------------------------------------------------------
                                # HtmlTop is from cgi-lib.pl
$BeginHtml = &HtmlTop($formtitle);

if (!&ReadParse) {          # ReadParse is from cgi-lib.pl
   &sendform;
   exit;
}
```

Part 3 uses the associative array *%in* which was created by &ReadParse to write a sequence of SAS %let statements defining the parameter data as macro variables. It also writes a filename statement defining *_webout.* It then %includes the external SAS program and runs SAS on the combined file. When the SAS program is done *_webout* is sent back to the browser and temporary files deleted.

sample contents of $sasfile

```
%let greet0 =2;
%let greet=Howdy;
%let greet1=Howdy;
%let greet2=Bonjour;
%let number0 =1;
%let number=2;
%let to0 =1;
%let to=Folks;
filename _webout "c:\p158\cgi19674.web";
%include "c:\fnord\cgi\minimal.sas";
```

minimal.pl (part 3)

```
# ------------------------------------------------------------------------------------
#               Write the SAS program file and forward its output.
#               Set up %let statements consistent with the CGI component
#                of SAS/IntrNet. Define filename _webout.
#               The file pointed to by $sasfile will get the complete SAS program,
#                the preface written by this program and the external SAS program.
# ------------------------------------------------------------------------------------
if(open (SASPROGFILE, "> $sasfile")){
   foreach $varname (sort keys(%in)){
      @valarray = &SplitParam($in{$varname});          # SplitParam is from cgi-lib.pl, makes an array of the multiples
      $valarraylen = @valarray;
                                                                  # Every PARAM gets PARAM0
      print SASPROGFILE ("%let $varname."0 =$valarraylen;  \n");
                                                                  # define PARAM as macro var.
      print SASPROGFILE ("%let $varname=$valarray[0];  \n");
                                                                  # multiple selects handled here
      if($valarraylen>1){
         for($ix=0; $ix<$valarraylen; $ix++){                      # define multiple selects
            $ixplus = $ix + 1;                                    # as PARAMi for i=1 to howmany
            print SASPROGFILE ("%let $varname$ixplus=$valarray[$ix];  \n");
         }
      }
   }
                                                                  # define filename _webout
   print SASPROGFILE "filename _webout \"$webout\";\n";
                                                                  # include in the SAS file
   print SASPROGFILE "%include \"$sasprogloc\";\n";
   close SASPROGFILE;
   }else {
   cgidie("could not find SAS file: $sasfile");
}
                                                                  # run the SAS program
system("$sasexe  -CONFIG $sasconfig -sysin $sasfile -log $saslog -print $saslist");
                                                                  # forward the SAS program's output
if(open (SASOUTFILE, "$webout")){
   binmode(SASOUTFILE);                    # In case there are graphics, explicitly specify binary mode for WIN32
   $nread = read(SASOUTFILE,$chunk,1024);
   while ($chunk ne ""){
      print ($chunk);
      $nread = read(SASOUTFILE,$chunk,5000);
   }
   close SASOUTFILE;
} else {
   cgidie("ERROR: unable to open the SAS output file");
}
unlink ("$saslog");                                               #clean up temp files
unlink ("$sasfile");
unlink ("$webout");
exit;
```

If minimal.pl is invoked with no parameter data as input, it will send back an HTML form using the subroutine &sendform. The listing in part 4 shows that subroutine.

The HTML form in this section will need to be modified by the user. The SAS program which processes it will also need to be changed.

The &sendform routine first sends back a response header which is ***Content-type:text/html\n\n*** ("\n" is a line ending), and should not be changed.

All of the HTML for the form document is found between the markers ***<<HTML_FORM*** and ***HTML_FORM*** and is typed just as if it were in a file by itself with the exception that the action parameter of the form tag contains *$scriptloc*. This is a Perl variable name which will be replaced by the string entered for it at the top of the program. It contains the URL of minimal.pl.

The form in this example defines 3 parameters:

- greet - in the first select box can take on multiple values.
- to - in the second select box can only have a single value.
- number - in a text box allows a user to enter any value

Having the form as part of the CGI script program has potential advantages besides housekeeping ones. The script could run a SAS program or use the SAS SQL Library for C to send an SQL query to a SAS/SHARE*NET server. Data from this external procedure could be used to generate a form. A modified version of this Perl script could also handle data validation by sending back a form with values from the previous submission and error messages.

Building a form from a query to a SAS program could keep it current automatically. For example a SELECT box could be populated with a current list of parts from a database or the current years in a file.

Multiple occurrences

The trickiest part of handling form parameters in SAS is what to do with multiple occurrences of parameters like *greet*. If the user, for example, selects "Kansan"[1] and "French", the Perl script will receive greet=Howdy&greet=Bonjour.

[1] Minimal.sas and the form with it are really a "Hello World" application. Translated into Kansan that's "Howdy Folks".

The CGI component of the SAS/IntrNet product would handle the preceding multiple occurrences by defining the following sequence.

