

Internet Publishing with SAS/Base using Perl in a UNIX Environment

Wade H. Van Buskirk, Pacific States Marine Fisheries Commission, Gladstone, Oregon

INTRODUCTION

The method developed here for publishing data from SAS using a web server is known as the Computer Gateway Interface or CGI scripting method. CGI scripts normally operate in a batch nature in reaction to a web form submitted from a web browser. The script working in conjunction with the web server updates a frame or document on the browser. The update passed back to the web server from the CGI script can be generated by a batch SAS session.

Development of this technique was independent of the SAS/CGI and Web Broker in the SAS/Internet product and other available SAS CGI scripts. This method and the Web Broker can be distinguished by their ability to execute SAS programs on an independent computer known here as a "SAS server" somewhere on the Internet. Other methods of serving SAS data through the Internet such as with Java applets or some other client side application exist. These other methods technically operate more interactively with a SAS server to update the client web browser.

Purpose

The purpose of this paper is to make the use of CGI scripts written in Perl and the supporting SAS programs for Internet publishing simpler. Although these were developed in UNIX, developers in other environments may also benefit.

This paper is written as a step by step guide to development of an interactive dynamic SAS Internet publishing system.

Background

SAS has been the primary database manager for the Marine Recreational Fisheries Statistics Survey (MRFSS) since 1979. The National Marine Fisheries Service (NMFS) operated the database in the DOS operating system until 1994 when it was moved onto UNIX to improve access options and speed. The MRFSS is an extremely important and complex fisheries survey, which can be difficult to administer. Until 1996, all of the data were reported using large sets of printed tables, which were limited to only a few general summaries on an annual basis.

The recent explosion of internet use by government agencies provided an obvious information reporting advantage over printed publications for the MRFSS

and fisheries data in general. The Pacific States Marine Fisheries Commission (PSMFC) acts as a fisheries data warehouse for the Pacific states. PSMFC established the Recreational Fisheries Information Network (RecFIN) to conduct the field collection of the MRFSS and centralize this with any other sport fishery data for the member states.

Review

There are many cases of CGI scripts written in Perl (practical extraction and reporting language) to connect web servers with SAS. All the available CGI scripts found in this review were configured with SAS installed on the same machine as the web server. The purpose of this arrangement is to allow the CGI process to easily communicate with SAS via a shared environment. This shared environment includes variables, an inter-process communication (IPC) pipe and a local file system.

However, this happy arrangement of sharing the same machine is not always practical. The web server machine may already may be tasked with several other processing requirements from other projects and have insufficient storage space. It also may be more convenient to develop the SAS programs on a separate machine where the source data are normally processed. In our case, we have a single SAS license for one machine so cost of an additional license was a big factor.

Other CGI Methods

An early CGI script written as a UNIX shell script (similar to a DOS batch file) by Larry Hoyle in 1994 started SAS on a web server machine. In this case, SAS did all the work of reading the environment, decoding what was requested and returning a product. Once the popularity of the web increased the use of Perl to write CGI scripts for UNIX was popularized.

Perl is a language developed for text pattern matching, file manipulation and process control that filled a niche between UNIX shell scripts and the C programming language. Perl is powerful, fast, is available for many operating systems and is free. Perl programmers have written many useful program libraries to assist in CGI scripting.

CGI scripts written in Perl normally re-code web server requests in a way that is more digestible to

Applications and Publishing

further processing. SAS programmers have used Perl to devise numerous methods to invoke SAS with a product request. These methods include either writing some SAS code to a file or adding more environment variables to be read by a SAS program. Early efforts used dedicated CGI scripts for each type of request or SAS program.

It is possible that one general-purpose CGI script can be written to start different SAS programs. We have found references to and copies of other general purpose CGI programs written in Perl. Most expect SAS to be installed on the web server machine in order to read a file or the environment and execute SAS. Some would require further modification to become truly general purpose. Five are documented on the Internet.

General Purpose Perl scripts for SAS

Author	Name	SAS Input Method
John Blodgett	get2sas.pl	pipes query string to SAS program via stdin
Larry Hoyle	minimal.pl	writes file with macro code '%let name=value' and %includes SAS program
Michael Friendly	sascgi	breaks up query string and writes individual values to the environment
David Shinn and John Hansen	sascgi.pl	writes file with SAS code 'call symput(name,value)' and %includes SAS program
Wade Van Buskirk	sas.cgi	passes macro declarations with '%let name=value' via sysparm

METHODS

Development of this method was on Sun Sparc 5 workstations running Solaris 2.6, Netscape Enterprise Server 3.51, Perl 5 and SAS 6.12 on an TCP/IP intranet behind a firewall connected to a fractional T1 internet connection. Custom CGI security was implemented to allow only listed scripts to execute under the owners permissions (this function now available on many web servers). The secure shell was installed and tested.

SAS programs were debugged and tested in interactive SAS sessions. SAS product programming was written within a main macro to allow for use of global macro values. Web forms were developed using either Netscape Composer or Microsoft Office 97 and edited with a text editor.

