

Connecting SAS® to the World Wide Web - Forms Across the Internet
Larry Hoyle
Institute for Public Policy and Business Research
University of Kansas

Abstract

The World Wide Web is an explosively growing internet facility which allows for hypermedia documents to be distributed across the internet. Recent extensions to the Hypertext Markup Language (HTML) upon which it is built allow for the creation of forms which can be serviced by a remote application using the Common Gateway Interface (CGI) to a Hypertext Transfer Protocol Daemon or server (HTTPD). This paper will describe how to use SAS as an HTML forms server.

A specific example for setting up under IBM AIX will be listed. The sample form will allow the selection of county level census data. Using a client program such as NCSA Mosaic, the user will request that a variable be displayed either as a table or as a PostScript map. The request is sent to an HTTP server program at another site on the internet which in turn executes a SAS program which generates the table or the graphic and sends it back across the internet to the user's client program which displays it. (It sounds complicated but it's really not too bad).

Introduction

Recent developments in the tools available for using the internet allow for a forms based front end to SAS®. Free client software such as NCSA Mosaic is available for multiple platforms, including Windows®, Macintosh®, and for Xwindows. The server daemon (httpd) which launches SAS is available for a variety of UNIX environments.

Forms can be created with embedded graphics, radio buttons, check boxes, and text input fields. When a client application connects to the form across the internet, the data entered into the form is sent from the client machine to the daemon on the server machine. The daemon sets certain environment variables and launches the requested application, in our case SAS, with

the data from the form as its standard input.

Once it has processed the input data the SAS program can then respond in a number of ways. It could send back another form, other output formatted for display by the client, such as a document, a graphic, a movie, or a sound. It could also send back a stream of data intended for downloading to the client.

This paper shows a simple form and a SAS application which responds to it. The form allows one to select one of the dozen variables in a SAS dataset for display in one of three modes: in a table, as a PostScript® map, or in a form convenient for spreadsheet input. The form also allows for a request for the display of the SAS program.

The World Wide Web (WWW or W3)

Long ago, in the internet's primeval days (the 80s), most people transferred information across the internet either embedded in e-mail or by using ftp. FTP transfers were done via a command line interface which was not particularly user-friendly.

In 1991 the University of Minnesota Microcomputer, Workstation, Networks Center developed the original gopher. Gopher has evolved into a global information system. Users of gopher select items from menus in order to retrieve various kinds of files which can contain text, images, executables and so on. Menu items can also represent remote connections (telnet sessions.)

The World Wide Web project started in 1989 at CERN in Geneva, Switzerland. By January 1993 there were around 50 known WWW servers around the world. Toward the end of 1993 The National Center for Supercomputing Activities (NCSA) had released browsers for X, Windows and the Macintosh. These browsers could also access gopher, ftp sites, and more.

Version 2 of Mosaic, the NCSA browser, added the capability to do forms. Lynx, a

character based browser from the University of Kansas, also does forms. The forms capability is the feature which enables the interface to SAS discussed here.

The World Wide Web is built upon a hypertext document format called **Hypertext Markup Language** (HTML) and upon a transfer protocol called **Hypertext Transfer Protocol** (HTTP). A client program such as Mosaic is pointed at a resource through a **Universal Resource Locator** (URL). The URL contains information about the type of resource and its location on the internet. The URL:

http://info.cern.ch/hypertext/WWW/History.html begins with *http://* indicating that it is to be accessed through Hypertext Transfer Protocol. It is found on the machine *info.cern.ch* in the directory *hypertext/WWW* as the file *History.html*.


HTML

HTML is an ISO Standard Generalized Markup Language (SGML) document type. An HTML document consists of ASCII text with some embedded markup strings. The markup codes are all sequences of printable characters. These codes allow for text formatting comparable to early word processing programs. They also allow for embedding graphics, links to other HTML documents, or links to other internet resources.

Figure 1 shows an example HTML file which defines the form shown rendered by Mosaic in figure 2. The line:
`<H2>Please select a variable</H2>` contains two markup tags. The first tag, `<H2>`, defines the beginning of a second level heading. The tag `</H2>` marks the end of the heading. The client program will typically render the heading as bold, large, flush left, text with some extra space above and below. There are other text formatting tags in figure 1. The tag `<hr>` defines a horizontal rule, `<p>` defines the end of a paragraph, `` defines the beginning of some bold text, and `
` defines a line break.

