

## A Hands-on Introduction to Creating Dynamic Web Pages

Mickey Waxman, Academic Computing, University of Kansas

Larry Hoyle, IPPBR, University of Kansas

### ABSTRACT

This hands-on workshop will give a basic introduction to using SAS Application Dispatcher to create World Wide Web pages on demand. Workshop participants will learn how to create an HTML form which will invoke a SAS program on a server via the Application Dispatcher. They will also learn how to construct a SAS program which produces web pages upon request from such an HTML form.

No previous knowledge of HTML is required.

### Overview

Consider the following transaction. Jane Doe fires up her Web browser and browses her way to my Web page. Looking at my Web page she finds she can request further information on a variety of topics. She makes a few selections, and clicks on the Submit button. Then SAS on my server wakes up, processes a set of data and sends a bar chart back to her browser showing the information she requested.

What did I have to do to make this possible? Two things, essentially: (1) create a Web page that contains a form that references my SAS program and (2) write the SAS program that collaborates with this Web page. This workshop teaches the basics on how to create these two components.

Figure 1. Jane's Web browser sends a request to my Web server asking for a copy of my Web page. This HTML file is duly sent to Jane's computer where her browser interprets the HTML and displays my Web page. In her browser Jane selects options and enters data in the HTML form specifying what further information she is requesting. Then she clicks on the Submit button.

Figure 2. Jane's browser assembles her selections into a message, which is sent to my Web server. My Web server sees that the request is addressed to my Application Broker, starts the Broker, a CGI program included in the SAS/IntrNet package, and passes it the message from Jane's browser. The Application Broker interprets and processes the request data and then establishes contact with my Application Server. The Application Server, a SAS session waiting for action, processes the input from the Broker, creates a fileref named `_webout`, which points back through the Broker to Jane's browser, and creates a set of macro variables containing the request data. The Application Server then finds the specified SAS program that I wrote and runs it with those predefined macro variables. The macro variables are what convey information from Jane's browser to my SAS program. Output from this SAS program is written to `_webout`, which streams the output to the Application Broker. The broker does a little processing of the output before passing it to the Web server. The Web server then sends the output to Jane's browser.

That's probably more detail than care to know, but someday you'll thank us for this.

