

ColdFusion® and SAS®: No Pain Meds Required

Carol Martell, UNC Highway Safety Research Center, Chapel Hill, NC

ABSTRACT

This paper describes a method of integrating dynamic SAS output into a Macromedia ColdFusion® application. The SAS/IntrNet Application Dispatcher responds to a request from the ColdFusion server, and the results are displayed in a ColdFusion page having the .cfm extension. The advantages to using this approach include isolating the SAS programmer from the design aspects of Web development.

The technique requires that SAS/IntrNet Application Dispatcher be available with the accompanying SAS/IntrNet Application Server, and that the user have access to a ColdFusion Server. The paper does not address setting up or configuring either Application Dispatcher or ColdFusion Server. The intended audience is Web application developers with experience using Application Dispatcher and with the desire to stop using PUT statements for enhancing Web design.

INTRODUCTION

The Web as we see it today is delivered to us through the use of a myriad of software applications. Information is delivered, collected, combined, refined and generally bandied about in a most amazing way. The competition for our attention as Web site visitors has created a new industry with new job titles containing the word 'Web'. With evolving Web site complexity, industry software tools evolved from basic HTML text editors to Web design packages that insulate the user from the actual markup language and Web designers evolved into Web developers. Static content evolved to dynamic content, creating a demand for Web application servers. An early application server offering for creating dynamic Web content, ColdFusion is no longer in its infancy and enjoys widespread use. This paper is written for SAS/IntrNet Developers who find themselves in a work setting where ColdFusion is available.

WHY BOTHER?

What is the benefit of integrating SAS with ColdFusion? The code behind an attractive and interesting Web page can be copious. Choosing 'View Source' from your favorite browser illustrates this point. In order for the results from a SAS program called through the Application Dispatcher to conform with the design of a Web site, that copious code must somehow be included. While it is possible to write SAS code that will create the entire page as output, it is no fun at all.

A common approach is to sandwich the SAS program between two null datasteps. The first datastep writes all the window dressing code that must precede the SAS output, and the second datastep writes the code that finishes dressing the page. The actual program uses ODS HTML, with directives to omit HTML header and footer code.

```
DATA _NULL_;
FILE _WEBOUT;
- put statements for beginning of page -
RUN;
ODS HTML FILE=_WEBOUT(NO_TOP_MATTER NO_BOTTOM_MATTER);
- SAS procedure(s) -
ODS HTML CLOSE;
DATA _NULL_;
FILE _WEBOUT;
- put statements for end of page -
RUN;
```

Why is the above no fun? The copious window dressing code contains quotation marks – both double and single. Consider this one anchor tag of Web page source code:

```
<a onmouseover="doit('men1','visible')" onMouseOut="doit('men1','hidden')"
href="javascript:void(0)">
```

The SAS PUT statements required to write the tag above follow, with each literal string on a separate line:

```
PUT
'<a onmouseover="doit('
'"men1','visible'"
')" onMouseOut="doit('
'"men1','hidden'"
')" href="javascript:void(0)">';
```

Alternatively, one can use two single quotes in a row to represent a single quote within the literal:

```
PUT
'<a onmouseover="doit(''men1'', ''visible'')" onMouseOut="doit(''men1'', ''hidden'')"
```

```
href="javascript:void(0)"> ';
```

The first approach makes it easier to find the individual literal strings; the second more closely resembles the desired result. Either one will make you go cross-eyed.

An alternative to writing literal strings is to place the desired code into two files, one to prepend, the other to append. The SAS output can be sandwiched between these files using null datasteps. The files are streamed to _WEBOUT using INPUT and PUT _INFILE_:

```
DATA _NULL_;
FILE _WEBOUT;
INFILE 'prependfile' lrecl=xx;
INPUT;
PUT _INFILE_;
RUN;
ODS HTML FILE=_WEBOUT(NO_TOP_MATTER NO_BOTTOM_MATTER);
- SAS procedure(s) -
ODS HTML CLOSE;
DATA _NULL_;
FILE _WEBOUT;
INFILE 'appendfile' lrecl=yy;
INPUT;
PUT _INFILE_;
RUN;
```