The CGI program produced (sas.cgi) was made as general as possible to allow for a diversity of situations. Editing of text files was in Nedit on Solaris and in Windows using notepad. Graphics were captured in Lview Pro and pasted into MS Word.

The exploratory approach used here expects the target audience to be experienced with the SAS macro language, SAS language and functions and some Base Procedures. The audience would benefit from some experience with other computing languages such as Perl and the UNIX operating system. Experience with Hypertext Markup Language (HTML) is desirable.

DEVELOPMENTAL PROCEDURE

The process of development is illustrated in steps with examples to show how each part of the CGI script works with SAS and the operating system. Explanations and comments are provided where necessary to assist in understanding the programming and illustrate operation. Similar steps could be followed to develop a solution on a different operating system.

The solution is derived in two converging tracks. First, we will develop the batch access to SAS from the command prompt. Second, we will develop the Perl script to invoke SAS from the command prompt. Then we will put the two together, implement its use in a browser and add more features.

Developing a Flexible SAS Application

This example data set will be used:

```
(SAS PROGRAM EDITOR)
data bedrock;
infile cards delimiter=',';
input name:$25. year month spent;
cards;
Fred Flintstone,1998,1,12.50
Fred Flintstone,1998,2,22.50
Barney Rubble,1998,1,1.50
Barney Rubble,1998,2,2.50
Wilma Flintstone,1998,1,10.50
Wilma Flintstone,1998,2,32.00
;
run;
```

A main macro is used to control product options and supply default values for a table. The data are also filtered with a 'where' option. These defaults are assigned to global macro variables outside of the main macro.

```
(SAS PROGRAM EDITOR)
* DEFAULT VALUES;
%let year=1998;
%let month=1;
%let row=name;
%let col=year;
%let var=spent;
```

```

* MAIN;
%macro main;
  * make table;
  proc tabulate
    data=bedrock (where=(year=&year
                        and month=&month));
    class year month &col &row;
    var &var;
    table &row,&col*sum=' '*&var*
      f=dollar6.2;
  run;
%mend main;
%main;      * call main macro;

```

The program is run in interactive mode and the product is examined for format and content in the output window.

(output window)

	YEAR
	1998
	SPENT
NAME	
Barney Rubble	\$1.50
Fred Flintstone	\$12.50
Wilma Flintstone	\$10.50

A title is added to the program and the output redirected to stdout (the terminal session window that started the SAS session)

```

(SAS PROGRAM EDITOR)
proc printto print=stdout;run;

title EXPENDITURES BY %upcase(&row) AND
%upcase(&col) FOR MONTH &month, &year;

```

When the program is run at this point, the table will appear in the terminal window instead of the output window.

```

(COMMAND PROMPT)
lingcod:/usr/demo>
EXPENDITURES BY NAME AND YEAR FOR MONTH 1, 1998

```

	YEAR
	1998
	SPENT
NAME	
Barney Rubble	\$1.50
Fred Flintstone	\$12.50
Wilma Flintstone	\$10.50

Now the program can be saved (in this example as 'bedrock.sas') and run as a batch program that will return this product to the screen.

The Sysparm Trick

It is possible to change any one of the default macro variables by including a macro definition along with the batch SAS command. This is done by including macro code with the '-sysparm' parameter. To execute the macro code put a '%sysparm' statement in the program before invoking the main macro. Now the program looks like this (shaded portions were added):

```

(SAS PROGRAM EDITOR - bedrock.sas)
* DEFAULT VALUES;

%let year=1998;
%let month=1;
%let row=name;
%let col=year;
%let var=spent;

* MAIN;

%macro main;

  * make table;

  proc printto print=stdout;run; * send to terminal;

  title EXPENDITURES BY %upcase(&row) AND
%upcase(&col) FOR MONTH &month, &year;

  proc tabulate data=bedrock (where=(year=&year and
month=&month));
    class year month &col &row;
    var &var;
    table &row,&col*sum*&var*f=dollar6.2;
  run;

  proc printto print=out;run; * send to output
window;

%mend main;

* PROGRAM;

%sysparm; * set passed macro values;
%main;    * call main macro;

```

Now when we run SAS as a command and pass a new value to the program with 'sysparm' we get a different product.

```

(COMMAND PROMPT)
lingcod:/usr/demo>
sas bedrock -sysparm "%let month=2;"

EXPENDITURES BY NAME AND YEAR FOR MONTH 2, 1998

```

	YEAR
	1998
	SPENT
NAME	
Barney Rubble	\$2.50
Fred Flintstone	\$22.50
Wilma Flintstone	\$32.00

Applications and Publishing

As many macro variables as necessary may be defined in this way. Separate each '%let name=value' with a semicolon.