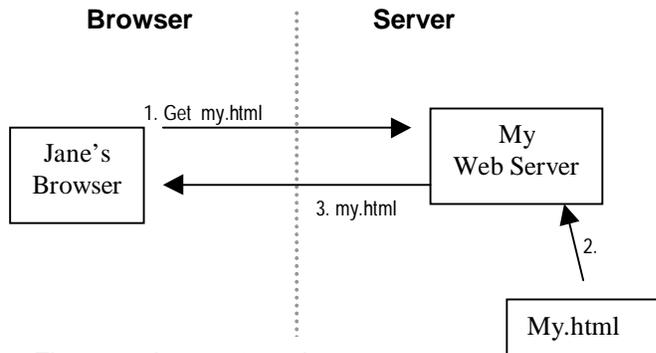


Figure 1, Jane gets web page

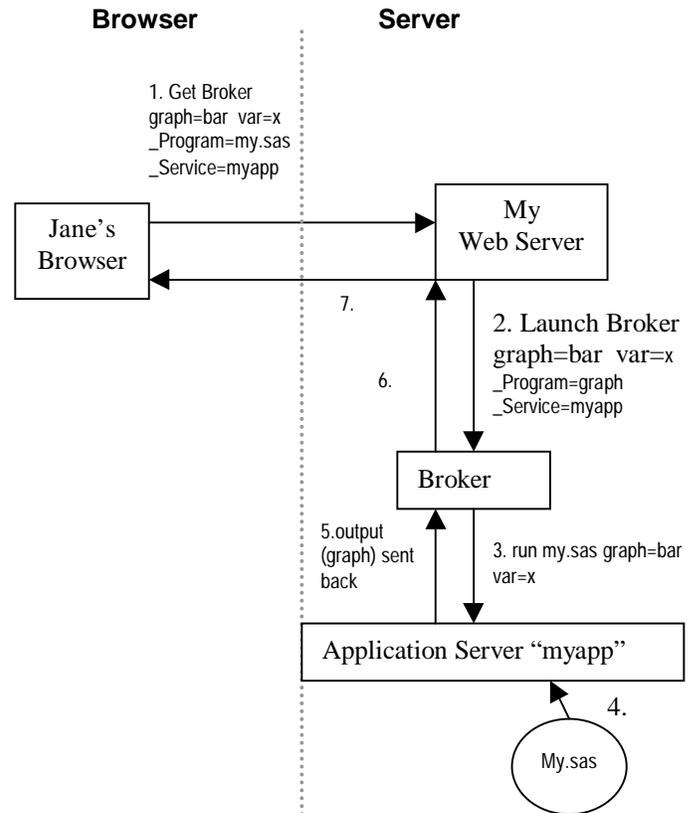


Figure 2, Jane runs my SAS program

## Example Applications

This workshop will use five example applications, each of which has an HTML file and a SAS program. Thumbnails of what the user sees when using the applications are shown on this page, with the HTML form on the left and the output from the SAS program, as delivered back to the browser, on the right.

The first example, in the first row, is a simple "Howdy World" program. It also displays the time to show that an application with no input values can deliver dynamic information.

The second application prints back the value of an input parameter.

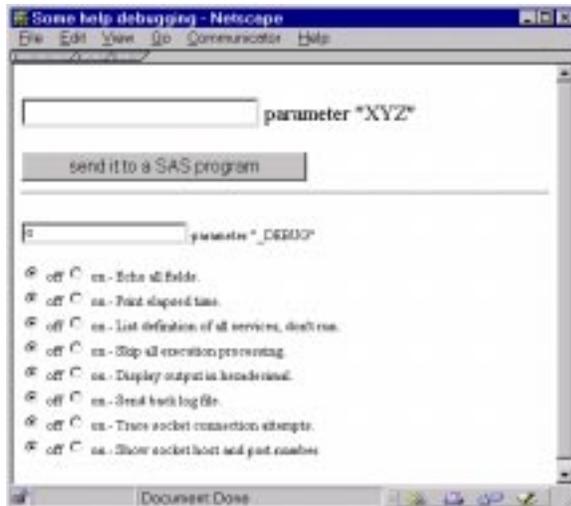
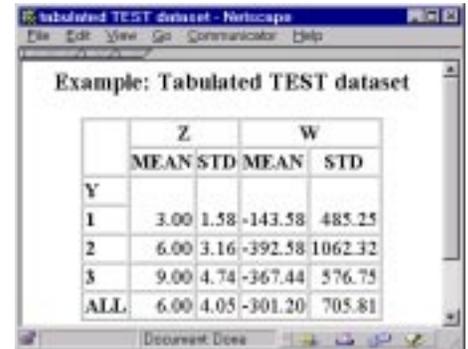
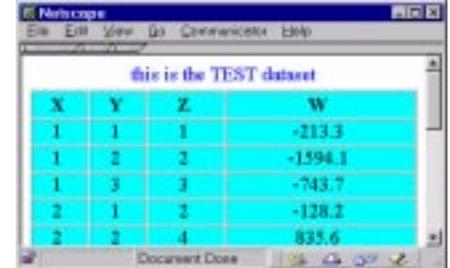
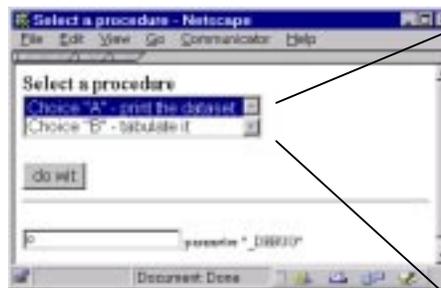
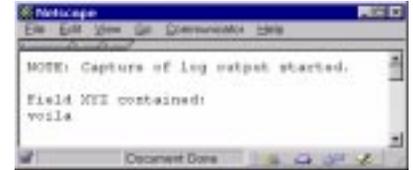
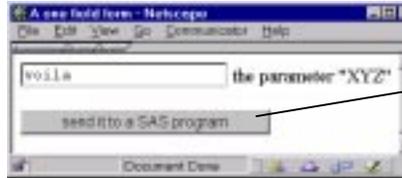
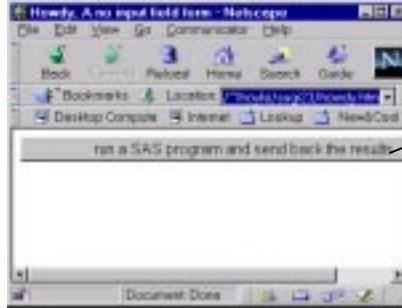
The third application sends back the value of the input field in a graphic image file (GIF).

