Paper 197-27

Dynamic Data Retrieval Using SAS/IntrNet®

Mary A. Bednarski, Washington University School of Medicine, St. Louis, MO
Karen A. Clark, Washington University School of Medicine, St. Louis, MO
Elizabeth M. Hornbeck, Washington University School of Medicine, St. Louis, MO
Mae O. Gordon, Washington University School of Medicine, St. Louis, MO
and the Ocular Hypertension Treatment Study Group

ABSTRACT

Our poster will demonstrate dynamic retrieval of clinical trial data via the web using SAS/IntrNet software. We use hyperlinks to remotely invoke SAS® as well as to create macro variables for accessing observations in SAS datasets. The data is sent back to the user's browser in a format that resembles a case report form. We accomplish this through background images and SAS/GRAPH®. We also discuss the security measures that we have implemented.

INTRODUCTION

DESCRIPTION OF THE OHTS

We represent the coordinating center for long-term, national, multi-clinic studies funded by the National Eye Institute, National Institutes of Health. The Ocular Hypertension Treatment Study (OHTS) is a randomized clinical trial evaluating the safety and efficacy of hypotensive medications in 1636 participants with ocular hypertension. Overall, we support over 200 staff members at 26 centers. Note: this paper will refer to clinics as "centers".

OHTS participants are followed twice yearly for a minimum of five years. We use more than 20 different paper case report forms (CRF). The OHTS Coordinating Center receives approximately 300 forms each week from our centers.

OHTS DATA

All OHTS data are maintained and analyzed using SAS. The data are entered via SAS on Windows NT workstations and then stored in SAS data sets on a UNIX machine. There is one SAS data set for each of the 20 CRF types. In this paper, we will use the terms "form" and "CRF" interchangeably, both referring to paper forms that are usually filled out by center personnel.

The OHTS data website, located on a UNIX machine, is accessible only by OHTS staff and requires a unique username and password. The website contains separate areas (UNIX directories) for each center. Most of the website reports are static reports written with the SAS *Output Delivery System (ODS)* or the SAS *%out2htm* output formatting tool. UNIX cronjobs schedule the SAS programs to run nightly, making the reports current to the end of the previous day.

ADDING A DYNAMIC SAS COMPONENT

We now discuss adding dynamic SAS to one of our static web listings, one that lists a participant's CRF history. For each CRF in a listing, we want to display all of the data contained in the appropriate SAS data set for that observation.

To retrieve this detailed information about individual CRFs, we need the capability to generate a large number of reports, specifically, one report for every CRF in our database. We currently have data from more than 93,000 CRFs in 20 SAS data sets. Creating 93,000 reports is impractical because of space and time considerations. Dynamic access allows us to view the data on demand, requires no disk space, and each request takes only seconds to run.

DYNAMIC OVERVIEW

To run SAS dynamically over the Internet, the SAS Application Dispatcher has been set up on our UNIX machine. The Application Dispatcher has two components: the Application Broker and the Application Server. The Application Broker, which interprets the information received from the web browser and passes it to the Application Server, is in a cgi directory. The Application Server, which invokes the SAS Dispatcher program, is in a directory inaccessible via the web.

Input must be sent from the static web page (the participant's CRF history) to the SAS Application Dispatcher. The input component that is sent from the form history contains name/value pairs: information such as the location of the SAS Application Broker, the type of service (socket, pool, launch), the location and name of the SAS program to be run, and any values we want the Dispatcher to pass to the SAS program in macro variables. There are several ways to send this input component to the Dispatcher application. We choose to send the input via hyperlinks that are imbedded in the HTML code of the static form history.

Once in the form history report, users need only a mouse to click on a study ID and CRF. Because we are using a preprogrammed hyperlink specifying the participant, CRF type, and CRF date, users do not enter any parameters. This makes it very easy to use.

For this paper, we have chosen our Adverse Event (AE) form to use as our data retrieval example. We will describe the major components of our Dispatcher program in the following sections. The code is included in this paper's Appendix.

HOW IT WORKS

Each night, a UNIX SAS program uses *ODS* to create 1636 CRF history reports: one listing of every CRF received for each of our 1636 OHTS participants.