Given the luxury of being able to modify the configuration file for the Application Broker, the sandwich effect can again be achieved. The files can be attached to the output of every program executed by the Broker, or by a specific service defined to the Broker by using one of the following pairs of configuration file directives:

```
PrependFile filepath
AppendFile filepath
ServicePrependFile filepath
ServiceAppendFile filepath
```

In addition to being difficult to code in the first place, the window dressings – the images, the thematic look, the layout, even the navigational aspects – are often a moving target. Because development of the SAS program is often concurrent with development of the entire Web site, window dressing changes can occur frequently, providing an opportunity to again have no fun at all. By divorcing the SAS program from the window dressing, it is possible to avoid rewriting the SAS program every time the look of the site is tweaked or a new navigational component is added.

SOLUTION

The solution offered here is to allow ColdFusion to have all the window dressing code and let SAS be responsible for analyzing data or otherwise presenting information. ColdFusion uses customized markup tags to provide enhanced functionality in a Web page. One of these tags, the <cfhttp> tag, sends out an HTTP request and holds the response as a variable. Normally, a request sent to a SAS Application Broker returns the results of a SAS program directly to the browser window. When the same request is sent using the <cfhttp> tag, those results are intercepted by ColdFusion rather than being streamed directly to the browser window. The SAS results can then be displayed in a 'dressed' browser window. The containing page is a ColdFusion page, which remains part of an overall ColdFusion Application, a point which carries weight with ColdFusion development. As we will see, the SAS results can also be tweaked prior to display.

The examples that follow represent neither sophisticated SAS code nor sophisticated ColdFusion code. Few style modifications are illustrated to enhance the combined application results. Because only SAS Version 8 Broker is available to the author at the time of this writing, the style 'minimal' is used whenever ODS HTML is employed. Version 9 of SAS provides many more style offerings that could prove useful when invoking SAS from ColdFusion.

CALLING SAS: THE <CFHTTP> TAG

This tag can have quite a few parameters, but the minimal set for these purposes would point to the SAS Application Broker and specify a method. A closing tag is required:

```
<cfhttp method="post" url = "server_and_broker_path">
</cfhttp>
```

PASSING PARAMETERS: THE <CFHTTTPARAM> TAG

One piece of information required by the Broker but missing from the <cfhttp> tag above is the name of the requested SAS program. Name/value pairs to submit with the <cfhttp> tag must be supplied in <cfhttpparam> tags. Whenever name/value pairs are supplied, all the <cfhttpparam> tags are nested between the opening and closing <cfhttp> tags, and the method specified in the <cfhttp> tag must be "post" :

```
<cfhttp method="post" url = "http://myserver/cgi-bin/broker">
  <cfhttpparam name="_program" value="mylib.mypgm.sas">
```

```
<cfhttpparam name="anotherparm" value="anotherval">
</cfhttp>
```

DISPLAYING RESULTS: THE <CFOUTPUT> TAG

The <cfoutput> tag is used to display results. Variable references are enclosed between # characters. The variable reference must appear nested between opening and closing <cfoutput> tags. The <cfhttp> request returns several variables; the SAS output is stored in the variable called cfhttp.FileContent .

```
<cfoutput>
  #cfhttp.FileContent#
</cfoutput>
```

SIMPLE EXAMPLES

In **Figure 1** we see a Web page, part of a larger ColdFusion application, that we will modify to include a menu of SAS requests and to display the results of those requests. We wish to place the menu on the right, above "LATEST NEWS," and to display the results to the left of our menu in the beige area.



Figure 1

ADDING A MENU PAGE

The areas on this page are defined by <div> tags. The area on the left is in a div called "project-areas-inner" and the area to the right is called "latest-news." Here is the relevant original code in "project-areas-inner."

```
<div id="project-areas-inner">
  
  <p>For over 40 years...roadway crashes.</p>
  <ul>
    <li><a href="xxx.cfm">Alcohol Impairment </a></li>
    .
    .
    <li><a href="yyy.cfm">Young Drivers </a></li>
  </ul>
</div>
```

We remove the "HSRC PROJECT AREAS" menu and header image so that the beige area to the left can become the target area for the SAS results.

```
<div id="project-areas-inner">
  <p> For over 40 years... roadway crashes.</p>
</div>
```