This program can also be run locally or remotely using the standard 'remote' or 'r' shell commands. When this is done then the path to both the SAS executable and the SAS program must be included in the command line. The product is still output on the screen. (The last parameter below has line wrapped)

From a local directory command line:

```
(COMMAND PROMPT)
lingcod:> rsh lingcod /opt/sas612/sas
/usr/demo/bedrock
```

From a remote directory command line:

```
(COMMAND PROMPT)
coho:> rsh lingcod /opt/sas612/sas
/usr/demo/bedrock
```

The SAS program is now ready to be evoked from a Perl program.

Developing a CGI Script

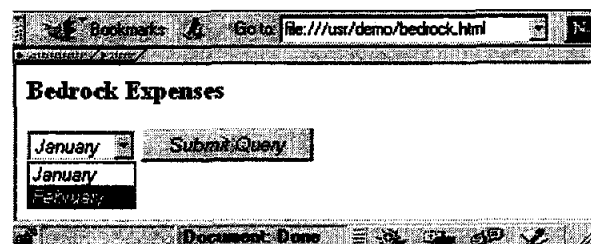
When a web form is submitted a query string is produced from the data elements in the form. Each data element is composed of a name and a value expressed as an equality with a '='. Each pair is further delimited by an '&'. This can be verified by submitting a form in your browser and examining the URL. The resulting pairs are based on the values selected in the form.

This example form will be used.

```
(TEXT EDITOR - bedrock.html)
<html><body>
<h3>Bedrock Expenses</h3>
<form action=http://www.psmfc.org>
  <select name=month>
    <option value=1>January
    <option value=2>February
  </select>
  <input type=hidden name=debug value=1>

  <input type=submit>
</form>
</html></body>
```

The form looks like this in a web browser with the select menu chosen to select 'February.' Note that the debug element is hidden from view.



When the form is submitted nothing happens because no program was specified in the form action, but the URL becomes:

```
(WEB BROWSER)
http://www.psmfc.org?month=2&debug=1/
```

The part after the '?' is the 'query string' and is put into the environment variable 'QUERY_STRING' when a CGI script is started by the web server. Which script is started is defined by the forms 'action' property (in this case no script has been specified).

In a Perl script we can read the query string into an array from the environment and print it out as a web page. Each name=value pair is associated in an array. In the above example, the array size is two.

This example Perl code will be used:

```
(TEXT EDITOR - sas.pl)
#!/bin/perl
# -*- perl -*-
# FILE: SAS.PL - DEMO PERL CODE

$in = $ENV{'QUERY_STRING'}; # get query
@in = split(/&/,$in); # make array

# associate name with value and decode query
foreach $i (0 .. $#in) {
  $in[$i] =~ s/\+/ /g;
  ($key, $val) = split(/=/,$in[$i],2);
  $in{$key} .= $val;
  $in{$key} =~ s/%(..)/pack("c",hex($1))/ge;
}
$in{pid}=$$; # add pid

# if debug value then print report
if ($in{debug}) {
  print "<h3>CGI Values:</h3><ul>"; # start list
  foreach $key (keys(%in)){
    print "<li>" . $key . " = " . $in{$key};
  }
  print "</ul>"; # end list
}
```

In this case, we have added the process id (pid) to the array and before printing a mini report for this query. Each CGI script started on the web server gets a separate process number. Each CGI process is a 'child process' of the web server 'parent process'. This will become a convenient way to keep track of simultaneous queries.

We can run the Perl program from the command line by saving it to a file and making it executable.

```
(COMMAND PROMPT)
lingcod:/usr/demo> chmod a+x sas.pl
ls -l sas.pl
-rwxr-xr-x wade staff 1185 May 22 16:13 sas.pl
```

Since the script will be looking for the query string in the environment, we will export one from this terminal session (parent process):

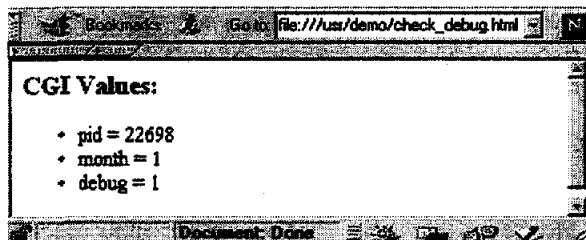
```
(COMMAND PROMPT)
lingcod:/usr/demo> QUERY_STRING='debug=1&month=2'
lingcod:/usr/demo> export QUERY_STRING
```

We can now run the Perl program to read the environment and produce output. We will produce a mini report on the form if we put a value in the name 'debug.' Having a flag like 'debug' is called a script function or a script control since it is trapped and acted on by the Perl program.

```
(COMMAND PROMPT)
lingcod:/usr/demo>./sas.pl
```

```
<h3>CGI Values:</h3><ul><li> pid = 25739<li>month =
2<li>debug = 1</ul>
```

Putting the above HTML result into a file (check_debug.html) and reading it with a web browser produces a web page.



