Running the SAS® System on the Web

Michael Friendly*
York University

Abstract

At the most basic level the WWW provides the means to share SAS datasets, programs, macros, etc. using a more convenient interface than was provided for these functions previously by other internet protocols (FTP, email, listserves, discussion groups). And, starting with SAS 6.11, SAS itself can access remote datasets and files using the WWW-based URL method provided by the filename statement. But what possibilities exist for connecting SAS more directly to the web, and how can these possibilities be explored?

This paper describes some of my experiments and experiences which explore possible forms of interaction between SAS and the World Wide Web, including

- writing custom CGI scripts for statistical and graphic applications
- a general SAS–web CGI gateway
- serving SAS programs intelligently to remote users

1 Introduction

Several years ago I began to convert SAS-related course materials for several courses ([1], [2]) from a mainframe to a Novell-based PC lab. Course documents were converted to HTML and linked to SAS files on the server.

Although this had many beneficial effects, it created two disconnected environments for the students' work: they could read course materials and the linked SAS files with Netscape, but had to use cut/paste or find the files on the local filesystem to run them with SAS. So, it made some sense to look for ways to connect SAS programs more dynamically to web documents.

Server-side vs. Client-side SAS processing

The client-server model provides for processing to be split cooperatively between a local machine and a remote host; ideally, each machine is delegated those parts of the task it can provide most readily. SAS processing can be carried out either on the client's local machine (using either the SAS-supplied Plugin for Netscape or MSIE, or by defining SAS as a "helper App" for your browser), or on the server by use of CGI scripts. Likewise, data resources and SAS programs can be located anywhere on the network. The trick is to connect them together so "it just works".

*Author's address: Psychology Dept., York University, Toronto, ON, Canada M3J 1P3. email: `friendly@yorku.ca` home page: `www.math.yorku.ca/SCS/friendly.html`

2 Server-side: Running SAS from a CGI script

Web browsers can be allowed to make use of the computing power, programs, or data provided by a server by setting up a CGI script on the server. The CGI script is simply a program running on the server which receives a request from the web server, runs a SAS process to meet that request, and returns the results to the web server. The web server, in turn, passes the results back to the browser like any other HTML information.

In the simplest case, the CGI script is a Unix shell script (or other executable program) which passes the request to the SAS program via standard input,

```
#!/bin/sh
sas -stdio < myWebApp.sas > myWebApp.lst
cat myWebApp.lst
```

This method, first illustrated by Larry Hoyle ([3],[4]) requires very little in the way of Shell programming, but places the burden of parsing the peculiar format in which the request is sent by the browser on the SAS program.

Perl CGI scripts

Over the last year I wrote several custom CGI programs in Perl. Perl is particularly attractive for CGI scripts because of its powerful parsing and pattern-matching expressions, strong security features, and because a large library of object-oriented modules for WWW applications is available with the standard distribution or on the CPAN ftp sites (e.g., `ftp://ftp.funet.fi/pub/languages/perl/CPAN/`). I particularly recommend Lincoln Stein's `CGI.pm` (version 2.3 or later) which handles most of the details of CGI processing.

WebPower

Determining the sample size for a factor or effect in an ANOVA design is usually difficult because of the need to specify *all* of the treatment means in order to calculate the non-centrality parameter of the F-distribution, on which power depends.

The WebPower form (`http://www.math.yorku.ca/SCS/Online/power/`) runs a SAS program that calculates power or sample size needed to attain a given power for one effect in a factorial ANOVA design. The program is based on specifying the range of treatment means, and calculating the minimum power, or maximum required sample size.

The form invokes the perl CGI script, `power.pl`, which parses the request parameters, writes a temporary SAS program, runs that program, and returns the listing file to the web browser.