The fourth application uses an input parameter to choose between two procedures.

The final application shows some debugging tools.

## HTML Form

## Results from SAS



## Howdy World

This Dispatcher application runs a SAS program that prints a message and the time. This is a dynamic application in that it produces a different page each time it is run, but it requires no input parameters from the user.

In order for the SAS program to be run, the HTML file which requests it must contain three references: a pointer to SAS Institute supplied program named "Broker", the name of a broker service, and a reference to the SAS program to be run.

The pointer to Broker appears in the "action=" clause of the HTML "FORM" tag. Broker itself is an executable which your system administrator configures to have one or more "services". Each service corresponds to one or more SAS sessions running on a server.

The name of the service to be used appears as the value of a field named "\_SERVICE" in the HTML form. It can be user selectable or it can appear in a "hidden" field as it does in figure 3. In this case the value "wkshp124" does not appear on screen for the user since the type of the INPUT field is "hidden". It is, however, passed to the server as "\_SERVICE=wkshp124" when the form is submitted.

The name of the SAS program to run in figure3 is "howdy.sas". Its location is pointed to by the libname "clientXX". This reference is passed to the server as "\_PROGRAM=clientXX.howdy.sas". Dispatcher applications may also run source programs, SCL code, or macros in SAS catalog entries,

```
<HTML>
<HEAD>
<TITLE>Howdy, A no input field form</TITLE>
</HEAD>

<BODY>

<FORM action='http://brokerpath/broker'>

<INPUT type=hidden
name=_SERVICE
value=wkshp124>

<INPUT type=hidden
name=_PROGRAM
value=clientXX.howdy.sas>

<INPUT type=submit
value="run a SAS program ">
</FORM>

</BODY>

</HTML>
```

figure 3, howdy.htm

The SAS program, clientXX.howdy.sas, which is referenced by the "\_PROGRAM" field above, is shown in figure 4. The important features of this program are the reference to the output fileref "\_webout", and the HTTP header which is the first output sent to \_webout.

When Broker sets up for howdy.sas to be run, it inserts a fileref to \_webout at the beginning of the code to be run. Anything written to this fileref ends up being sent back to the browser from which the form in figure 3 was submitted.

In order for the browser to interpret this output correctly, it must be preceded by a "Hypertext Transfer Protocol (HTTP)" header. This header must end with a null line. Note the "/" at the end of the highlighted "put" statement in figure 4. The "/" is necessary in that it generates the needed null line. The HTTP header in this example, 'Content-type: text/html', informs the browser that what follows is to be interpreted as HTML. While many browsers will display output without any HTML codes, it is a good idea to include at least the required elements: <HTML></HTML>, <HEAD></HEAD>, and <BODY></BODY>.

```
/* ----- */
/* howdy.sas - a hello world program */
/* for trying the application dispatcher. */
/* ----- */

data _null_;

file _webout;

t=time();
d=date();

put 'Content-type: text/html';

put '<H1>Howdy, a SAS program wrote this on:</H1>';
put d date7. ' at: ' t time. ;

run;
```

figure 4, clientXX.howdy.sas

## A Form With One Input Field

The second example, shown in figure 5, adds a field to the form into which a user can enter information. In this particular case the field has a name of "XYZ", and has no default value. If a user were to enter the value "13" into the field, the browser would send "XYZ=13" to Broker when the form was submitted. A separate *name=value* pair is sent for each field entered in the form.

```

/* ----- */
/* 1field.sas - Reads one field from the browser (form) */
/* ----- */

%global XYZ;

options nosource nonotes;
/* ----- */
/* send the log window to the browser */
/* ----- */

%out2htm(capture=on, window=log );

data _null_;

sg=symget('XYZ');

put "Field XYZ contained: " / sg ;

run;

%out2htm(htmlfref=_webout,
capture=off,
window=log,
runmode=s,
openmode=replace);