We insert a list of SAS requests before the "LATEST NEWS" on the right. For the sake of simplicity, each request will refer to a separate page. Here is the original "latest-news" section:

```
<div id="latest-news">
  
  .
  .
</div>
```

We add our menu at the top of the div section:

```
<div id="latest-news">
  <p>Choose one of the following to see embedded SAS results</p>
  <ul>
    <li><a href="SASrequestprint.cfm">Print</a></li>
    <li><a href="SASrequestmeans.cfm">Means</a></li>
    <li><a href="SASrequestnull.cfm">Null Datastep</a></li>
    <li><a href="SASrequestthsql.cfm">htmsQL</a></li>
    <li><a href="SASform.cfm">Form Example</a></li>
  </ul>
  
  .
  .
</div>
```

The results of the modifications are seen in **Figure 2**.

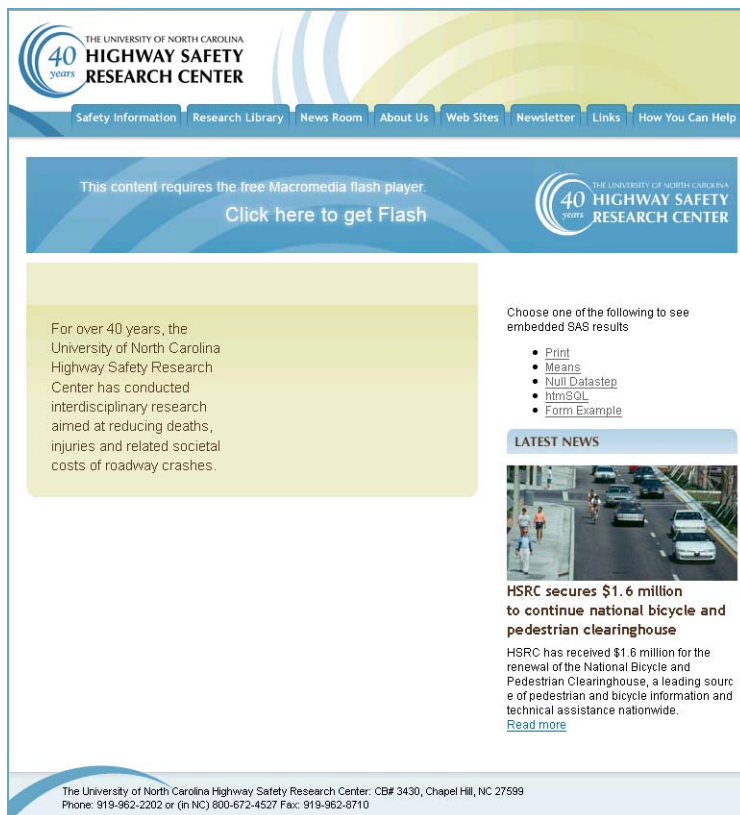


Figure 2

EXAMPLE: PROC PRINT

The page SASrequestprint.cfm, shown in **Figure 3**, is loaded when the 'Print' link is chosen from the list of SAS offerings. **Figure 3** shows our first example of SAS output placed in a ColdFusion page. We will deal with the fact that it is not attractive next, but first we examine the SAS and ColdFusion code creating the page. Here is the SAS program named SASrequestprint.sas and located in the program library defined to the Broker as 'martell':

```
ODS HTML FILE=_WEBOUT (NO_TOP_MATTER NO_BOTTOM_MATTER) STYLE=MINIMAL;
TITLE 'Print of Females in Sashelp.Class Table';
PROC PRINT DATA=SASHELP.CLASS; WHERE SEX='F'; RUN;
ODS HTML CLOSE;
```

It may seem awkward to the SAS programmer that both the request and the results are in the same page. ColdFusion both requests the PROC PRINT and displays the output of the PROC PRINT in one page. Two items must be added to the ColdFusion page. One sends the request to the Broker and the other displays the program results. Here is the new "project-areas-inner" div section:

```
<div id="project-areas-inner">
  <cfhttp url="http://mybroker/cgi-bin/broker" method="post">
    <cfhttpparam name="_program" value="martell.SASrequestprint.sas" type="formfield">
  </cfhttp>
  <cfoutput>
    #cfhttp.FileContent#
  </cfoutput>
</div>
```