```
%let greet=Howdy;
%let greet0=2;
%let greet1=Howdy;
%let greet2=Bonjour;
```

The macro variable corresponding to the parameter name gets the first value received. Then a "0" is appended to the parameter name and it is assigned the number of multiple occurrences. Finally each occurrence appears as PARAM*j* where *j* is a sequential counter starting with 1. Minimal.pl adopts this approach to multiple occurrences.

minimal.pl (part 4)

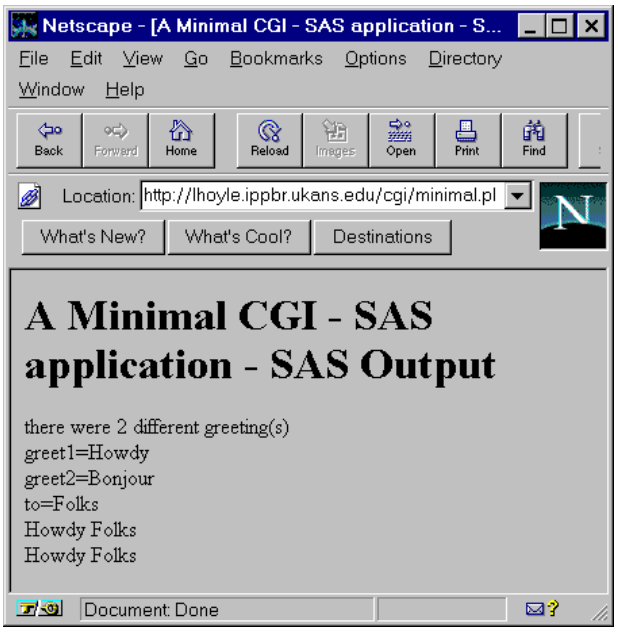
```
# ------------------------------------------------------------------------------
# Send back a form
#  Customize this section for your application.
#  Form variable names should be short enough to allow appending
#  at least one digit - more if a multiple select variable.
#  The variable names with appended digits must meet SAS naming
#  rules for macro variable names.
# ------------------------------------------------------------------------------
sub sendform {
print "Content-type:text/html\n\n";

print ($BeginHtml);
print <<HTML_FORM;
<HR>
<FORM ACTION="$scriptloc" method="POST">
Send a greeting
<BR><SELECT NAME="greet" multiple >
      <OPTION VALUE="Hello">Formal
      <OPTION VALUE="Hi">Informal
      <OPTION VALUE="Howdy" SELECTED>Kansan
      <OPTION VALUE="Bonjour" >French
      </SELECT>
<BR>To the planet:
<BR><SELECT NAME="to" >
      <OPTION VALUE="World">World
      <OPTION VALUE="Earth">Earth
      <OPTION VALUE="Folks" SELECTED>Folks
      <OPTION VALUE="Monde" >Monde
      </SELECT>
<br>How many greetings do you want?
<br> <input type="text" name="number" value="2">
<HR>
<INPUT TYPE="submit" VALUE="Send Greeting">
</FORM>
</BODY>
</HTML>
HTML_FORM

}  # end sendform
exit;
```

The included SAS Program - minimal.sas
Minimal.sas is the SAS program which is included into the SAS program submitted by minimal.pl. The first thing it must write to *_webout* is a header like:
put 'Content-type:text/html'/;
The trailing "/" is important. If the header is incorrect the browser will not be able to display the return page.

Minimalg.sas at the lower right returns a GIF file. It first sends the header (Note the two line feeds - 0A):
'Content-type:image/gif' '0A0A'x



Macros
This SAS program contains a macro *%macro mini* which is necessary to process the multiple occurrences of the variable *greet*. If the form has no multiple occurrence variables, a macro may not be necessary.