In order to convert the query string 'month=2 &debug=1' into the SAS macro code '%let month=2; %let debug=1' is very simple in Perl using the associative array:

```
(TEXT EDITOR - sas.pl)
foreach $key (keys(%in)){
    $out = $out . "%let " . $key . "=" . $in{$key} .
    "\; ";
}
print "\n" . $out . "\n"
```

We can run the program again now to see what we put into the '\$out' variable.

```
(COMMAND PROMPT)
lingcod:/usr/demo>./sas.pl
```

```
<h3>CGI Values:</h3><ul><li> pid = 25740<li>month =
2<li>debug = 1</ul>
%let pid=25740; %let month=2; %let debug=1;
```

The out variable contains the 'sysparm' string we need to produce the SAS product programmed in the previous section.

Putting CGI and SAS Together

Since we can produce a user defined product by using SAS by passing macro value declarations with a batch command. In addition, we can generate those macro values declarations from a Perl program by reading the query environment. Then we can connect the two and run the Perl program to get a user defined SAS product from a web query.

We need to get the Perl program to evoke the batch SAS program and pass the sysparm we built. This is done using the 'system' function in Perl.

```
(TEXT EDITOR -sas.pl)
```

```
system(" sas bedrock -sysparm '$out'");
```

We should check the environment for our query string we exported earlier.

```
(COMMAND PROMPT)
lingcod:/usr/demo>echo $QUERY_STRING
month=2&debug=1
```

Since the query is in the environment, we can run the Perl program to start SAS and pass the macro code.

```
(COMMAND PROMPT)
lingcod:/usr/demo>./sas.pl
```

```
<h3>CGI Values:</h3><ul><li> pid = 25741<li>month =
2<li>debug = 1</ul>
```

EXPENDITURES BY NAME AND YEAR FOR MONTH 2, 1998

NAME	YEAR
	SPENT
Barney Rubble	\$2.50
Fred Flintstone	\$22.50
Wilma Flintstone	\$32.00

We have changed the default 'January' table to the user defined 'February' table using the environment variable we exported. When a web server starts a CGI script, it exports the query string automatically.

We can add another line of Perl code to inform the user that we are using the power of SAS to produce a report. Next we will put this script in the web server CGI area and run it from a web browser.

In this case, we are running the SAS program on a separate machine on the network using the 'rsh' command described earlier. The parts of the 'rsh' command have been broken out into separate variables to allow changes to be made from the query string and the resulting sysparm. The quoting for 'sysparm' was a little tricky in Perl. The HTML '<pre>' tags were added to define the plain text tables as being pre-formatted with carriage returns.

Here is the Perl program 'sas.cgi' (now a CGI script) after our editing (shaded portions show changes):

```
(TEXT EDITOR - sas.cgi)
#!/bin/perl
# -*- perl -*-
# FILE: SAS.PL - DEMO PERL CODE

print "Content-type: text/html\n\n"; #HTML prefix
$in = $ENV{'QUERY_STRING'}; # get query
@in = split(/&/,$in); # make array

# associate name with value and decode query
foreach $i (0 .. $#in) {
    $in[$i] =~ s/\+ / /g;
```

Applications and Publishing

```

($key, $val) = split(/=/,$in[$i],2);
$in[$key] .= $val;
$in[$key] =~ s/%(..)/pack("c",hex($1))/ge;
}
$in{pid}=$$; # add pid

# if debug value then print report
if ($in{debug}) {
    print "<h3>CGI Values:</h3><ul>"; # start list
    foreach $key (keys(%in)){
        print "<li>" . $key . " \=" . $in{$key};
    }
    print "</ul>"; # end list
}

$server="lingcod";
$saspath="/opt/sas612/sas";
$prog="/usr/demo/bedrock";
$parm = "-sysparm \"" . $out . "\"";

print "<html><pre>Web process " . $in{pid} . "
Submitting query to SAS...</pre>";print "\n";
print "<pre>"
system(" rsh '$server' '$saspath' '$prog'
'$parm'");
print "</pre>"

exit (0);

```

The script must be installed on a web server in the CGI bin in order to be executed. Scripts anywhere else will be downloaded instead of executed. This security feature prevents the browser from executing remote programs on itself and possibly causing damage to the file system.

The form action must be changed to point at the CGI bin and the CGI program to run.