Figure 3

Figure 3 is nice, but could use some cleanup. We remove the "Obs" column in SAS with the NOOBS parameter in PROC PRINT. The lines are distracting; tweaking the HTML table style is more easily done from the ColdFusion side. A 'View Source' reveals that SAS defines the table border and cell separators as follows:

```
<TABLE cellpadding=1 cellspacing=7 rules=all frame=Box border=3>
```

The horizontal rule and tag line after the procedure output are not appropriate here. The source code looks like this:

```
<HR>
```

```
<ADDRESS>
```

```
This request took 0.32 seconds of real time (v8.2 build 1391).
```

```
</ADDRESS>
```

We use ColdFusion functions and variables to change the table tag and to remove the horizontal rule and tag line. The ColdFusion Replace and Mid functions are like the TRANWRD and SUBSTR functions in SAS. We want a cellpadding of 0 to remove the lines between cells and we only want to keep the cellpadding value. Although a more universal fix might be desirable, for these purposes we will simply do a string substitution. Assigning the existing code to the variable "badstr" and the desired code to the variable "goodstr", we substitute one for the other with the Replace function and store the results in a variable called "outlist". After that modification we use the Find function to locate the position of the <HR> tag in "outlist". We subtract one from this value and store the results in the variable "endit", which now indicates the position immediately preceding the <HR> tag. This location value is used as the length in the Mid function to designate the endpoint of the string. The results of this final function are directed to be displayed in the Web page.

```

<cfoutput>
<cfset badstr="cellspacing=1 cellpadding=7 rules=all frame=Box border=3">
<cfset goodstr="cellspacing=0 cellpadding=7">
<cfset outlist=#Replace(cfhttp.FileContent,badstr,goodstr)#>
<cfset endit=find('<HR>','#outlist#)-1>
#Mid(outlist,1,endit)#
</cfoutput>

```

Our ColdFusion-tweaked SAS output appears in **Figure 4**. The same modifications will be applied to the remainder of the sample output in this paper, removing table lines and the ending tagline. The examples for **Figure 5** and for **Figure 6** also use the ColdFusion code seen above.