```

figure 6, clientXX.1field.sas

```

%macro noexit(myname);
data _null_;
m = symget("&myname");
n = translate(trim(m),'_____', '%&'()';,');
call symput("&myname",n);
run;
%mend noexit;

```

figure 7, a macro to remove nasty characters

```

<HTML>
<HEAD>
<TITLE>A one field form</TITLE>
</HEAD>
<BODY>

<FORM action='http://brokerpath/broker'>
<INPUT type=hidden
name=_SERVICE
value=wkshp124>

<INPUT type=hidden
name=_PROGRAM
value=clientXX.1field.sas>

<INPUT type=text
name=XYZ> the parameter "XYZ"<P>

<INPUT type=submit
value="send it to a SAS program">

</FORM>
</BODY>
</HTML>

```

figure 5, 1field.htm

The SAS program clientXX.1field.sas, shown in figure 6, receives the value "XYZ" as a predefined macro variable. Broker sets up this macro variable automatically for each name=value pair it receives. It is a good idea, though, to have a %global statement for each parameter the SAS program references. This will force the creation of an empty macro variable in case no name=value pair was sent.

An Application Dispatcher program can reference a macro variable containing a form parameter (e.g. XYZ) as: "&XYZ";  
or  
%superq(XYZ);  
or  
with symget('XYZ');  
Parameters are also available to an Application Dispatcher SCL program via an SCL list.

Using the "&XYZ" method, however, is a security risk. It can be used to insert undesired SAS code into your application. The symget method is safest. Figure 7 shows a sample macro to strip out dangerous characters from the form fields.

Figure 6 also shows the use of the "%out2html" macro to format SAS output as HTML. Details of the HTML formatting tools will be presented in another workshop.

Note, though, that this example doesn't explicitly write an HTTP header, unlike the clientXX.howdy.sas example. This is because the "runmode=s" option of the %out2html macro indicates that the application is running in "server" mode. When both "runmode=s" and "openmode=replace" are specified, an HTTP header is automatically written.

## Creating a Graphic

The next example has a form which has 2 user editable fields. The second field is a "select box" which allows the user to select from a list of options. When the form is submitted, the browser receives a graphic image (a GIF file).

```

/* ----- */
/* slide.sas - show the input as a GIF file */
/* ----- */

%global XYZ TCOLOR;

/* these statments can be used to test without the
application dispatcher
filename _webout 'd:\inetpub\wwwroot\sugi23\grphout.gif';
%let XYZ=testme;
*/

/* ----- */
/* generate an HTTP header for a GIF file */
/* don't run this data step if testing locally */
/* ----- */

data _null_;
file _webout;
put 'Content-type: image/gif';

run;

/* ----- */
/* use a GIF graphics device */
/* ----- */

options device=gif160
gsfmode=replace
gsfname=_webout;

/* ----- */
/* set the color from an input parameter */
/* ----- */

options ctitle=%superQ(TCOLOR);

/* ----- */
/* generate the graphic */
/* ----- */

proc gslide frame;
note h=10 move=(8,15) f=brush %superQ(XYZ);

run;

```

figure 9, clientXX.slide.sas

```

<HTML>
<HEAD>
<TITLE>Make a GIF file</TITLE>
</HEAD>
<BODY>
<FORM action='http://brokerpath/broker'>
  <INPUT type=hidden
        name=_SERVICE value=wkshp124>
  <INPUT type=hidden
        name=_PROGRAM
        value=clientXX.slide.sas>

  The text for the graphic<BR>
  <INPUT type=text name=XYZ>

  <P>The text color<BR>
  <SELECT name=tcolor size=4>
    <OPTION value="blue" SELECTED>Blue </OPTION>
    <OPTION value="red">ruby red</OPTION>
    <OPTION value="green">oz green</OPTION>
    <OPTION value="black">plain old black</OPTION>
  </SELECT>

  <P><INPUT type=submit value="send me a GIF file">
</FORM>
</BODY>
</HTML>

```

figure 8, slide.htm

The SAS program clientXX.slide.sas first writes an HTTP header to tell the browser that a GIF graphic follows. That header is **'Content-type: image/gif'**. Note that the SAS program should write no other text to \_webout, otherwise the browser would see a corrupted gif file.