The SAS program itself simply calls a SAS macro, `fpower.sas`, with the user's input values substituted for '`$`' variables `$a`, `$b`, `$delta`, `$alpha`, and `$out`:

```
options nodate nocenter nonumber;
title 'Power analysis for ANOVA designs';
%include "$power";
%fpower(a=$a, b=$b, delta=%str($delta),
    alpha=$alpha, $out);
```

`power.pl` uses one of the standard Perl CGI utility packages to parse the request parameters passed to it from the form. It does some rudimentary error checking to make sure that the request parameters are valid, and checks that the SAS program ran correctly, producing a listing file with the results. The output is returned as pre-formatted text (wrapped in `<PRE>...</PRE>` tags), but it would be easy enough to use a SAS macro to return the output in the form of an HTML table.

Although the SAS program is quite simple and `power.pl` uses the Perl CGI package, error trapping and formatting the query parameters for insertion in the program template did take some additional effort.

Sieve Diagrams

A second web application (`http://www.math.yorku.ca/SCS/Online/sieve`) provides a web interface to a SAS/IML program to draw sieve diagrams for data in a two-way contingency table. The CGI script, `sieve.pl`, was designed to:

- print the initial HTML form,
- retrieve the input parameters (row and column labels and the frequency table),
- write a SAS program, interpolating the user's data into a template,
- run the program, producing either a PostScript or GIF file,
- return the image to the browser, together with links to the SAS program and the output listing file.

The program template again looks quite simple (although a fair bit of effort in the script was devoted to massaging the input from the form to be syntactically correct in SAS/IML):

```
%let gsasfile= %sysget(GSASFILE);
%let title   = %sysget(TITLE);
%let data    = %sysget(DATA);
filename gsasfile  "&gsasfile";
goptions device=pscolor
   gsfname=gsasfile gsfmode=replace;
title "&title";
proc iml;
   %include "$sieve";
   f = { $data };
   vnames = { $var };
   lnames = { $lab };
   title = '$title' ;
   font = '$font';
   run sieve(f, vnames, lnames, title );
   quit;
```

Note again that all the '`$`' names are Perl variables whose values are substituted when the SAS file is written. However, some parameters in the program (`GSASFILE`, `TITLE`, and `DATA`) are passed in the environment, then retrieved to macro variables using the `%sysget()` function.

Writing—and debugging—`sieve.pl`, however, convinced me that there ought to be a better way, so I wrote a general-purpose program to connect *any* SAS program to a web form.

3 `sascgi`: a SAS-WWW gateway

`sascgi` is a Perl CGI script designed to provide a gateway between a web server and a SAS program which returns results to the web browser. My intention was to provide a relatively uncluttered, general protocol for running SAS on the web, so that the SAS application can be made as simple as possible, and a single script could serve a wide variety of applications. The script handles most of the interaction with the web server, making it much easier to write SAS applications to be run on the web.

It works like this. You have a SAS application you want to make accessible to users on the web. That application requires some input from the user, to select records or variables to be processed, or to set parameters for some computation. You write an HTML form in which the user can enter the required information. When the user presses the SUBMIT button, the browser calls `sascgi`, passing the parameters defined in the form; `sascgi` retrieves the parameters, runs SAS, and returns the results to the user.

The script passes input parameters to the SAS program via the environment, which is much easier than trying to parse stdin in SAS. The SAS program is assumed to retrieve these parameters via `%sysget(PARAM)`, or `sysget('PARAM')` in a data step. The SAS program can return results to the browser by one of three methods:

- Printing to standard output (STDOUT) from a DATA step or via PROC PRINTO
- Printing to a listing file
- Producing one or more graphic output files in PostScript or GIF format.

The SAS program can communicate success or failure (with an error message) by writing a message to a `.err` file or by returning a message starting with 'ERROR:' to STDOUT.