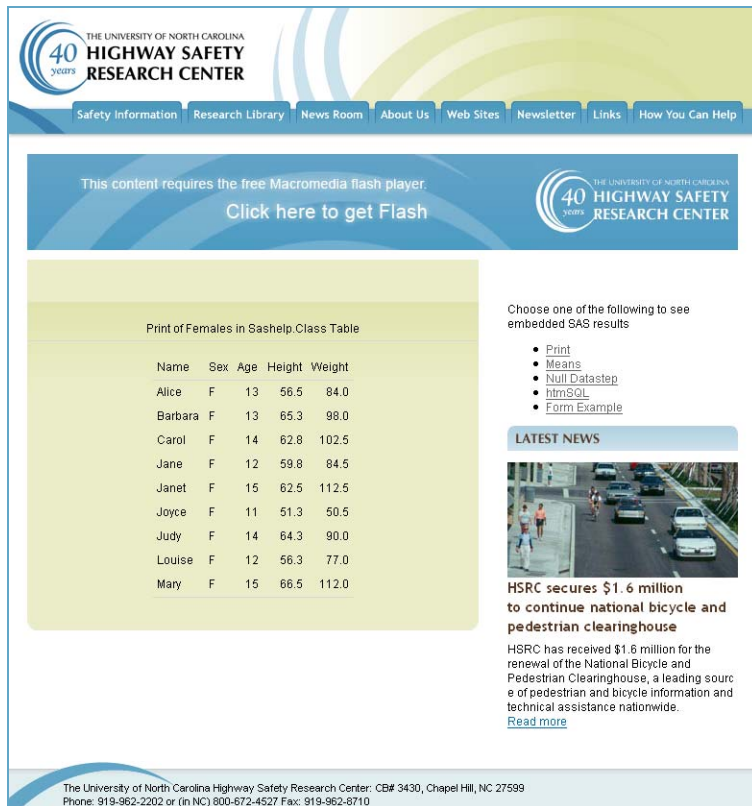


Figure 4

Similar results, shown in **Figure 5**, can be obtained by requesting the following MEANS procedure:

```

ODS HTML FILE=_WEBOUT (NO_TOP_MATTER NO_BOTTOM_MATTER) STYLE=MINIMAL;
TITLE 'Mean Weight and Height in Sashelp.Class';
PROC MEANS DATA=SASHELP.CLASS MAXDEC=3;
CLASS AGE;
VAR WEIGHT HEIGHT;
RUN;
ODS HTML CLOSE;

```

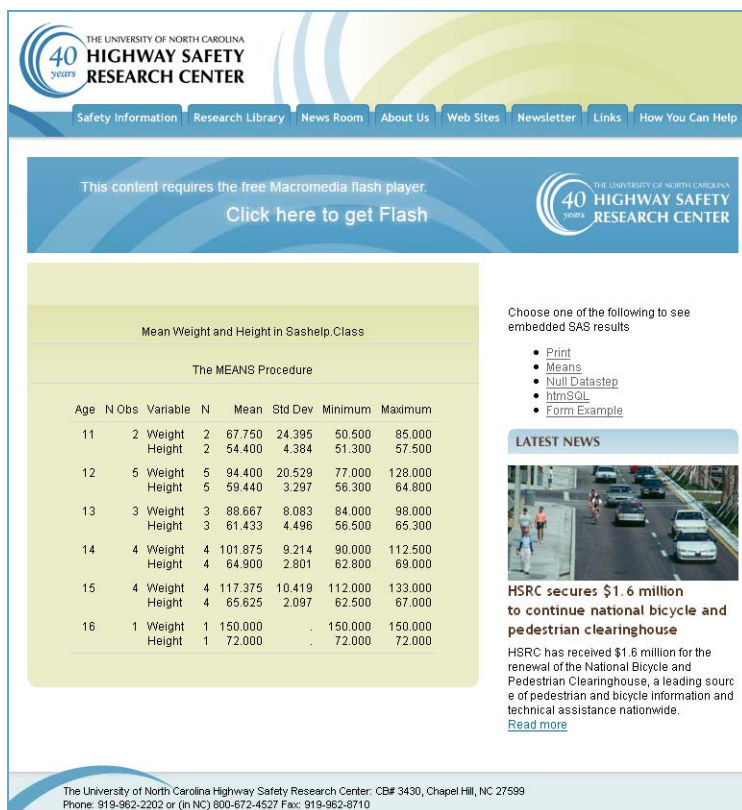


Figure 5

The SAS results need not be procedure output. A null datastep can write to _WEBOUT. This sample program writes some text, including HTML-formatted text, as well as displaying an image. The results are shown in **Figure 6**.

```
DATA _NULL_;
FILE _WEBOUT;
PUT '<center>This shows the results of a null datastep that writes directly to the
web page.';
PUT '<h2>HTML tags can be included. This line is surrounded by h2 tags.</h2>';
PUT 'Images in the target environment can be included. <br>Here''s the logo repeated
from the top of the page:';
PUT '<br></center>';
RUN;
```



Figure 6

Results obtained using htmSQL can also be captured and displayed. Calling the following htmSQL page yields the results seen in **Figure 7**:

```
<h3>This example shows that results from htmSQL, another SAS/IntrNet product, can
also be imbedded in a ColdFusion page</h3>
{QUERY SERVER="MYSERVER"}
{SQL}
  SELECT SUM(SEX='F') AS GIRLS, SUM(SEX='M') AS BOYS,
         MIN(HEIGHT) AS MINHT, MAX(HEIGHT) AS MAXHT,
         MIN(WEIGHT) AS MINWT, MAX(WEIGHT) AS MAXWT,
         MEAN(HEIGHT) AS MHT FORMAT=F6.1, MEAN(WEIGHT) AS MWT FORMAT=F6.1
  FROM SASHELP.CLASS
{/SQL}
{EACHROW}
  There were {&GIRLS} girls and {&BOYS} boys in the class dataset.<br><br>
  They range in height from {&MINHT} to {&MAXHT} inches with a mean of {&MHT}.
<br><br>
  Class weight ranges from {&MINWT} to {&MAXWT} pounds,
  with a mean weight of {&MWT}.
{/EACHROW}
{/QUERY}
```