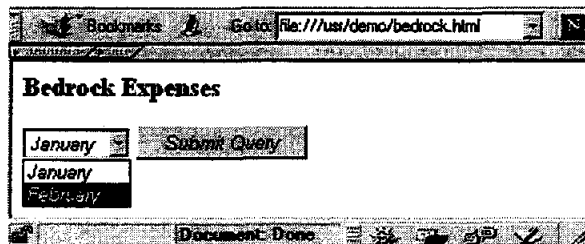
```

(TEXT EDITOR -bedrock.html)
<html><body>
<form action=http://www.psmfc.org/cgi-
bin/recfin/sas.pl>
  <select name=month>
    <option value=1>January
    <option value=2>February
  </select>
  <input type=hidden name=debug value=1>

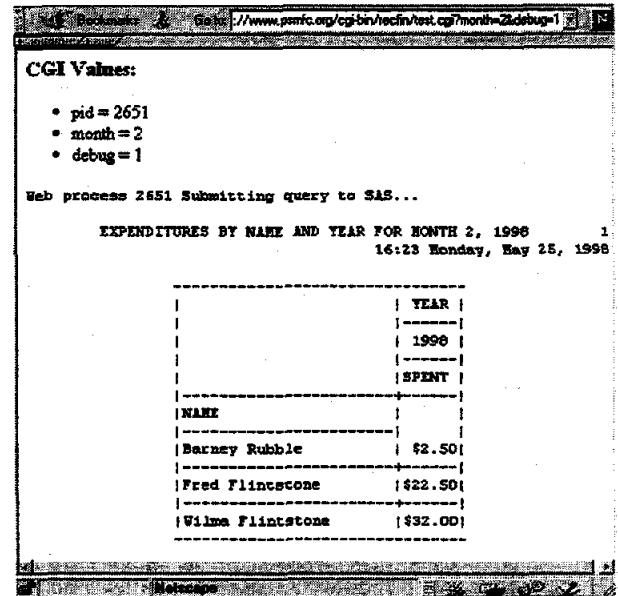
  <input type=submit>
</form>
</html></body>

```

Our form looks the same as before.



When we submit it now we get the URL defined in the form action with the query string added after the '?'. We also get our combination Perl and SAS product page.



The hidden 'debug' field can be removed from the web page to eliminate the mini 'CGI Values' report.

Making Enhancements

At this point, we are ready to define the parts of the 'rsh' command using the CGI script by passing parameters from hidden fields in the HTML form. This will let us define the SAS server and SAS program to be used in the web. This will make the CGI program independent of the SAS server and the SAS program to be run.

Making sas.cgi Independent

First, we define some hidden fields in the HTML form to define the rsh command to be used in sas.cgi.

```

(TEXT EDITOR - bedrock.html)
<html><body>
<form action=http://www.psmfc.org>
  <select name=month>
    <option value=1>January
    <option value=2>February
  </select>
  <input type=hidden name=debug value=1>
  <input type=hidden name=server value=lingcod>
  <input type=hidden name=saspath
value=/opt/sas612/sas>
  <input type=hidden name=progpah value=/usr/demo>
  <input type=hidden name=program value=bedrock>
  <input type=hidden name=logpath value=/usr/logs>

  <input type=submit>
</form>
</html></body>

```

Now we can replace the rsh parameters we created in the last edit of *sas.cgi*:

```

(TEXT EDITOR - sas.cgi, deleted)
$server="lingcod";
$saspath="/opt/sas612/sas";
$prog="/usr/demo/bedrock";

```

With the associative array values that will be present when we submit the revised form *bedrock.html*.

```
(TEXT EDITOR - sas.cgi, revised)
system("rsh '$in(server)' '$in(saspath)'
'$in(progpath)$in(program)' '$sparm' -log
'$in(logpath)'/'$in(program)$in(pid)'.log");
print "</pre>"
```

We have separated the program path from the program name so that we can create a unique identifier for each query. The unique query identifier or 'query reference' will be the program name and the PID together (i.e. 'bedrock12345'). We also add a parameter to define the path and name of the SAS log; in this case, we name it the 'query reference'.log

Writing HTML from SAS

To make the *sas.cgi* output completely independent, we should move any HTML tags to the SAS program. To output HTML from SAS, we have written %comment. In %comment we make a global macro variable called '&options' to hold the default options statement. When a text string is passed to the %comment macro, we temporarily change the options so no SAS messages are displayed. Then, we redirect log output to the fileref 'out', the UNIX stdout device with long records, and put the text string. Then the log is directed back to normal and the default options are reinstated. This method avoids the limitation of 256 characters in a data step 'put'.

```
(SAS PROGRAM EDITOR - bedrock.sas)
%let options=OPTIONS mtrace mprint symbolgen
nonumber source notes ps=1000 ls=70;
&options.;
file out '/dev/stdout' lrecl=1000;
```

```
* DEFAULT VALUES;
%let year=1998;
%let month=1;
%let row=name;
%let col=year;
%let var=spent;
```

```
%macro comment(text);
options nostimer nofulltimer nosource nonotes
ls=256 nomtrace nomprint nosymbolgen;
proc printto log=out;run;
%put &text.;
proc printto log=log;run;
&options.;
%mend comment;
```

```
* MAIN;
%macro main;
```

```
* make table;
```

```
proc printto print=out;run; * send to terminal;
```

```
moname=%scan(JANUARY FEBRUARY MARCH APRIL MAY JUNE
JULY AUGUST SEPTEMBER OCTOBER NOVEMBER DECEMBER,
&month);
%comment(<h3>BEDROCK EXPENDITURES BY %upcase(&row)
AND %upcase(&col) FOR MONTH %moname
&year<h1><pre>);
```

```
proc tabulate data=bedrock (where=(year=&year and
month=&month));
```

```
class year month &col &row;
var &var;
table &row,&col*sum*&var*f=dollar6.2;
run;
```

```
%comment(</pre>Have a comment? <a href=mailto:
wade@psmfc.org?subject=&program&pid>Send Mail</a>);
```

```
%mend main;
```

```
* PROGRAM;
```

```
&sysparm; * set passed macro values;
%main; * call main macro;
```