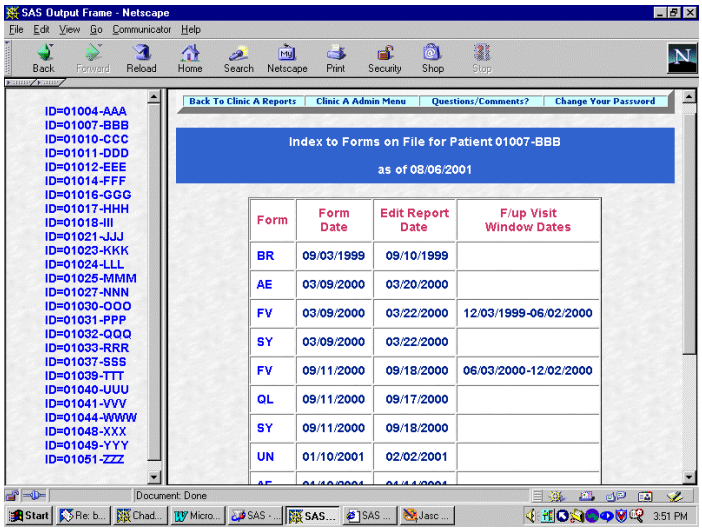
The program creates a hyperlink in a data step by inserting HTML text in a variable, newform:

```
newform='<a href="' ||
compress('http://www2.ohts.wustl.edu/cgi-
bin/broker?_service=default'   ||
'&_program=ohtsprog.pthis_'    || lowcase(substr(form,1,2))      ||
'.sas&thedate=' || put(examdate,mmddyy10.) ||
'&theid=' || id ||
'">' || form    || '</a>' ) ;
```

The program uses *proc report* along with *ODS* and a *style definition* (not detailed in this paper) to format the page and create the static form listing:

```
ods listing close;
ods html body='pthistory_body.htm'
      contents='pthistory_content.htm'
      frame='pthistory_frame.htm'
      path="/public_html/ohts&sitelc/"
      (url=none)
      newfile=page
      headtext='<style type="text/css"> A
      {text-decoration:none;}</style>'
      style=styles.webrpts;
ods proclabel ' ';

proc report nowindows data=work.printout
     split='*' contents="    colwidth=10  ;
     by patient ;
     label patient='ID' ;
     title1 "Index to Forms on File for Patient #byval(patient)";
     title2 "as of &curdate";
     column newform examdate editdate window ;
     define newform  / 'Form';
     define examdate / 'Form*Date' format=mmddyy10.;
     define editdate   / 'Edit Report*Date' format=mmddyy10.;
     define window   / 'F/up Visit*Window Dates';
run;
quit;
ods html close;
```

This code creates a form history listing:



An approved user goes to the table of contents that lists study participant IDs and clicks on the one of interest. The CRF history list for that participant pops up on the screen.

The dynamic part now begins. To view all data for a specific CRF, the user clicks on a CRF name (AE, in this example). This activates a hyperlink (created previously with the variable newform), sending a command to invoke SAS on the server. The hyperlink shows up in the browser's location/address bar:

```
http://www2.ohts.wustl.edu/cgi-bin/broker
?_service=default&_program=ohtsprog.pthis_ae.sas&thedate=0
3/09/2000&theid=01007
```

Everything to the left of the question mark (?) is the location of the Broker. The information to the right of the question mark is called the query string and the character & is the separator between name/value pairs.

_service=default tells the Broker to use the default service, however the Broker has defined it.
_program=ohtsprog.pthis_ae.sas gives the program library (defined in the Dispatcher's appstart.sas file) and the Dispatcher program to be run.
(the above two name/value pairs are required)
thedate=03/09/2000 tells the Application Dispatcher to create a macro variable &thedate and give it a value of '03/09/2000'.
theid=01007 tells the Application Dispatcher to create a macro variable *&theid* and give it a value of '01007'.