The tags listed above began with either `<` or `</` and ended with `>`. Within those delimiters they contained only the name of the tag. Other tags also contain attributes. The two tags in figure 1 which begin with `<a` both contain an

attribute which begins with `href=`. The `<a` tag defines an "anchor" which defines a hypertext link. The `href=` attribute defines the address of the entity to which the user will be transferred when the link is activated. In figure 1 both anchors point to other html documents which will be displayed if the user clicks on the text between the `<a` and `</a` tags. In the first instance, figure 1 would be rendered as:

Click  for more information

When the user clicks on the map, the document `ghostscript.html` would be retrieved from the machine `www.ncsa.uiuc.edu` from across the internet.

The map is placed in the form by a link defined by the `<IMG` tag. The `SRC="showdg.gif"` attribute points to a GIF file in the same directory as the HTML file containing it. The `ALT="here"` defines the text string to be printed if the client can't display the graphic file.

The `<FORM` tag in figure 1 has two attributes. The first `METHOD=POST` tells the client how to structure the data it is sending to the server for the form. The `ACTION=` attribute points to the executable to which the data from the form will be sent. In this case the executable is a shell script named `cgisas_mwsug94`. This shell script starts a SAS program which expects to receive data from the form in figure 1.

Between the `<FORM` and the `</FORM` tags lies the form itself. In this case there is first some heading information followed by two menus and finished by an `<INPUT` tag with a `TYPE="submit"` attribute. The latter causes the client to send a request to the server to execute the `ACTION=` application.

The two menus are delimited by `<MENU>` and `</MENU>` tags. Each item in a menu begins with a `` tag and contains a `<INPUT` tag. The `<NAME=` and `<VALUE=` attributes in each list item define what will be sent to the server if that item is selected. The `<TYPE="radio"` attribute in each list item says that for all items with the same value for `<NAME=`, only one can be selected at a time. Thus if the user selects "Live Births", the data stream sent from the client to the server will include the string `"sasvar=Births"` and no other `"sasvar="` value.

```

<TITLE>Example for MidWest SAS Users Group 1994 Conference - Larry Hoyle </TITLE>
<H1>Connecting SAS to the World Wide Web - Forms Across the Internet</H1>

This is the example for a paper to be presented to the
1994 MidWest SAS Users Group Conference September 25-27 1994 in Omaha.
When you press "Go Get It", your selections will be
passed to a SAS job which extracts 1990 Census data from a SAS dataset
and sends it back to you in the form you requested.
You must have a PostScript viewer such as Ghostscript in order to
view a shaded <b>PostScript map</b> generated by this form. Click
<a href=http://www.ncsa.uiuc.edu/SDG/Software/WinMosaic/ghostscript.html>
<IMG align="middle" alt="here" src="showdmg.gif"> </a>
for information on how to find Ghostscript.
<HR>
<FORM METHOD="POST"
ACTION="http://stat1.cc.ukans.edu:8000/cgi-bin/cgisas_mwsug94">

<H2>Please select a variable</H2>
<MENU>
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Births"> Live Births.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Deaths"> Deaths.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Foodstmp"> Food Stamp recipients.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Mdvhome"> Median Home Value.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Medage"> Median Age.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Pcapinc"> Per Capita Income.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Pcslstax"> Per Capita Sales Tax.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Pop"> Population.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Popdense" CHECKED> Population Density.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Precip"> Precipitation.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Pupteach"> Pupil Teacher Ratio.
<LI> <INPUT TYPE="radio" NAME="sasvar" VALUE="Valcrops"> Total Value Field Crops ($million).
</MENU>

<H2>Please select a format</H2>
<MENU>
<LI> <INPUT TYPE="radio" NAME="request" VALUE="table" CHECKED>HTML table.
<LI> <INPUT TYPE="radio" NAME="request" VALUE="file" >
fixed ascii file for spreadsheet input (use with "load to disk").
<LI> <INPUT TYPE="radio" NAME="request" VALUE="map" >
PostScript map (must have PostScript viewer - see above).
<LI> <INPUT TYPE="radio" NAME="request" VALUE="source" >List the SAS Program.
</MENU>
<br>
To receive your information, press this button: <INPUT TYPE="submit"
VALUE="Go Get It">. <br>

</FORM>
<HR>
Larry Hoyle, <i>lhoyle@stat1.cc.ukans.edu,</i>
<a href=http://kufacts.cc.ukans.edu/cwis/units/IPPBR/IPPBR_main.html>
Institute for Public Policy and Business Research, University of Kansas
</a>

```

Title and Heading

Note that his text gets justified

First anchor
Embedded graphic

Form begins
cgisas_mwsug94 will be invoked

First menu

CHECKED

Second Menu

Submit button

Form ends

Second anchor

Figure 1, An HTML file defining a form.