Above we have changed the title to HTML level 3 title format and opened a preformatted area with the 'pre' tag for the SAS table. Then we close the 'pre' tag. We have also used %comment to add a hyperlink to an email message for sending a comment. We have set the subject of the email to the 'query reference' (program name + process id). When we submit this form now we get back a nicer product.

NAME	YEAR	SPENT
Barney Rubble	1998	\$2.50
Fred Flintstone	1998	\$32.50
Wilma Flintstone	1998	\$32.00

We also have a way for the user to make a comment to that specific query 'bedrock25933' so we can check the SAS logs for 'bedrock25933.log' to see if there was a problem.

Linking HTML Queries with SAS logs

Due to the length of the query string, it would be difficult to reproduce queries for debugging purposes. However, the CGI script can capture the query string and save it to an HTML query log as a hyperlink. With the hyperlink we can reproduce queries and see

Applications and Publishing

what product was produced. This requires a piece of Perl code in sas.cgi to write to an HTML SAS query log. The '\$in' variable is the query string we are using after the '?' in the first hyperlink below.

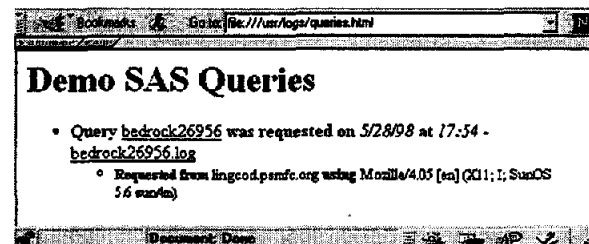
```
(TEXT EDITOR - sas.cgi, revised)
open(qlog,">>" . $in(logpath) . "/queries.html");
print saslog "<li><b>Query </b><a
href=http://www.psmfc.org/cgi-bin/recfin/sas.cgi?\"
. $in . ">>\" . $in(program) . $in(pid) . "</a><b>
was requested on </b><i>\" . $mon . "/" . $mday .
"/\" . $y . "</i><b> at </b><i>\" . $hour . ":" .
$min . "</i><b>\" .
print saslog \"<a href=ftp://wade\@\"
$in(server) . "/" . $in(logpath) . "/" . $in(program)
$in(pid) . ".log\" . $in(program) . $in(pid) .
".log</a>\n\";
print saslog <ul><li><small><b>Requested
from</b> \" . $host . \"<b>using</b> \" . $agent .
\"</small></li></ul>\n\";
close qlog;
```

To show the date, time and machine that requested the query along with the query hyperlink, we read the web server environment and call the system clock.

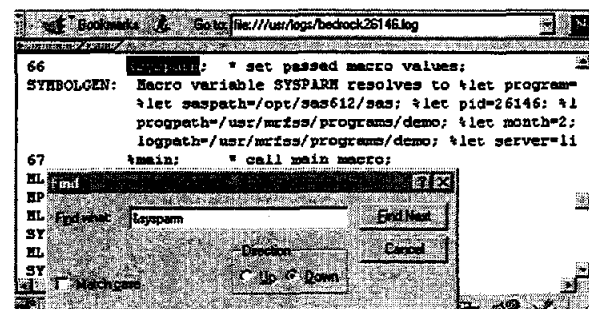
(TEXT EDITOR - sas.cgi, revised)

```
$host = $ENV{'REMOTE_HOST'}; # get host
$agent = $ENV{'HTTP_USER_AGENT'}; # get browser
($s,$min,$hour,$mday,$mon,$y,$w,$d,$i)=localtime(t
ime); # time and date
$mon=$mon+1;
```

When we open the SAS query log file in a web browser we see an entry for each query made through sas.cgi. Each entry has hyperlinks to reproduce the query and view the SAS log. The HTML header and title for the query file were set when the queries.html file was created.



When we click on the link to the log we can view the SAS log in the web browser and search the contents.