The hyperlink is subsequently sent to the Apache Web Server, then on to the SAS Application Broker. The Broker checks the information sent in the link, then sends the request to the Application Server. The Application Server runs the Dispatcher program specified in the hyperlink (pthis_ae.sas). Meanwhile the name/value pairs have been made into SAS macro variables. The program pthis_ae.sas uses the macro variables theid and thedate to extract the requested observation from the AE data set:

```
if id="&theid" and examdate=input("&thedate",mmddyy10.);
```

note: macro variable values can also be retrieved with the *symget* or *%superq* functions.

If the ID and examdate are found in the AE data set, the program sends all of that observation's data to the user's browser. Below are the mechanics of the Dispatcher program pthis_ae.sas:

The AE paper form is two pages in length, so we first create annotate data sets for each page and call them work.pthist1 and work.pthist2. These data sets contain the AE data and (x,y) coordinates that position the data to make the output resemble the layout of the paper AE form. The creation of the annotate data sets is trivial yet requires a lot of patience and is beyond the scope of this paper. If you would like information on their creation, please contact the authors.

We use two graphics files, aepage1.gif and aepage2.gif, as the backgrounds of their respective AE pages. They were created by scanning in the pages of the CRF (AE, in our example) and saving in a .gif format. The program uses these files by specifying *iback=imgloc&page* in the *goptions* statement.

For each page of the CRF, the program executes a macro named sendinfo. With each macro call, a *proc template* runs and creates the *ODS style definition* named pthis. The *style definition* pthis creates an HTML navigation bar for the top of each page of output. For simplicity, we have excluded the

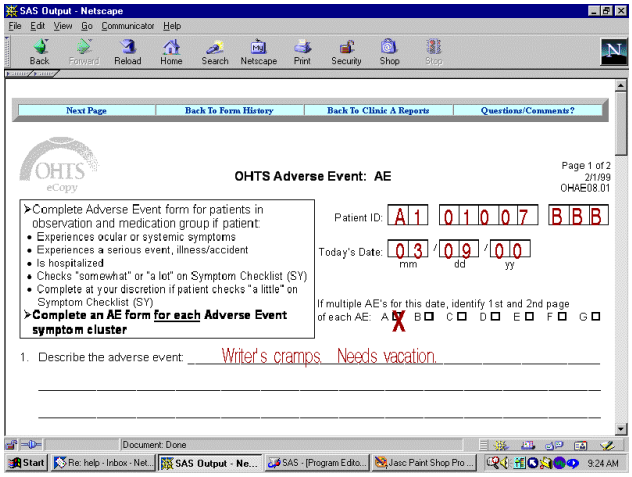
template procedure from this paper. The *proc template* code is available from the authors upon request.

Macro sendinfo uses *proc ganno* to creates gif that are combinations of the annotate data sets and their respective background image files. For example, a gif is created that integrates work.pthist1 with aepage1.gif. All of this information is sent to the user's browser in the form of an HTML page with embedded gifs and navigation bars appearing before each gif.

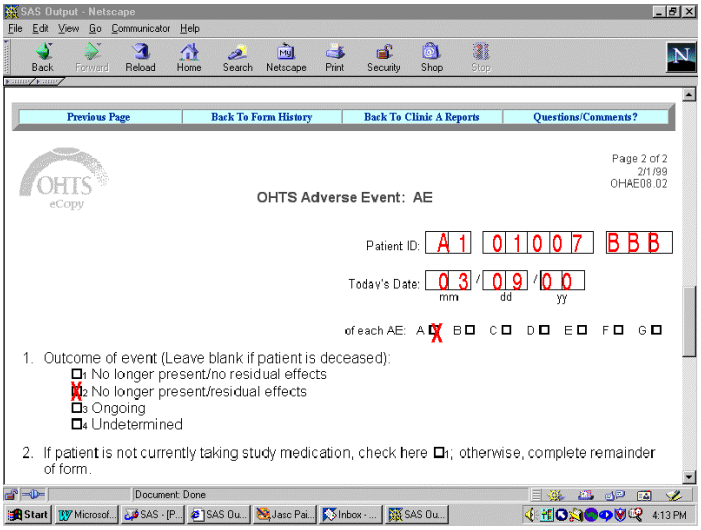
The code for macro sendinfo is as follows:

```
%macro sendinfo(page);    *** page is 1 or 2;;;

ods html body=_webout path=&_tmpcat
          (url=&_replay) rs=none
          style=pthis;

***The ods statement above can be explained at
http://www.sas.com/rnd/web/intrnet/dispatch/ods.html;;;

filename imgloc1 '/public_html/aepage1.gif';
filename imgloc2 '/public_html/aepage2.gif';
goptions reset=all;
goptions device=gif gunit=pct aspect=1
    imagestyle=fit transparency noborder
    iback=imgloc&page
    vsize=11  HSIZE=7.7  xpixels=800
    ypixels=1100 ymax=11  xmax=8;

proc ganno annotate=work.pthist&page; run;
%mend sendinfo;
```