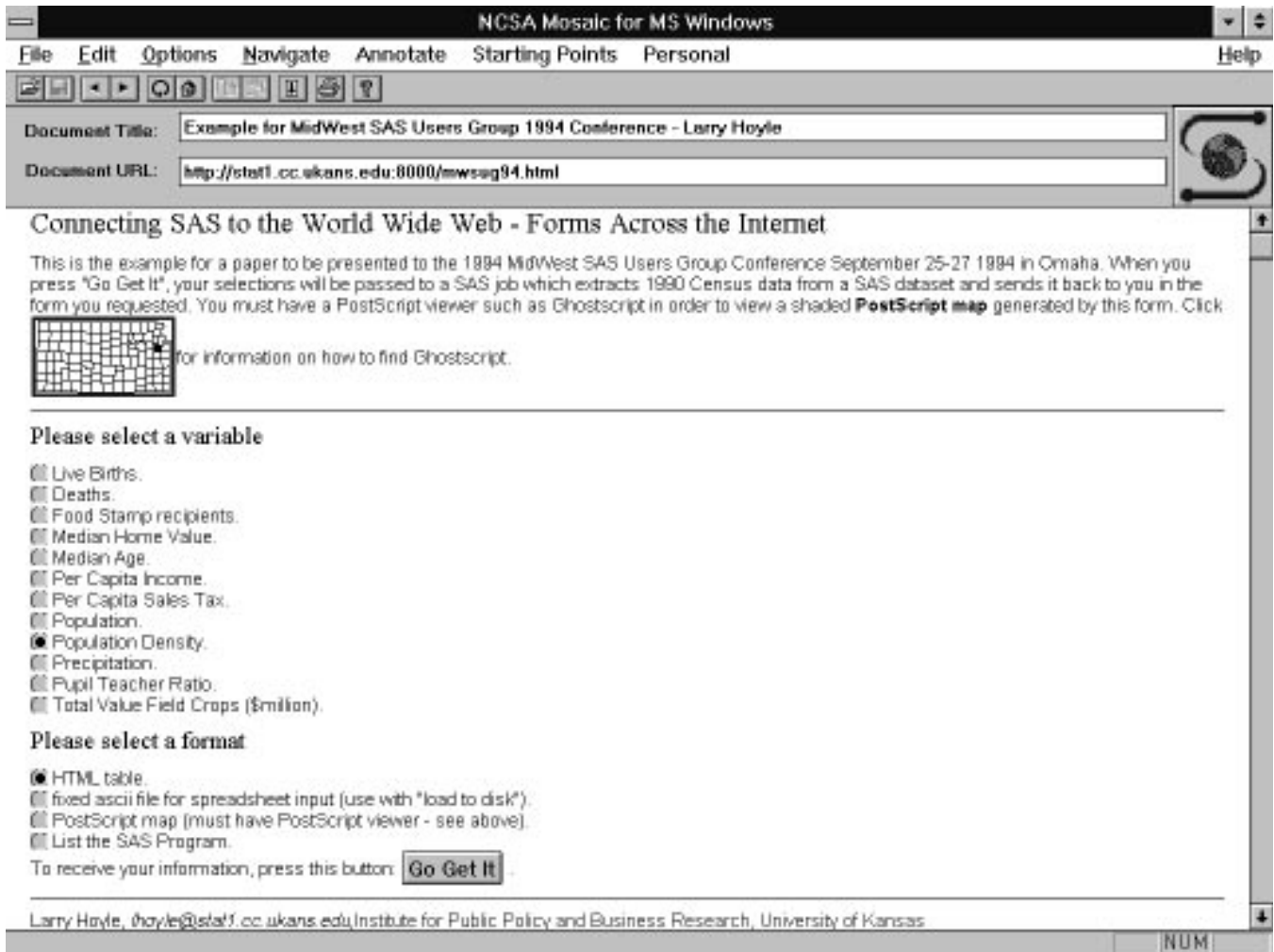


Figure 2, the form as rendered by NCSA Windows Mosaic 2.0 Alpha Release 2

About WWW - <http://info.cern.ch/hypertext/WWW/TheProject.html>
 WWW Home page - <http://info.cern.ch/>
 WWW History - <http://info.cern.ch/hypertext/WWW/History.html>
 HTML Primer - <http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html>
 Forms Overview - <http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html>
 Gopher FAQ - [gopher://gopher.uiuc.edu:70/00/Gopher/FAQ](http://gopher.uiuc.edu:70/00/Gopher/FAQ)
 Overview of httpd - <http://hoohoo.ncsa.uiuc.edu/docs/Overview.html>
 Mosaic for Windows home page - <http://www.ncsa.uiuc.edu/SDG/Software/WinMosaic/HomePage.html>

This is a list of URLs that might be useful in getting started with HTML forms.