The script is used in an HTML document by embedding a `<FORM>` block in the document, referencing this script as the ACTION attribute, as follows:

```
<form method="POST"
 action="http://your.server/cgi-bin/sascgi">
 <input type="hidden" name="SASFILE"
        value="getlist.sas">
 <input type="hidden" name="TITLE"
        value="Page title">
 <input type="hidden" name="OUTPUT_METHOD"
        value="STDOUT">
 <input type="hidden" name="REQUIRE"
        value="ITEMS LISTS">
   ... other form elements ...
</form>
```

The `<INPUT>` tags define names of parameters which are passed to the SAS program. Use `type="hidden"` when you don't want the user to select or change the value; otherwise, you can use *any* form element which generates a value.

Parameters

All query parameters defined in the form are processed by `sascgi`; however, only those parameters which are given a value in the form (by default, or entered by the user) are placed in the SAS environment. Five form parameters have a special meaning to `sascgi`:

SASFILE The value defines the name of the SAS program to be run.

TITLE The value is used as a title for the output.

OUTPUT_METHOD The value tells `sascgi` how the SAS program intends to return results. Possible values are:

- `STDOUT` assumes the SAS program produces output which can be sent directly to the browser, e.g., as wrapped with HTML tags like `<PRE>...</PRE>` or `<TABLE>...</TABLE>`.
- `LST` to return a listing file. The output is wrapped in `<PRE>...</PRE>` by `sascgi`, formfeeds are translated to a horizontal rule (`<HR>`), and the result is sent to the browser.
- `IMAGE/type` returns an inline image, output as `<img src="$sasfile.type">`. The image type defaults to 'gif'. `sascgi` expects the SAS program to produce an image file in the curent directory named by the value of environment variable `GSASFILE`.

REQUIRE Specifies a list of the names of form parameters which *must* have a value in the input from the form in order for the SAS program to be run. It is generally preferable to provide a default value in the form (e.g., `<input type="text" name="ITEMS" value="10">`) or to design the SAS program to provide a default.

DEBUG If non-zero, turns on verbose output for debugging.

Writing your SAS application

Your SAS application should be written to

1. retrieve any required parameters from the environment, using the `%sysget()` macro function or the `sysget()` datastep function.
2. produce results which are to be returned to the browser according to the `OUTPUT_METHOD` specified in the form.

   - `OUTPUT_METHOD = LST` is the simplest. Just use whatever SAS facilities produce printed output to the listing file.
   - For `OUTPUT_METHOD = STDOUT`, use a `data _null_` step, specifying `file STDOUT`, e.g.,

     ```
     data _null_;
        file STDOUT;
        put "<pre>";
          ... (output results) ...
        put "</pre>";
     ```

     My `%htmltab` macro can also be used to produce output to `STDOUT` containing either a preformatted printout or an HTML table from a SAS dataset.
   - For `OUTPUT_METHOD = IMAGE`, obtain the name of the `GSASFILE` from the environment, and use a `GOPTIONS` statement with `GSFNAME=` referring to this fileref:

     ```
     *-- Retrieve output file name;
     %let gsasfile = %sysget(GSASFILE);
     filename gsasfile  "&gsasfile";

     *-- Device options;
     goptions device=pscolor
        gsfname=gsasfile gsfmode=replace;
     ```

%include files

If your application uses files, dataset, or macros, you can simplify your applications by defining an autocall library, or by appropriate `filename` statements in the `config.sas` file which is used by `sascgi`. For example, if your `config.sas` contains

```
filename macros
   ('~web/sasuser/macros',
   ('~web/sasuser/webmacros');
```

an application could use

```
%include macros(htmltab);
```

Alternatively, specify the full pathname to the file on the include statement.

Testing your application

You should be able to test your application from the command line by itself, by placing any required parameters in the environment:

```
>setenv LISTS 1
>setenv ITEMS 20
>sas getlist
```

sascgi is also designed so that it can be run from the command line for debugging, e.g.,

```
>setenv DEBUG 1
>./sascgi 'SASFILE=getlist.sas&LISTS=1&ITEMS=20'
```

The parameters OUTPUT_METHOD, SASFILE, TITLE, and DEBUG may all be passed to this script via the form or the environment.