The results are sent to the user's browser:



The resultant web page is designed to look like an actual copy of the CRF. This was accomplished via the background images in the *goptions* statement. The navigational bar at the top of the AE page was inserted through the *ODS style definition*. Clicking on the "Next Page" navigational button takes the user to page two of the AE form :



Pages one and two are actually returned at the same time to the same browser window, with page one appearing at the top of the screen. Likewise, for a form that is ten pages in length, each of the ten pages would be returned in the same browser window.

SECURITY

OHTS must ensure that our data are secure and that OHTS personnel see data from their centers only. To accomplish this, we use the htaccess access control method for authentication on our UNIX based Apache web server. We place .htaccess files in directories to which we want to restrict access to only authorized OHTS users. Each center has its own directory and is not allowed to access any other center's directory. This is easily managed through .htaccess files. The administrator uses a cgi program named user_manage to maintain the .htaccess files, primarily assigning userids/passwords to individual users. The userids are four characters long, with the center's initial serving as the first character. Htaccess tutorials can be found on many websites.

When users first access any report on the OHTS website, a userid/password screen pops up. Entering a valid userid/password combination allows the user to access any static report that resides in the UNIX directory dedicated to their center's reports. It also allows the user to dynamically run any program in the Application Dispatcher directory.

In addition to htaccess restrictions, we have added additional security measures. Since our dynamic pages are generated via hyperlinks, such links can be sent to our Apache web server via a browser's location bar. We need to prevent any astute user from simply changing the name/value pairs in the location bar and possibly retrieving another center's data. To this end, we use the SAS macro variable *_RMTUSER* to verify that the remote user is in fact permitted to view the data that he/she is requesting. Apache passes the htaccess userid to the Broker and the Broker puts the userid in the macro variable *_RMTUSER* for SAS programs to use.

```
if substr("&_rmtuser",1,1)=put("&theid",$center.);
```

This code works because when OHTS assigns userids, we make the first character of a userid the user's center. For instance, a

user with initials 'nal' from center 'a' would have the userid anal. We retrieve the participant's center through a permanent SAS format, $center, associated with that study participant. We compare the center name as gotten from the userid (in our example, the center name would be 'a') with the participant's center.

To further secure our website, we need to ensure that the *_RMTUSER* variable that SAS receives is coming from the Broker and not from any circumvention of the system. Only if *_RMTUSER* comes from the Broker are we certain that *_RMTUSER* is the same as the htaccess userid. We accomplish this by setting the variable *passkey* in the broker configuration file to some confidential, unique value. This is the line from our broker configuration file (value changed for this paper):

```
ServiceSet passkey "supersecret007word"
```

When the Broker receives a request, the Broker sends the Application Server *passkey* as a macro variable. In the SAS program, we check the value of *passkey*.

```
if symget('passkey')='supersecret007word';
```

If the value is not what we set up in the Broker configuration file, this means that the request has not come through the Application Broker and the request is denied.