One entry in each list also contains the attribute *CHECKED* which makes it the default selection. The form in figure 1 will send a data stream to the server containing *sasvar=Popdense* and *request=table* if the user selects no radio buttons.

The Data Stream

The stream of data sent to the server is one long string of printable ASCII characters. The string contains no line breaks but does contain a combination of three delimiters which are:

- & ends a name/value pair. Note that the last pair may not be followed by an &.
- = separates a name from a value.
- % an escape for non printable characters. The % will be followed by two hexadecimal digits. Note that even the ASCII space character is escaped. Space characters will appear as %20.

The data stream for the default form from figure 1 will look like:

```
sasvar=Popdense&request=table
```

with no carriage returns, line feeds, or eof mark.

Launching the Form Server

When a form is submitted the client program sends a message to the daemon (httpd) already running on the server. The daemon receives this message and launches the application specified in the *ACTION=* attribute of the *<FORM* tag. The shell in which the application is launched has no *PATH* or *HOME* defined. It does have three environment variables defined:

REQUEST_METHOD, *CONTENT_TYPE*, and *CONTENT_LENGTH*. Of these *CONTENT_LENGTH* is the most important for our application. Since the data stream piped to the application by httpd has no end of file indicator, the SAS program must use *CONTENT_LENGTH* to terminate reading.

The SAS Program

Figure 3 shows the shell script which launches SAS on the IBM RS6000 (under AIX) to run this SAS program. The pipe of the *cat -u* was necessary to avoid what seems to be a buffering problem. There may be better solutions.

```
#!/bin/sh
# this script runs the SAS program mwsug94.sas
# when launched from httpd
# HOME and PATH are not defined
# 3/25/94 Larry Hoyle <lhoyle@stat1.cc.ukans.edu>

HOME=where/the/heart/is; export HOME
PATH=$HOME/public_html/cgi-bin:$PATH
PATH=$HOME/public_html:$PATH

cd $HOME/public_html/cgi-bin

#
# SAS startup shell script modified from one by:
# 11/18/92 Wes Hubert <wes@kuhub.cc.ukans.edu>

# Modify the following line to point to the location of SAS
SAS_ROOT=/homeb/sas609/sas609
export SAS_ROOT

# Search SAS directory first
PATH=$SAS_ROOT:$PATH ; export PATH

# SAS requires its own TERMINFO files
TERMINFO=$SAS_ROOT/terminfo
export TERMINFO
TERMINFOADD=$SAS_ROOT/terminfo
export TERMINFOADD

# Use absolute path for SAS executable
cat -u | $SAS_ROOT/sas "mwsug94.sas"
```

Figure 3, the shell script "cgisas_mwsug94"

Figure 4 shows the portion of the SAS program which reads the environment variables and creates macro variables with those values.

```
%global rqmeth;
%let rqmeth=%sysget(REQUEST_METHOD);

%global cnttyp;
%let cnttyp=%sysget(CONTENT_TYPE);

%global cntlen;
%let cntlen=%sysget(CONTENT_LENGTH);
```

Figure 4, reading the environment variables

```

%global maxlen; /* max length of name or value string */
%let maxlen=200;
%global sasvar; /* SAS variable to be displayed */
%global saslbl; /* label of the variable to be displayed */
%global request; /* type of request eg. "map" or "table" */

data namevals;
  length name $ &maxlen value $ &maxlen hexstr $ 2;
  array s{2} name value;
  keep name value;
  length c $ 1;
  retain ixc name value;
  keep name value;
  infile stdin lrecl=1 recfm=f;

  ixc=1; name=' '; value=' ';
  do ixc=1 to &cntlen;
    input c $char1. @@; put c=;

    select(c);
      when('&')do;
        name=left(name); value=left(value);
        output;
        put value=;
        ixc=1; name=' '; value=' ';
        end; /* when '&' */

      when('=')do;
        put name=;
        ixc=2;
        end; /* when '=' */

      when('%')do;
        input c $CHAR1.;
        substr(hexstr,1,1)=c;
        input c $CHAR1.;
        substr(hexstr,2,1)=c;
        c=input(hexstr,$HEX2.);
        s(ixc)=trim(s(ixc))||c;
        ixc=ixc+2;
        if ixc gt &cntlen then do;
          /* if ixc increments past &cntlen -- big trouble*/
          put 'index loop overrun';
          abort;
        end; /* if ixc */
        end; /* when '%' */

      otherwise do;
        s(ixc)=trim(s(ixc))||c;
        end; /* otherwise */

    end; /* select(c) */
  end; /* do ixc */

  /* the last name value pair has no trailing & */
  name=left(name); value=left(value); output;
  put value=;
  stop; /* no more input after loop */

```

Figure 5, creating a dataset from the form's data

Figure 5 shows the portion of the SAS application which creates a SAS dataset containing the NAME=VALUE pairs from the data stream from the client which rendered the form. This DATA step can be used in other applications processing a form. Note that the data are read from *stdin* with a *lrecl* of 1 and *recfm* of *f*. The data step ends with a *STOP*, so that the data are read by the *DO* loop which reads *&cntlen* characters. The macro variable *cntlen* was set from the environment variable as shown in figure 4.