Selecting one of the GIF drivers creates the graphic file. In figure 9 the driver is "gif160" which produces an image 160 pixels across. Specifying the *gsfname=\_webout* graphic option results in the graphic being sent back to the browser. The *gsfmode=replace* option causes the driver to write a proper internal header for the graphic.

The actual creation of the graphic can be done by a number of components of the SAS system. In figure 9, PROC GSLIDE is used to create a slide from the contents of the XYZ field.

## Selecting Code

The Dispatcher program clientXX.iffy.sas is shown in the 4 part figure 10. This application shows the use of an input field to select code to be executed.

The initial section in figure 10a creates a test dataset.

Figure 10d contains the code which actually makes the selection. If the value of the input field is "A", then the macro "CHOICEA" is invoked. Input of "B" causes "CHOICEB" to be invoked. Any other input invokes "OOPS".

Figure 10b contains the macro definition for CHOICEA. It uses the "%ds2htm" macro to send the browser the test dataset as an HTML table. Figure 10c contains the definitions for CHOICEB and OOPS. ChoiceB uses the tab2htm macro to send back an HTML table from a PROC TABULATE.

```

/* ----- */
/* iffy.sas - select logic based on a form */
/* ----- */

%global CHOICE;

options mprint;

/* ----- */
/* make a test dataset */
/* ----- */

data test;
do x=1 to 5;
do y=1 to 3;
z=x*y;
w=round(1000*rannor(1213131),.1);
output;
end;
end;
run;

```

figure 10a, part 1 of clientXX.iffy.sas

```

/* ----- */
/* first choice - dataset to HTML formatting tool */
/* ----- */
%macro CHOICEA;

%ds2htm(data=test,
runmode=s,
openmode=replace,
htmlfref=_webout,
caption=this is the TEST dataset,
ccolor=blue,
tbbgcolr=cyan);

%mend CHOICEA;

```

figure 10b, part 2 of clientXX.iffy.sas

```

/* ----- */
/* the second choice - tabulate to HTML formatting tool */
/* ----- */
%macro CHOICEB;

%tab2htm(capture=on);

options linesize=96 pagesize=54 nocenter nodate nonumber;
title 'Example: Tabulated TEST dataset';
proc tabulate data=WORK.TEST
formchar=82838485868788898a8b8c;x;
table Y ALL , (Z W) * ( 'MEAN' 'STD' );
var Z W ;
class Y ;
run;

%tab2htm(capture=off,
runmode=s,
openmode=replace,
htmlfref=_webout,
brtitle=tabulated TEST dataset,
center=Y);

%mend CHOICEB;

/* ----- */
/* this prints out an error message */
/* ----- */
%macro OOPS;
data _null_;
file _webout;
pick=symget('CHOICE');
put 'Content-type: text/html// 'Unknown choice:/' pick;
run;
%mend OOPS;

```

figure 10c, part 3 of clientXX.iffy.sas

```

/* code selection - macro PICKONE invokes */
/* either the macro "CHOICEA" or "CHOICEB" */

%macro PICKONE;

%IF %upcase(%superq(CHOICE))=A %THEN %DO ;
%CHOICEA;
%END;
%ELSE %IF %upcase(%superq(CHOICE))=B %THEN
%DO;
%CHOICEB;
%END;
%ELSE %DO;
%OOPS;
%END;

%MEND PICKONE;
%PICKONE;

```

figure 10d, part 4 of clientXX.iffy.sas

## Debugging Tools

A special parameter, “\_DEBUG” is interpreted by Broker. The value of \_DEBUG is the sum of a number of powers of 2. If \_DEBUG includes a “1” in the sum, then Broker will echo all of the fields sent from the client’s form. If it contains a “2”, Broker will send back the time.

Suppose, for example, Broker receives \_DEBUG=3. Then it will echo all fields and send back the time.