```
/*minimal.sas  L. Hoyle December 1996 */
/*  a sample SAS program to be used with minimal.pl */
/*  the Perl program will append this file to a sequence of */
/*  %let statements which define each form variable as a macro
variable */
options MPRINT;

%macro mini;
data _null_;
file _webout;

put 'Content-type:text/html'/;
put '<HTML>';
put '<HEAD>';
put '<TITLE>';
put 'A Minimal CGI - SAS application - SAS Output';
put '</TITLE>';
put '</HEAD>';
put '<BODY>';
put '<H1>';
put 'SAS Output';
put '</H1>';

put "there were &greet0 different greeting(s)";
%IF &greet0 eq  1 %THEN
  %DO;                /* only one greeting selected */
     put "<br>greet=&greet";
  %END;
%ELSE                 /* multiple greetings */
  %DO  i=1 %TO &greet0;
       put "<br>greet&i=&&greet&i";
  %END;
put "<br>to=&to";
%DO n=1 %TO &number;
   put "<br>&greet &to";
%END;

put '</BODY>';
put '</HTML>';

run;
%mend mini;
%mini
```

```
/* minimalg.sas returns a GIF file */
goptions device=gif260  gsfmode=append
          gsfname=_webout
          gprolog = 'Content-type:image/gif' '0A0A'x;

proc gtestit pic=1;
run;
```

The CGI component of the SAS/IntrNet product
As this paper is going to press there is a new product from SAS Institute undergoing alpha testing. The CGI component of the SAS/IntrNet product will simplify the construction of WWW form to SAS program links.

The biggest difference comes in the introduction of the SAS application server. With the CGI method used by minimal.pl, each SAS program was run as a separate session. This can involve a noticeable startup time in some environments - although it may not be any problem in other environments.

With the application server, each SAS program may run within the one server. This avoids startup time. It also could protect your server from a sudden startup of hundreds or thousands of SAS jobs if you get listed in “What’s cool”. On the other hand if there are not enough application servers running, client applications might have to wait under heavy loads.

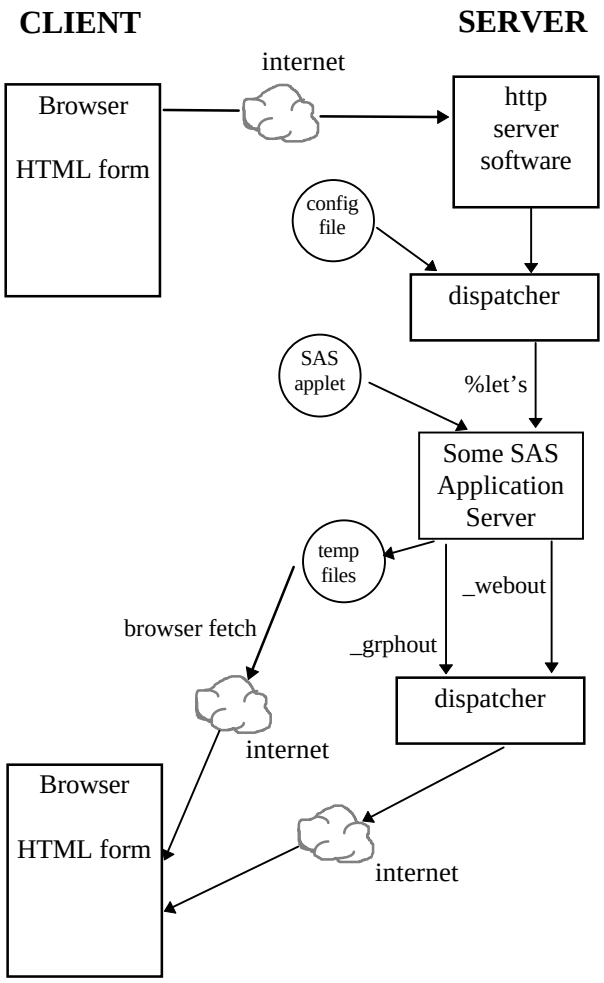
To write applications which use the CGI component of the SAS/IntrNet product you would write an HTML form with two new tags using the special variables *_program* and *_service*. The first declares the SAS program to be run:
<INPUT TYPE="HIDDEN" NAME="_program" VALUE="sample.hello.sas">
The second declares which application server it is to run on. These servers might be on different machines. The SAS Institute written CGI program, the *dispatcher*, handles all communication between the browser and the application server. The following would give a choice between the servers “pickle” and “relish”:
<SELECT NAME="_service">
<OPTION VALUE="pickle" SELECTED> Default
<OPTION VALUE="relish" SELECTED> Default
</SELECT>

The SAS applet would look a lot like minimal.sas. It would write HTML output to _webout and graphics output to _grphout. It would have macro variables defined the same as did minimal.sas, including the treatment of multiple selections with the exception that minimal.pl will always define a PARAM0 for PARAM. With the new product, it may not be defined if only a single selection is made.

The following is a sample SAS applet which corresponds to minimalg.sas on the preceding page.