For more information, see SAS Institute's web page on "Application Server Security" at http://www.sas.com/rnd/web/intrnet/dispatch/appsec.html

We put these security measures in one datastep in the SAS program:

```
data work.ptfound;
  set unixdata.ae;
  if id="&theid" and
    examdate=input("&thedate",mmddyy10.) and
    substr("&_rmtuser",1,1) = put("&theid",$center.) and
    symget('passkey')='supersecret007word';
run;
```

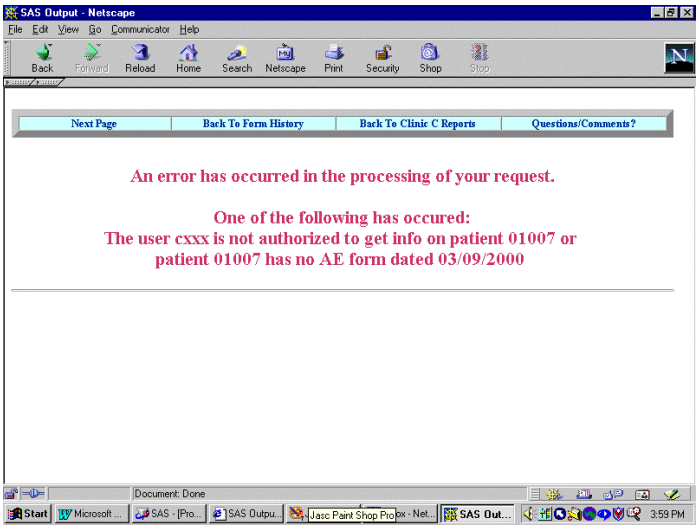
If the *_rmtuser* and *passkey* values pass our security checks and the participant's ID and the AE examdate are found in the AE data set, work.ptfound will contain the observation that we will format and send to the user's browser. If either of the security checks fails or the observation is not found in the AE data set, work.ptfound will be empty. An error page is then sent to the browser window.

In this case, we still call macro sendinfo but only to create the navigation bar. SAS does not output anything from the ganno part because the annotate data set is empty. Since we don't get ganno output, we add a data step that sends an error message back to the browser by using the statement *file _webout*:

```
data _null_;
  if howmany=0 then call
  symput('howmany',howmany);
  else call symput('howmany','1');
  set work.ptfound nobs=howmany;
run;
```

```
***The null datastep gives the macro variable howmany the
value of 0 if work.ptfound is empty; otherwise, a value of 1.;;;

%if &howmany=0 %then %do;
  %sendinfo(1);

data _null_;
 file _webout;
  put "<center><h2><font color='#cc3366'><br>";
  put "An error has occurred in the
    processing of your request.<br><br>";
  put "One of the following has occurred:<br>";
  put "The user &_rmtuser is not authorized
    to get info on patient &theid or <br>";
  put "patient &theid has no AE form dated  &thedate.";
  put "</font></h2></center>";
run;
%end;
```

No further processing is done and the following error screen is sent to the browser window:



One more security measure we have implemented is disabling all debug values in the broker file except for debug=2. Disabling debugging prevents the SAS Log or any of the macro variables passed to SAS by the Broker (for example, the confidential *passkey)* from being displayed in the user's browser. Because of this, we cannot debug programs on-line. Therefore, we use a *proc printto* statement to save our SAS Log to a file when testing a new SAS program. You can get more information on debug values at http://www.sas.com/rnd/web/intrnet/dispatch/brwebsec.html and at http://www.sas.com/rnd/web/intrnet/dispatch/debuginp.html. Finally, we have ensured that our SAS programs exist in a UNIX directory that is not accessible via the web and that there is nothing in our cgi-bin directory (where the Broker resides) that will be a risk to our security.

FUTURE PLANS

At this time, our dynamic data retrieval report is meant to be viewed on the screen; printing gives inconsistent results. To make printing a viable option, we would like to offer the reports as pdf files in the future. In preparation for printing, we have added a watermark to the image to distinguish it as an e-Copy.

We also have planned to use SAS/IntrNet to display an audit trail of the changes to form data that have occurred.

Since we finished this project, we have implemented SAS/IntrNet projects that use HTML forms instead of hyperlinks as the means of sending the input component to SAS. We've had much success with these interactive queries because users have the freedom to design their own reports within predefined parameters and the users can create these reports without having to submit a request to a programmer.