Figure 11b contains a form and an associated JavaScript script which allows you to select components of the \_DEBUG field with radio buttons. These buttons appear in pairs with the same field name, e.g. df2. The browser will allow only one button of each pair to be selected at a time. If you turn a parameter “on” the off button is deselected automatically.

Note that the form which sends parameters Broker must have a “Name=f” clause, and an \_DEBUG field for this script to work. Figure 11a and 11b together are an example HTML file using the \_DEBUG field. The associated SAS program “W\_debug.sas” is not shown.

```
<HTML>
<HEAD>
<TITLE>Some help debugging</TITLE>
</HEAD>
<BODY>
<FORM name=f action='http://brokerpath/broker'>
<INPUT type=hidden name=_SERVICE
value=wrkshp124>
<INPUT type=hidden name=_PROGRAM
value=clientXX.W_debug.sas>
<BR>
<INPUT type=text name=XYZ> parameter "XYZ"
<P>
<INPUT type=submit
value="send it to a SAS program">
<HR>
<FONT size=-2><P>
<INPUT type=text name=_DEBUG
value=0> parameter "_DEBUG"
</FORM>
```

figure11a, part 1 of W\_debug.htm

```
<SCRIPT Language="JavaScript">
// This function, together with the FORM which follows
// sets the _DEBUG variable in the preceding form.
// The form with _DEBUG must have a "name=f" parameter
// in its "FORM" tag
function comp_debug(){
with(document.dbf){
newdb=0;
if(_df1[1].checked) newdb=newdb+parseInt(_df1[1].value);
if(_df2[1].checked) newdb=newdb+parseInt(_df2[1].value);
if(_df4[1].checked) newdb=newdb+parseInt(_df4[1].value);
if(_df8[1].checked) newdb=newdb+parseInt(_df8[1].value);
if(_df16[1].checked) newdb=newdb+parseInt(_df16[1].value);
if(_df128[1].checked)newdb=newdb+parseInt(_df128[1].value);
if(_df256[1].checked)newdb=newdb+parseInt(_df256[1].value);
if(_df512[1].checked)newdb=newdb+parseInt(_df512[1].value);
document.f._DEBUG.value = newdb;
}
} // ends comp_debug
</SCRIPT>

<FORM name=dbf>
<INPUT type=radio name=_df1 value=0 checked
onClick="comp_debug();">off
<INPUT type=radio name=_df1 value=1
onClick="comp_debug();">on - Echo all fields.

<INPUT type=radio name=_df2 value=0 checked
onClick="comp_debug();">off
<INPUT type=radio name=_df2 value=2
onClick="comp_debug();">on - Print elapsed time.

<INPUT type=radio name=_df4 value=0 checked
onClick="comp_debug();">off
<INPUT type=radio name=_df4 value=4
onClick="comp_debug();">on - List definition of all services, don't run.
<INPUT type=radio name=_df8 value=0 checked
onClick="comp_debug();">off
<INPUT type=radio name=_df8 value=8
onClick="comp_debug();">on - Skip all execution processing.
<INPUT type=radio name=_df16 value=0 checked
onClick="comp_debug();">off
<INPUT type=radio name=_df16 value=16
onClick="comp_debug();">on - Display output in hexadecimal.
<INPUT type=radio name=_df128 value=0 checked
onClick="comp_debug();">off
<INPUT type=radio name=_df128 value=128
onClick="comp_debug();">on - Send back log file.
<INPUT type=radio name=_df256 value=0 checked
onClick="comp_debug();">off
<INPUT type=radio name=_df256 value=256
onClick="comp_debug();">on - Trace socket connection attempts.
<INPUT type=radio name=_df512 value=0 checked
onClick="comp_debug();">off
<INPUT type=radio name=_df512 value=512
onClick="comp_debug();">on - Show socket host and port number.
</FONT>
</FORM>
</BODY>
</HTML>
```

figure11b, part 2 of W\_debug.htm

## Errors You Might See

Figure 12a contains some error messages you might see when debugging an Application Dispatcher application. The condition under which you might see the error is shown in boldface, and the error message Broker sends back is shown below that.

In example 1 the Uniform Resource Locator (URL) listed in the "action=" clause of the form tag had a typo in the directory portion of the path to Broker. You might see other messages if the typo is in the name of the server.