```
goptions gsfname=_grphout gsfmode=replace dev=gif733
        gprolog='Content-type: image/gif' '0A0A'x ;

proc gtestit pic=1;
  run;
```



In general the CGI component of the SAS/IntrNet product simplifies the creation of SAS-CGI applications. It will also offer very useful debugging and security features

htmSQL
Another new component of the SAS/IntrNet product currently available in beta test form, is htmSQL. htmSQL uses CGI as its communication method but is quite different from the other CGI tools discussed here so far.

With htmSQL an author embeds special tags in an HTML document which describe an SQL query (or update in future versions). The SAS Institute supplied CGI server program interprets those tags and submits the SQL query to a SAS/SHARE*NET server. It then formats the results of the query as described by the special tags and sends back the HTML with the embedded results of the query.

No SAS program is launched so queries can be quick. Results are, however, limited to those which can be generated by an SQL query.
The two pairs of principal tags for htmSQL are:
{sql}...{/sql} and
{eachrow}...{/eachrow}
The {sql} tag describes the sql query. The {eachrow} tag describes the layout of each row of the result of the query.

Suppose the weather data on the first page of this paper were available to a SHARE*NET server in the data set weather.buoys. The following would retrieve an HTML table of the air and sea temperatures.

```
<table>
<tr><th>46045<td>air><td>sea>
{query datasrc="buoys"}
{sql}
select utchour, utcday, tempair, tempsea
  from weather.buoys
  where buoy = 46045
{/sql}
{eachrow}
<tr>
<th>{&utcday}/{&utchour}<td>{&tempair}<td>{&tempsea}
{/eachrow}
{/query}
</table>
```

Further information on htmSQL can be found at:

SAS Institute Web Tools - htmSQL
http://www.sas.com/rnd/web/htmsql.html

JDBC
Another important method for accessing SAS through the Internet is Java Database Connectivity (JDBC). JDBC allows a Java applet embedded in a WWW page to submit SQL queries to a SAS/SHARE*NET server.

More information about JDBC appears in the paper *Choosing a Method for Connecting Java to the SAS® System Across the Internet - CGI, JDBC or Socket?*

Acknowledgments
Thanks to the SAS Institute Web Tools Development Group for information on new components of the SAS/IntrNet product (and for the components too).

Resources and References
The best source for information on using SAS with the Internet are the SAS Institute World Wide Web pages. The home page is:

SAS Institute Inc., *SAS Institute Web Tools*. http://www.sas.com/rnd/web/intro.html.

Other references:

Friendly, Michael, *Online Statistics* http://www.math.yorku.ca/SCS/Online/

Hoyle, Larry, *Examples of Connecting SAS to WWW* http://www.ukans.edu/cwis/units/IPPBR/ksdata/ksdata.htm#ecsw

Hoyle, Larry. "Choosing a Method for Connecting Java to the SAS® System Across the Internet - CGI, JDBC or Socket?" *Proceedings of the Twenty-Second Annual SAS Users Group International Conference.*, San Diego, March 1997.

Hoyle, Larry, "SAS Software and the WWW - What Next?", *Proceedings of the Twenty-First Annual SAS Users Group International Conference.,* Chicago, March 1996.

Hoyle, Larry, *More on using SAS with WWW* MidWest SAS Users Group Conference, Cleveland, October 1995.

Hoyle Larry, *Connecting SAS to the World Wide Web - Forms Across the Internet* MidWest SAS Users Group Conference (MWSUG94) Omaha, September 1994

Sun Microsystems Inc*.,* JavaSoft Home Page http://java.sun.com/

Sun Microsystems Inc*., The JDBC(tm) Database Access API* http://splash.javasoft.com/jdbc

Larry Hoyle
IPPBR, University of Kansas
607 Blake Hall
Lawrence, KS, 66045-2960
l-hoyle@ukans.edu
http://www.ukans.edu/cwis/units/IPPBR