Within the *DO* loop a *SELECT* statement looks for the delimiters. An array is used to switch back and forth between appending to the name and value variables. The final code outside the loop catches the last pair, which are not terminated by an *&*.

Figure 6 shows the creation of the macro variables *sasvar*, *request* and *saslbl*. *Sasvar* contains the name of the variable from the SAS dataset to be processed and *saslbl* its label. *Request* indicates the form in which the variable is to be displayed. These are used later in the application to select the form of output to be returned to the client.

```

data _null_;
  set namevals;

  select(name);
    when('sasvar')do;
      call symput(name,trim(value));
    end;

    when('request')do;
      call symput(name,trim(value));
    end;
  end; /* select */
data _null_;
  set demo;
  length lbl $ 40;
  call label(&sasvar,lbl);
  call symput('saslbl',trim(lbl));

```

Figure 6, macro variables *sasvar*, *request*, and *saslbl*

Figure 7 shows the portion of the SAS application which creates the HTML document

containing the table of data requested. This document is written by SAS to STDOUT which is routed through httpd back to the client. This stream of data begins with the string *content type: text/HTML* followed by an empty line. This tells the client to treat the rest of the data as HTML and try to display it.

```
%macro reply;

%if %upcase(&request)=TABLE %then
%do; /* table section */

  data _null_;
  put 'Executing table section';

  proc sort data=demo; by county;

  data _null_;
  file STDOUT line=curline col=curcol
    pagesize=50 n=ps;
  set demo end=last;
  retain firstlin c;
  if _n_ = 1 then do;
    put 'content-type: text/HTML';
    put;
    put '<' 'h1>';
    put "&saslabl";
    put '<' '/h1>';
    put '<' 'pre>';
    firstlin=curline;          /* this is the first line of the table */
    c=1;                       /* this is the column in which to print */
  end;

  if curline ge (firstlin + 35) then do;
    put #(firstlin) @;
    c=c+25;
  end;                               /* if curlin */

  put @c cntynm @(c+14) &sasvar best9.;

  if last then put @1 #(firstlin + 36) '<' '/pre>';
  run;
%end;                               /* table section */
```

Figure 7, creating an HTML reply

The reply created in figure 7 contains the label of the selected variable in an H1 heading followed by a <PRE> tag. This tag says that what follows is pre-formatted text. The client application will display it in a fixed spaced font and will not justify it. The data are output in three columns using SAS pointer controls. The <PRE> tag is also useful for something like a

program listing where indentation is important, since by default text in HTML files is justified.

Other content types might be a graphics file, a sound file, a movie file, or whatever. The client can launch a viewer (also called a helper) application to present the file. This is how the sample SAS application can display a PostScript graphic.

Figure 8 shows the SAS code needed to send a PostScript graphic back to the client. The code to generate the map will not be shown here. Any SAS/GRAPH procedure would be able to use this technique.

```
data _null_;
file STDOUT;
put 'content-type: application/postscript';
put;

goptions device=ps gsfname=STDOUT
  gsfmode=append nodisplay gaccess=sasgaedt
  gsflen=80 penmounts=255 characters
  chartype=9;

/* 9 is helvetica, 11 helv bold*/
```

Figure 8, to return a PostScript graphic under UNIX

Getting Started

To get started with WWW forms you will need httpd and a client such as Mosaic. The httpd binaries are available for anonymous ftp from the directory:
 ftp.ncsa.uiuc.edu/Web/ncsa_httpd/current
 Mosaic for Windows is available from:
 ftp.ncsa.uiuc.edu/PC/Mosaic/wmos20a4.zip
 you may also need the file win32s.zip.

SAS, and SAS/GRAPH are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. Other brand and product names are registered trademarks or trademarks of their respective companies.

Contact Information

Larry Hoyle email:lhoyle@stat1.cc.ukans.edu
 IPPBR voice:(913) 864-3701
 607 Blake Hall fax:(913) 864-3683
 University of Kansas
 Lawrence, KS, 66045-2960