Several working examples, along with source code and further explanation are available online at http://www.math.yorku.ca/SCS/Online/sascgi.

Enhancements

sascgi can be enhanced in several ways:

- The SAS program can be run on a separate server (which may have more data or processing resources). On a Unix system, this can be done just by invoking SAS through rsh $sas_server sas $sasfile instead of sas $sasfile.
- Configuration information (server name, program paths, etc.) is presently contained in sascgi itself, and must be modified for each server. Providing a CONFIG variable in the form would allow the configuration information to be read from a separate file, and would allow several configurations to be used at one site.
- Most web applications require only a relatively small number of user-specified parameters. sascgi does not at present provide the ability for the user to supply an input dataset to be analyzed. This can be done by making use of the file upload facilities in browsers (which are handled by the CGI.pm code), or by SAS itself, using the URL or FTP methods provided by the filename statement.

4 Server-based CGI: PHP/FI

An alternative to using custom CGI scripts, or a general-purpose script like sascgi is to embed script-handling capabilities into HTML documents in the manner of server-side includes or JavaScript. PHP/FI (http://www.vex.net/php/) is one public-domain package which can be compiled into the Apache httpd server as an extension module. When installed, all pages with the extension .phtml are automatically parsed for PHP commands.

This is more convenient (the script is part of the document), more efficient (no need to fork an additional CGI process for each use), and also more secure (security is controlled by the same access restrictions used by the server itself).

For example, here is a form which allows a user to enter bivariate data for a regression model:

```
<FORM METHOD="POST" ACTION="glm-sas.phtml">
<H2>Input or Paste  Y and X</H2>
Each line should contain a value of the
response, Y, followed by a value of X.
  <TEXTAREA NAME="yx" ROWS=10 COLS=20>
  </TEXTAREA>
<BR>
What Type of Regression ?
  <SELECT NAME="type">
    <OPTION VALUE="1" SELECTED>Linear
    <OPTION VALUE="2">Quadratic
    <OPTION VALUE="3">Cubic
  </SELECT>
  <INPUT TYPE=SUBMIT VALUE="Submit"><BR>
</FORM>
```

The form ACTION file, glm-sas.phtml is like an ordinary html file, except that it contains embedded PHP code, bracketed by <? ... > delimiters. The lines below process the regression type (created in PHP as the variable $type) value.

```
<HTML><HEAD>
<TITLE>SAS Regression Calculator</TITLE>
</HEAD><BODY>
<H1>SAS Regression Calculator</H1>
<?
  switch (intval($type));
  case 1;
    $model = "X";
    Echo "<H2>Linear Model</H2>"; break;
  case 2;
    $model = "X X*X";
    Echo "<H2>Quadratic Model</H2>"; break;
  case 2;
    $model = "X X*X X*X*X";
    Echo "<H2>Cubic Model</H2>"; break;
  default;
    $model = "X";  break;
  endswitch;
>
```

You can define functions in PHP to process or transform inputs. Here is one which creates a SAS datastep from the input entered into the YX field of the form:

```
<? Function Make_DataStep $dataset, $yx (
    $text = "data " + $dataset + ";\n";
    $text = $text + "  input y x;\n"  ;
    $text = $text + "  cards;\n"  ;
    $text = $text + $yx + ";\n"  ;
    Return ($text);
  );
>
```

Finally, these lines build the SAS program as a string, add lines to call PROC GLM, and echo the program as the standard input to SAS. The output listing file is then sent back to the browser.

```
<?
  $sas_program = Make_Datastep ("web", $x, $y)
    + "proc glm data=web;\n"
    + "   model y = " + $model + ";\n";

  $sas = "/usr/local/bin/sas";
  $execarg = "echo " + $sas_program +
     " | " $sas + "-stdio  > glm.lst"
  Exec ($execarg);
```

```
  $result = File("glm.lst");
  $n = Count ($result);
  $i = 2;
  Echo "<pre>";
  while ($i < $n) {
     Echo "%s<br>" $result[$i];
     $i++;
    }
   Echo "</pre>";
>
</BODY>
</HTML>
```