CONCLUSIONS

Since most of our users are novice computer users, making the data retrieved on-screen look more like a familiar paper form is preferable to doing a standard SAS *proc print*. This reduces confusion and assuages fears of "the unknown". Giving the centers glimpses into our database helps them and us investigate data anomalies between paper forms and our database. On-screen viewing also gives us immediate access to a "form" instead of having to wait 24 hours or more to retrieve older CRFs that we have placed in off-site storage.

Dynamic SAS is the only practical way we could have given centers the option of viewing any of the 93,000 CRFs in our SAS database. The time and space requirements for our dynamic implementation are minimal and the results are impressive.

ACKNOWLEDGMENTS

NIH EY09341, EY09307, Unrestricted grants from Research to Prevent Blindness.

CONTACT INFORMATION

Mary Bednarski
Washington University School of Medicine
Department of Ophthalmology and Visual Sciences
Campus Box 8096
660 South Euclid Avenue
St. Louis, MO 63110
Phone: (314) 362-4348
Email: maryb@vrcc.wustl.edu

APPENDIX

```
*** pthis_ae.sas;
*** invoked from a hyperlink located in the static CRF history
      listing ;
*** retrieves all information for a specific AE
      form in the AE data set;
*** uses a background .gif image to send the data to the user's
      browser in a format that resembles the AE paper form;

%macro beginnow;

libname unixdata '/data/ohts/unixdata';
libname library    '/data/ohts/library';
libname sasuser   '/data/ohts/';
proc printto log= '/users/ohts/dispatcher/aeinfo.log' new;
run;

%annomac;

%macro sendinfo(page);

 data _null_;
 *** create two macro variables, prevar and  postvar, to be
      used  by proc template.  Prevar contains html code that
      creates the  navigation bar to be inserted at the top of
      the browser page.  Postvar contains html  code that defines
      the top of AE page two as a target of a hyperlink. This lets
      us create a "next page" navigational button to link to page
      two. The value of &postvar is "<a name='aep2'>
      nbsp;</a>", the value of  &prevar is too lengthy to include
      in this discussion;;;
 run;

 proc template;
   define style pthis;
    parent=styles.minimal;
       Style Body from body /
         prehtml= symget("prevar")
         posthtml=symget("postvar");
  end;
 run;

 ods listing close;
 ods html body=_webout path=&_tmpcat
        (url=&_replay) rs=none style=pthis
        headtext='<style type="text/css"> A
        {color:#ffffff; text-decoration:
        none; font-size:9pt} </style>';

 filename imgloc1 '/public_html/aepage1.gif';
 filename imgloc2 '/public_html/aepage2.gif';
 goptions reset=all;
 goptions device=gif gunit=pct aspect=1
   imagestyle=fit transparency noborder
   iback=imgloc&page
   vsize=11 size=7.7 xpixels=800  ypixels=1100 ymax=11
   xmax=8;

 proc ganno annotate=work.pthist&page; run;

%mend sendinfo;

data work.ptfound;
  set unixdata.ae;
  if id="&theid" and
    examdate=input("&thedate",mmddyy10.) and
    upcase(substr("&_rmtuser",1,1))= put("&theid",$center.)
    and symget('passkey')='supersecret007word';
run;

data _null_;
   if howmany=0 then call
   symput('howmany',howmany);
   else call symput('howmany','1');
   set work.ptfound nobs=howmany;
run;

%if &howmany=0 %then %do;
 %sendinfo(1);

 data _null_;
   file _webout;
   put "<center><h2><font color='#cc3366'><br>";
   put "An error has occurred in the
         processing of your request.<br><br>";
   put "One of the following has occurred:<br>";
   put "The user &_rmtuser is not authorized
         to get info on patient &theid or <br>";
   put "patient &theid has no AE form dated &thedate.";
   put "</font></h2></center>";
 run;

%end;
%else %do;

 data work.pthist1;
    set work.ptfound;
    ***create annotate data set for page one of  the AE using the
        annotate macros;;;
  run;

  data work.pthist2;
     set work.ptfound;
     ***create annotate data set for page two of the AE using the
         annotate macros;;;
  run;

  %sendinfo(1);
  %sendinfo(2);

%end;

ods html close;

%mend beginnow;

%beginnow;
```