In example 2 the libname portion of the `_PROGRAM` field was misspelled.

In example 3 the program name portion of the `_PROGRAM` field was misspelled.

In example 4 the program type portion of the `_PROGRAM` field was misspelled.

In example 5 the service in the `_SERVICE` field is misspelled.

Example 6 happened when the SAS program failed to write anything to `_webout`. This could have been due to a missing "FILE" statement, or an incorrect parameter in the `"%out2html"` macro.

Example 7 happened when Broker detected an incorrect HTTP header. This could be due to a typo or the lack of a null line at the end of the header.

In example 8 the `"%out2html"` macro contains an `"htmlfile"` parameter. This causes the output to go to an actual file, not the `_webout` fileref. The solution is to use the `htmlfref` parameter.

The example in figure 12b is a little different. Here the SAS program used the `"&XYZ"` construct in an assignment statement. When the contents of field XYZ is as in the second line of the figure, the **put 'oops'** portion of the macro variable value was executed as SAS code. This illustrates the nature of the security problem using the `"&XYZ"` type construct on unfiltered input. Dangerous characters can be edited out with code like that in figure 7.

Try this value of XYZ with `w_debug.sas`  
`test3";put 'oops';put"`

NOTE: Capture of log output started.

**oops**  
 Field XYZ contained:  
 test3

*Figure 12b, insecurity with "&XYZ"*

### 1 - Bad "Action=URL"

HTTP/1.0 403 Access Forbidden (Execute Access Denied – This Virtual Directory does not allow objects to be executed.)

### 2 - Bad libname

Application Error

The library clientX is not allocated for the current service. Check the spelling of the library name. If it is spelled correctly contact the server administrator and notify him/her of the problem.

### 3 - Bad program name

Application Error

The program clientXX.field.sas does not exist.

### 4 - Bad Program type

Application Error

The program type 1FIELD is invalid.

### 5 - Bad Service

Error in HTML form

The service "wax\_oy" is not listed in the configuration file.

### 6 - No output to \_webout

Error reading SAS output

The SAS program did not produce any output. This could happen if one of the early steps failed. Set `_DEBUG=131` and resubmit in order to see the SAS Log file, or set `_DEBUG=16` to see a hex dump of the output.

### 7 - bad HTTP header

Invalid HTTP header

The SAS program did not produce a valid HTTP header. It must at least have a line like:

Content-type: text/html

followed by a blank line to define the output MIME type.

"Location:" is also allowed. Set `_DEBUG=131` and resubmit in order to see the SAS Log file, or set `_DEBUG=18` to see a hex dump of the output.

### 8 - %out2htm(htmlfile=\_webout,

`capture=off,`

`window=log,`

`runmode=s,`

`openmode=replace);` (you should use `htmlfref`)

Error reading SAS output

The SAS program did not produce any output. This could happen if one of the early steps failed. Set `_DEBUG=131` and resubmit in order to see the SAS Log file, or set `_DEBUG=16` to see a hex dump of the output.

*Figure 12a, error messages you might see*

## Resources and References

The best source for information on using SAS/IntrNet is the SAS Institute World Wide Web site. Click on the Web Enablement link on their home page. The home page is:

SAS Institute Inc., *SAS Home Page*.  
<http://www.sas.com>.

Other references:

The National Center for Supercomputing Applications *NCSA Beginner's Guide to HTML*  
<http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html>

Michael Grobe  
*HTML quick reference*  
[http://www.cc.ukans.edu/~acs/docs/other/HTML\\_quick.shtml](http://www.cc.ukans.edu/~acs/docs/other/HTML_quick.shtml)  
Academic Computing Services  
The University of Kansas

Michael Grobe  
*An instantaneous introduction to CGI scripts and HTML forms*  
<http://www.cc.ukans.edu/~acs/docs/other/forms-intro.shtml>  
Academic Computing Services  
The University of Kansas

SAS/IntrNet is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries.  
® Indicates USA registration.

Mickey Waxman [mickey@ukans.edu](mailto:mickey@ukans.edu)  
Larry Hoyle [lhoyle@ukans.edu](mailto:lhoyle@ukans.edu)