This tight integration of CGI capabilities with HTML should make it much easier to develop powerful web-based applications. All of the standard capabilities for which people write separate CGI scripts (form mail, page counters, browser-sensitive responses, etc) are built right in, and the script language is rich enough to support interesting forms of client-server interaction. PHP also provides file upload capability and integrates a variety of database and SQL packages (mSQL, Postgress95, Sybase), and the GD library for GIF creation. It would not be difficult to add support for SAS datasets or for direct communication with the SAS System.

5 Client-side SAS processing

Server-side SAS processing seems most useful and/or appropriate when: (a) the result to be returned to the client is a single file such as a table or a graph; (b) the result is based on datasets which must remain with the server; (c) the server can handle a possibly large number of processes being run.

However, when one or more of these conditions is not true, it may be more useful for web-based interaction to run SAS on the client machine.

SAS Plugin

The recently released SAS Plugin for Windows95 (`www.sas.com/rnd/web/plugin.html`) allows an HTML file to embed a reference to a remote SAS program, data set, SAS catalog, or output file. The HTML `<EMBED>` tag is used to reference the URL, in a form similar to the `<IMG>` tag for a sound or graphic:

```
<EMBED SRC="http://www.server/path/file.sas"
    ACTION=OPEN>
```

When a page containing an `<EMBED>` tag is loaded, the browser downloads a temporary copy of the file and draws an action button. When you click the action button, the plugin sends the action to the SAS System. For a SAS program, the Open action simply displays the program in the Program Editor window. Other actions (Browse, Print, Query, Submit) are defined for different types of SAS files.

However, note that the Submit action is inherently dangerous: it runs the program without the opportunity for the user to intervene.

Server setup for client-side SAS processing

Even without a plugin, servers and browsers can be configured so that SAS program files launch SAS as an external “viewer”. *This is not a good idea*, but I’ll first describe how it works.

At the server-side, `.sas` files are normally sent as the MIME type `text/plain`, which means they are displayed as-is in the browser window. To allow those files to be run locally by SAS, the server needs to send them as the MIME type `application`, with a subtype such as `x-sas` (”x” designates an experimental MIME type). With the NCSA and Apache httpd servers, this is accomplished by adding the following line to the server’s `.srm.conf` configuration file:

```
AddType application/x-sas .sas
```

For the browser to be able to launch SAS as a viewer, you have to set up your web browser to recognise SAS files which are served as `application/x-sas`. The details vary considerably between Netscape, Mosaic, MSIE and other browsers; instructions for setting up some browsers are detailed in `http://www.math.yorku.ca/SCS/Online/sasweb/`.

Once this has been set up, when Netscape receives a file which is served as `application/x-sas`, it will

1. Save the SAS file in a temporary directory.
2. Run SAS in batch mode on that file.
3. Write the SAS `.log` and `.lst` files to Netscape’s temporary directory.

This is not very useful for web-based SAS applications, because the results are not displayed in the browser window, and insecure, because there is nothing to stop the `.sas` program from performing unwanted housecleaning on your hard-disk.

Let the user decide

An alternative to serving *all* `.sas` files as an application mime type is to use a cgi script which allows the file to be served either as `plain/text` or as `application/x-sas`. The choice can be made either by examining the CGI environment variable `HTTP_ACCEPT` (which contains the list of mime types acceptable to the browser), or by presenting the file first as `text/plain` for the user to view, and providing a link which would cause the script to run again, but serve the file as `application/x-sas`.

`sasfile` CGI script

As an example of this kind of interaction, I’ve written a Perl CGI script, `sasfile`. In addition to providing a more secure environment for running SAS on the client, the script allows more intelligent and literate ways to serve SAS programs to remote users.