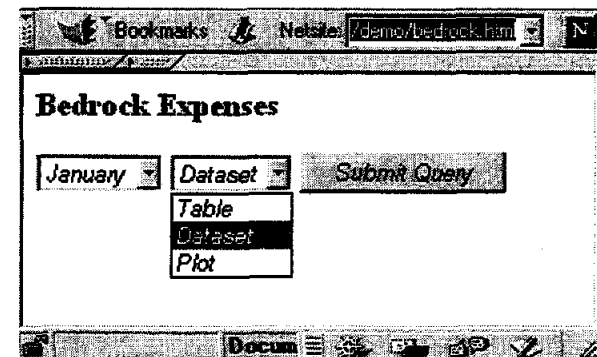


Linking Output Files

Providing data and graphic files to the user from SAS consists of writing the file on the local file system, moving the file to a public area on the web server and including a hyperlink to the file in the result. For data files the link is typically an '<a href>' tag that links to a separate document (the data). For graphics, the link is an '' tag that displays the graphic in the same document. For our example, we add an option for a data file in the form and in the SAS program.

```
(TEXT EDITOR - bedrock.html)
<html><body>
<h3>Bedrock Expenses</h3>
<form action=http://www.psmfc.org/cgi-
bin/recfin/sas2.cgi>
  <input type=hidden name=server value=lingcod>
  <input type=hidden name=saspath
value=/opt/sas612/sas>
  <input type=hidden name=progpath
value=/usr/mrfss/programs/demo>
  <input type=hidden name=program value=bedrock>
  <input type=hidden name=logpath
value=/usr/mrfss/programs/demo>
  <input type=hidden name=logpath
value=/usr/www/htdocs/demo>
  <select name=month>
    <option value=1>January
    <option value=2>February
  </select>
  <select name=product>
    <option>Table
    <option>Dataset
    <option>Plot
  </select>
  <input type=submit>
</form>
</html></body>
```

The form now has a title, a new hidden object and a new visible object for selecting the type of product.



We can use SAS to remote copy the bedrock data file to the webserver using the filepath defined in the web form. A new macro '%linkfile(dsn)' provides the code to make the transport file, move the file from the /tmp area on the SAS server to the web server and output the hyperlink to the file.

```
(SAS PROGRAM EDITOR - bedrock.sas)
%macro linkfile(dsn);
  libname home xport "/tmp/&program.&pid..xpt";
  proc copy in=work out=home;
    select &dsn /memtype=data;
  run;
```

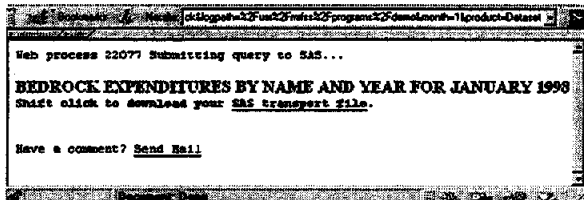
```

%sysexec rcp /tmp/&program.&pid..xpt
  linkcod:=&filepath/sas&pid..xpt;
%comment(<b>Shift click to download your <a
href=/recfin/demo/&program&pid..xpt>SAS transport
file</a>.</b><p> );
%mend linkfile;

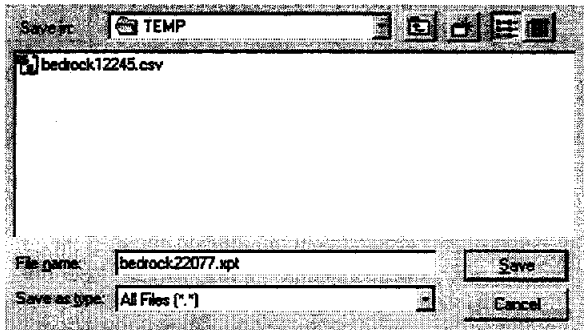
%if &product=Dataset %then %linkfile(bedrock);

```

The response for the web server when the product type is 'Dataset' now contains a hyperlink to the SAS transport file.



Shift-clicking the file opens the 'save as' dialog with the default name of the file. The default name is the query reference with the appropriate extension.



The final versions of the code used in these examples is located on the authors website.

REMOTE ACCESS SECURITY

Most sites connected to the Internet operate their servers behind firewalls due to security concerns. The firewall is effective at filtering protocols and services in either direction between the local Intranet and the Internet at large. It would be a breach of security to allow incoming rsh commands to penetrate the site as this could be an avenue of attack by hackers. Most sites would have both the web and SAS servers behind a firewall. Standard remote commands are adequate in most cases. There is an option for allowing secure access to remote machines across the Internet.

Standard 'R' Commands

Standard remote shell commands (rsh, rcp, rdist, etc.) within an intranet between trusted machines that rarely go down are secure from all but the most powerful attacks from hackers on the internet. UNIX machines typically come with the remote shell server

or 'daemon' running on the host server. Trusted client machines are listed in a file on the server and are checked against the identity of the requestor when the remote shell server receives a request.

Secure Shell

The secure shell standard (ssh, scp, sdist, etc.) can operate in a secure manner in an Intranet and out on the Internet. It uses the RSA security protocol with private and public encryption keys. The Client machine provides a public key to the server when the service is set up. During connections the keys are used to validate the initial connection and triple DES encrypt the contents of the data.

The secure shell is not vulnerable to listening, snooping and IP, DNS or router spoofing. The default keys are currently up to 1024 bits and sustained connections are renegotiated with new random encryption keys hourly. Only a very powerful external attack or internal tinkering could compromise the private key of a client. Passcode access is an option for increased security. Brute force deciphering of the private keys or packets is impossible in a human lifetime with current technology.