The ColdFusion code to embed an htmSQL page follows. Because there are no name/value pairs, the method is not required to be "post".

```
<cfhttp url="http://statweb.unc.edu/~martell/SASrequesthtml.hsql" method="get">
</cfhttp>
<cfoutput>
#cfhttp.FileContent#
</cfoutput>
```

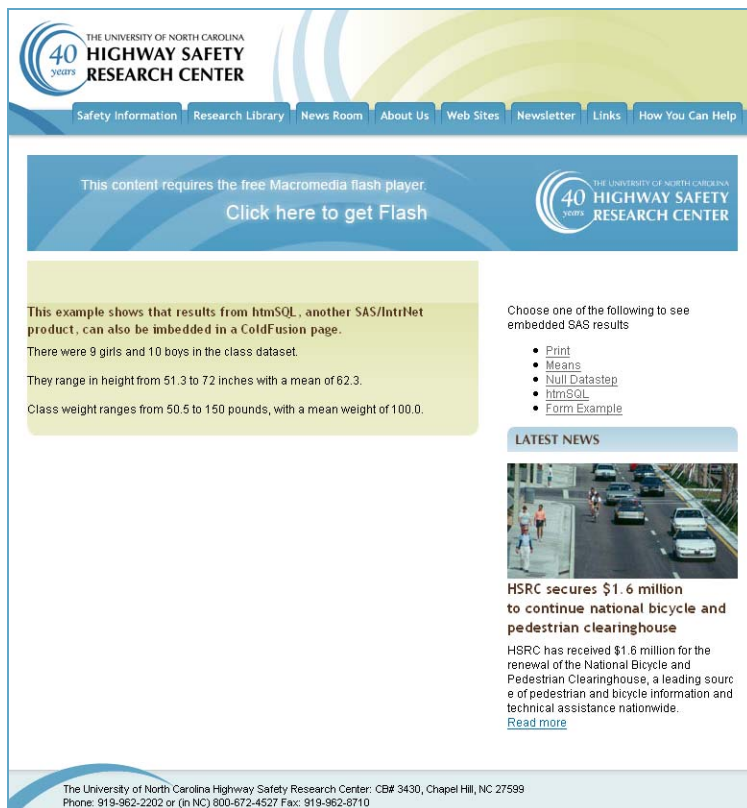


Figure 7

USING A FORM

In a real world application, a dynamic request would supply a form, allowing the user to customize the program to be run. With SAS/IntrNet, the action parameter of the form would normally be a call to the Broker. Since we do not want the results streamed to the web page, the Broker must be called via the `<cfhttp>` tag in a ColdFusion page. In this case, we make the action of the form be the current ColdFusion page, causing it to reload.

ColdFusion incorporates conditional logic using `<cfif><cfelse></cfif>` tags. Another ColdFusion feature, creating variables and setting default values, helps with the conditional processing. This example conditionally displays either the form or the results of submitting the form. The form variable "notrun" is created and assigned a default value of "1." When the page is loaded, if the value is "1," only the form is displayed. If the value is "0," only the results are displayed. The value is set to "0" in the form code. If the form is submitted, the ColdFusion logic uses the value "0" and displays only the results. When first loaded, the page appears as in **Figure 8**. Upon selecting 'males' and clicking submit, the page appears as in **Figure 9**.