The script is normally invoked from a web document by including a reference of the form

```
<a href="http://server.name/cgi-bin/sasfile/
path/to/file.sas">file.sas</a>
```

When invoked in this way, the script sends an HTML file which includes the text of file.sas and a link like this:

```
<a href="http://server.name/cgi-bin/sasfile/
path/to/file.sas?run">file.sas
<IMG SRC="launch.gif"> Click here to run me
on your machine!</a>
```

The addition of the keyword parameter **?run** to the URL is the signal the script uses to decide to send the file as `application/x-sas`. To plug a security hole, the script checks that it was indeed called from the previous instance, rather than having been entered directly with the **?run** parameter appended.

Handling %includes

The sample programs for my courses are set up to include datasets stored in a common directory, and referenced in the program by

```
%include data(datafile);
```

where `data` is defined as a collection of directories to be searched by a `filename` statement in the system `autoexec.sas` file, e.g.,

```
filename data ('n:\data',
    'n:\psy3030\data',
    'n:\psy6140\data' );
```

These programs also use a collection of directories for macro programs, set up as an autocall library. Since a remote user is not likely to have these data and macro files, the `sasfile` script scans the lines of the original files for lines starting with `%include`, and presents a list of them, in the form:

> `file.sas` includes references to the following server data files and macros. You may wish to download them first. (With most browsers, shift-Click on the link and select an appropriate directory.)
>
> - %include data(nations)
> - %include macros(resline)

where each of these is an active link to the included file.

Literate programming

Serving `.sas` files by a script offers other advantages as well. SAS programs can be made more literate by incorporating documentation together with the program source, in a way that the program code can be read more easily, can be used to describe algorithms, and so forth.

Here are a few ideas:

- Collections of SAS programs placed on the web can be automatically scanned for their descriptions (extracted from `title` statements, for example), to create meaningful `index.html` files for users. I do this with my course files, which makes it far easier for students to find what they are looking for.
- Rather than documenting a program with boxed comment blocks, one can use special comment delimeters, such as

  ```
  /*:
  :*/
  ```

  which can signal a cgi script to treat lines contained within as ordinary text, rather than pre-formatted text.
- General macros can also be documented with special comment delimiters which contain descriptions and usage notes, for example:

  ```
  /*<
  =head1 Description:
    The omelet macro prepares an omelet from
    the contents of a SAS data set.
  =head1 Usage:
      %omelet(data=_last_, eggs=, style=);
      ...
  >*/
  ```

  The lines included within such documentation blocks can then be automatically extracted and transformed into web documentation or other formats, similar to the way in which Perl “pod” documentation is handled, or the way Unix man pages can be transformed.

In a related idea, [5] describes a literate programming style for LispStat in which LispStat code and LaTeX description are blended in one hyper-document. The hyper-document can be printed with LaTeX, and the code can be run from the same document.

Further information, source code, and online demonstrations of these ideas are available on the Web at `http://www.math.yorku.ca/SCS/Online/`. Pointers to other web-based statistical computing applications can be found in `http://www.math.yorku.ca/SCS/StatResource.html`.

References

[1] M. Friendly. Psychology 3030: Intermediate statistics. WWW document, 1995. `http://www.yorku.ca/dept/psych/lab/psy3030/`.

[2] M. Friendly. Psychology 6140: Multivariate data analysis. WWW document, 1995. `http://www.yorku.ca/dept/psych/lab/psy6140/`.

[3] L. Hoyle. Connecting SAS to the WWW - Forms across the Internet. In *Mid West SAS Users Group Conference*, 1994. `http://stat1.cc.ukans.edu/pub/ippbr/papers/mwsug94/mwsug94.pdf`.

[4] L. Hoyle. More on using SAS with WWW. In *Mid West SAS Users Group Conference*, October 1995. `http://stat1.cc.ukans.edu/scripts/ippbr/cgisas_mwsug95`.

[5] B. Narasinham. A literate program for a Hyper-Documnent. WWW document, 1996. `http://euler.bd.psu.edu/lispstat/stbl-prog.html`.