RESULTS

The sas.cgi program saved us money and time. This technique has saved the project, and ultimately the public, the cost of annual licenses for SAS/Internet. It has also given us the opportunity to utilize, to our advantage, control of communication between the web server and the SAS server. The Perl language has allowed us to develop important tools and methods for monitoring query activity and enhancing SAS output more easily than programming in SAS.

Time is saved in development and debugging of new products. SAS development can be done interactively using global macros before or after the web interface is designed. Further CGI programming is not required since sas.cgi does not need to be modified for new SAS products. Development and production queries occur in the same data environment on the SAS server, which makes for rapid location and repair of production problems.

DISCUSSION

The batch nature of web server to browser interaction on the Internet is conducive to the use of SAS batch sessions to produce page content. Other methods such as Java applets can operate within a browser to produce page content with an event driven model.

Applications and Publishing

Both methods can be used to produce an interactive dynamic interface to SAS data and products.

SAS can be used to produce 'hidden' data structures in HTML and JavaScript. These SAS derived data structures can be read using JavaScript to populate web form objects dynamically. This would probably be slower than a Java Applet communicating with a SAS/Share*Net server, but the cost of development would be less and JavaScript is much simpler to program than Java with sockets. When using this CGI method all the socket connection details are taken care of by the web server and the remote commands.

CONCLUSION

The development of a general-purpose sas.cgi with associated techniques has greatly reduced the complexity of our previous Internet publishing system. The Internet publishing system has nearly eliminated the need for hardcopy distributions. The ability to rapidly develop web based applications in SAS allowed the production of many more types of products than would be the case with lower level languages. Many of our existing SAS programs were quickly converted to produce web content.

Some limitations in HTML forms can be overcome with JavaScript on the client side. Although Perl is not a simple language it can be characterized that, like SAS, it is very powerful for what it was designed to do. We hope that general-purpose CGI scripts for SAS will allow SAS programmers to concentrate on SAS, HTML and perhaps JavaScript in their development efforts.

The use of our data in fisheries management has increased in importance and is becoming more widely used and respected. The system currently averages about 20 queries per day.

REFERENCES

All Internet URLs were active on May 31, 1998

Blodgett, John. get2sas.pl. <http://www.oseda.missouri.edu/uicapps/cgis/get2sas.pl>

Friendly, Michael. sascgi: A SAS - WWW Gateway. <http://www.math.yorku.ca/SCS/Online/sascgi/>

Gleason Chapman et. al. Interfacing SAS to The WWW. <http://jse.stat.ncsu.edu:70/R1275216-1281371-1m/othergroups/sas1/log9602>

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

Jakubson George. Web Interface to SAS. http://instruct1.cit.cornell.edu/courses/econ321/public_html/sasinfo/runsas2.htm

Lefebvre, Eric and Thomas Kunselman. From Surfing-to-Jet Skiing: A Comparison of SAS. Web Integration Tools Before and After SAS/IntrNet Software. <http://www.pace.edu/nesug/97/PREVIEW/iwp07.html>

RSA Data Security. Home Page. <http://www.rsa.com/>

Shinn, David and John Hansen. SAS Software and the Web: Creating a Common Gateway Interface. http://www.sas.com/service/doc/periodicals/obs/ind_files/v6n22pp6.html

Sloan, Faith R. Sample SAS/CGI Application using HTML Data and Report Formatting Tools. <http://faith.hypno.net/sugi22/sugi22.txt>

Wehr, Paul. Re: Web Pages and SAS. <http://jse.stat.ncsu.edu:70/R1339489-1342267-1m/othergroups/sas1/log9601>

Workman, Robert. Using PERL To Prepare Data Collected from the Internet for SAS. Input. <http://scsu.ctstateu.edu/~workman/NESASUG97.HTML>

Ylönen, Tatu. SSH (Secure Shell) Remote Login Program. <http://www.cs.hut.fi/ssh/>

CONTACT

The authors contact preference is email.

✉ Wade H. Van Buskirk
✉ Pacific States Marine Fisheries Commission
✉ 45 SE 82nd Drive, Gladstone, OR 97027-2522
✉ 503-650-5400
✉ 503-650-5426
✉ <http://www.psmfc.org/recfin>
✉ wade@psmfc.org

⚖ Trademark names belong to the owners.

🎵 All coding is GNU Copyright(C) 1998 by author.

☐ This is not an endorsement for any product.

CITATION

Van Buskirk, Wade H. Internet Publishing with SAS/BASE using Perl in a UNIX Environment. Proceedings of the Pacific Northwest SAS Users Group, June 11, 1998

or

Van Buskirk, Wade H. Internet Publishing with SAS/BASE using Perl in a UNIX Environment. <http://www.psmfc.org/~wade/pub/sas/nwsug17>