To pass the name/value pair "sex" to SAS, the `<cfhttpparam>` tag for sex is assigned the value of the form variable "sex". All of the required code is placed within the `<div></div>` section:

```
<div id="project-areas-inner">
  <cfparam name="form.notrun" type="string" default="1">
  <cfif #form.notrun#>
    <cfform action="SASform.cfm" method="post" preservedata="yes">
      <input name="notrun" value="0" type="hidden">
      <cfinput type="radio" name="sex" value="1">Both sexes
      <cfinput type="radio" name="sex" value="M">males
      <cfinput type="radio" name="sex" value="F">females
      <input type="submit" value="submit">
    </cfform>
  <cfelse>
    <cfhttp url="http://mybroker/cgi-bin/broker" method="post">
      <cfhttpparam name="_program" value="martell.SASform.sas" type="formfield">
      <cfhttpparam name="sex" value="#form.sex#" type="formfield">
    </cfhttp>
  </cfout>
  #cfhttp.FileContent#
</div>
```

```
</cfif>
</div>
```

The SAS code in SASform.sas follows:

```
%MACRO MYJOB;
ODS HTML FILE=_WEBOUT (NO_TOP_MATTER NO_BOTTOM_MATTER) STYLE=MINIMAL;
TITLE Print of %IF &SEX=M %THEN Males in; %ELSE %IF &SEX=F %THEN Females in;
Sashelp.Class Table;
PROC PRINT NOOBS DATA=SASHELP.CLASS; %IF &SEX NE 1 %THEN WHERE SEX="&SEX";; RUN;
ODS HTML CLOSE;
%MEND MYJOB;
%MYJOB
```

The screenshot shows the website of the University of North Carolina Highway Safety Research Center. The header includes the center's name and a navigation menu with links: Safety Information, Research Library, News Room, About Us, Web Sites, Newsletter, Links, and How You Can Help. A banner indicates that the content requires a Macromedia flash player and provides a link to get it. Below this, there is a form with radio buttons for 'Both sexes', 'males', and 'females', and a 'submit' button. To the right of the form, there is a section titled 'Choose one of the following to see embedded SAS results' with links for 'Print', 'Means', 'Null Datasets', 'htmlSQL', and 'Form Example'. Below this is a 'LATEST NEWS' section featuring a photo of a street scene and a headline: 'HSRC secures \$1.6 million to continue national bicycle and pedestrian clearinghouse'. The text below the headline states: 'HSRC has received \$1.6 million for the renewal of the National Bicycle and Pedestrian Clearinghouse, a leading source of pedestrian and bicycle information and technical assistance nationwide.' and includes a 'Read more' link. The footer contains contact information for the center.

THE UNIVERSITY OF NORTH CAROLINA
40 years HIGHWAY SAFETY RESEARCH CENTER

Safety Information Research Library News Room About Us Web Sites Newsletter Links How You Can Help

This content requires the free Macromedia flash player.
Click here to get Flash

☐ Both sexes ☒ males ☐ females

Choose one of the following to see embedded SAS results

- [Print](#)
- [Means](#)
- [Null Datasets](#)
- [htmlSQL](#)
- [Form Example](#)

LATEST NEWS

HSRC secures \$1.6 million to continue national bicycle and pedestrian clearinghouse

HSRC has received \$1.6 million for the renewal of the National Bicycle and Pedestrian Clearinghouse, a leading source of pedestrian and bicycle information and technical assistance nationwide.
[Read more](#)

The University of North Carolina Highway Safety Research Center: CB# 3430, Chapel Hill, NC 27599
Phone: 919-962-2202 or (in NC) 800-672-4527 Fax: 919-962-8710

Figure 8



Figure 9

CONCLUSION

It is a fairly simple matter to blend the work of Web designers and ColdFusion developers with that of the SAS developer. This paper has presented one way, the use of the <cfhttp> tag, to incorporate the powerful processing and analyses that SAS provides into the flexible infrastructure of a ColdFusion application. Using this approach eliminates the need for the SAS program to incorporate the copious, complicated code behind a sophisticated Web page.

REFERENCES

SAS/IntrNet is documented in the Web Technologies Community at <http://support.sas.com>.
Syntax for ColdFusion tags are documented at <http://livedocs.macromedia.com/coldfusion>.

ACKNOWLEDGMENTS

The Web page used in the examples was lifted from the Highway Safety Research Center (HSRC) Web site. This site was recently redesigned by the Web development team at HSRC.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Carol Martell
Highway Safety Research Center
730 Martin Luther King Jr. Blvd.
Chapel Hill, NC 27599-3430
Work Phone: (919) 962-8713
Email: carol_martell@unc.edu
Web: <http://www.hsrc.unc.edu>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.