

DATA RETRIEVAL USING SAS INTRNET[®]

Karen A. Clark, Mary A. Bednarski, Joel F. Achtenberg, Julia A. Beiser, Denise M. Randant, Christopher J. Ewing, Mae O. Gordon, and the Ocular Hypertension Treatment Study (OHTS) Group, Washington University School of Medicine, St. Louis, MO

ABSTRACT

The Ocular Hypertension Treatment Study (OHTS) is a multi-center, randomized clinical trial involving 22 clinics and 1636 participants. The OHTS Coordinating Center and clinics use the study's website to view summary reports such as case report form (CRF) history and clinic performance reports. These reports are produced statically using the SAS %OUT2HTML output formatting tool. By adding the SAS IntrNet component, we provide dynamic access to detailed CRF data by creating reports on demand without tying up electronic storage space. This dynamic web system reduces the need to retrieve remotely stored hard copy CRFs and presents a tool for ensuring data quality. This paper will demonstrate the dynamic electronic retrieval of CRF data using SAS ODS and IntrNet software. Security measures are described in detail and future enhancements are discussed.

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 22 clinics.

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

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. Each night the active data sets are copied to another UNIX directory that is accessible by web reports. We do this to avoid possible conflicts between data entry and accessing the data via the internet.

STATIC WEB SYSTEM

The OHTS website, located on a UNIX machine, is accessible only by OHTS staff and requires a unique username and password. The OHTS website contains separate areas (UNIX directories) for each OHTS center. The website reports are static reports written with the SAS %out2html output formatting tool or ODS. UNIX cronjobs schedule the SAS programs to run

nightly, making the reports current to the end of the previous day.

ADDING DYNAMIC SAS

We need the capability to generate a large number of reports, specifically, one report for every CRF in our database. We currently have more than 83,000 CRFs in 20 different SAS data sets. Creating 83,000 reports is impractical because of space and time considerations. Dynamic access allows us to view the data on demand, occupies nominal disk space, and each request takes only seconds to run.

DYNAMIC IMPLEMENTATION

One of our static web reports is a list of each participant's CRF history. We are now adding dynamic SAS to that report to give detailed information about the CRFs. Given a specific participant, CRF type, and CRF date, SAS sends all of the data contained in the SAS database for that observation to the user's browser.

In order to run SAS dynamically over the Internet, the SAS Application Dispatcher has been set up on our UNIX machine. The Application Broker is in a cgi directory and the Application Server, which invokes the SAS program, is in a directory inaccessible via the web.

Input must be sent from the static web page (the participant's form 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 have chosen 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 pre-programmed hyperlink, users do not enter any parameters, which 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 be describing the major components of our SAS programs in the next few sections. The complete code is included in the appendix at the end of the paper.

THE PROCESS

Each night, a UNIX SAS program uses ODS to create 1636 form history reports: one listing of every CRF received for each

participant. The program uses *proc report* along with an *ODS style definition* (not included in this paper) to format the page and creates a hyperlink by inserting HTML text in a variable. We create the hyperlink in the variable newform:

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

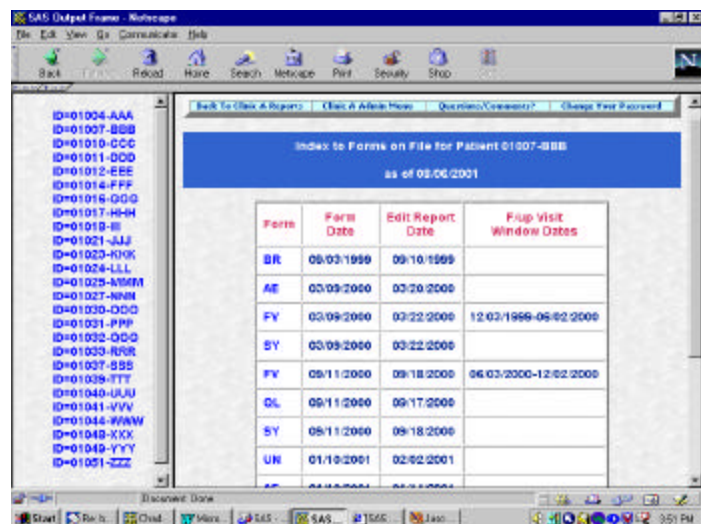
The code that creates the static form listing is:

```
ods listing close;
ods html body='pthistory_body.htm'
      contents='pthistory_content.htm'
      frame='pthistory_frame.htm'
      path="/public_html/ohs&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 report:



Form	Form Date	Edit Report Date	F/up Visit Window Dates
BR	03/03/1999	03/10/1999	
AE	03/09/2000	03/20/2000	
FY	03/09/2000	03/22/2000	12/03/1999-05/02/2000
SY	03/09/2000	03/22/2000	
FY	05/11/2000	05/18/2000	06/03/2000-12/02/2000
GL	05/11/2000	05/17/2000	
SY	05/11/2000	05/18/2000	
UN	01/19/2001	02/02/2001	

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

The dynamic part now begins. To view all the data on a specific CRF, the user clicks on a CRF name (AE, in this example). This activates the hyperlink, sending a command to invoke SAS on the server:

```
http://www2.ohs.wustl.edu/cgi-bin/broker
?_service=default&_program=ohsprog.pthis_ae.sas
&thedata=03/09/2000&theid=01007
```

/*note: this is the link that was created by the variable newform*/

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=ohsprog.pthis_ae.sas gives the library and the program name to be run.

(the above two name/value pairs are required)

thedata=03/09/2000 tells the Application Dispatcher to create a macro variable &thedata 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 which runs the program specified in the hyperlink (pthis_ae.sas). Meanwhile the name/value pairs have been made into SAS macro variables. The program uses these macro variables (theid, thedate) and extracts the requested observation from the CRF database.

```
if id="&theid" and
examdate=input("&thedata",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, a *proc ganno* is run that outputs all of that observation's data to the user's browser. In the program pthis_ae.sas, we first create annotate data sets work.pthist1 and work.pthist2 that hold AE data formatted to resemble the layout of the AE form. The creation of the annotate data sets is beyond the scope of this paper. If you would like information on their creation, please contact the authors. *Ganno* outputs the information in these data sets to the web via the following macro, sendinfo.

/*This is a simplification of the macro sendinfo that creates the dynamic report. */

```
%macro sendinfo(page);
```

/** For this AE program, the value of page is either 1 or 2 because the AE paper form is only two pages in length**/

/** The macro sendinfo also includes a *proc template* that creates the *ODS style definition* pthis. For simplicity, we have excluded the template procedure from this paper. The *proc template* code is available from the authors upon request. **/

```
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>. The style option uses the *style definition* pthis that we created earlier. **/

```
filename imgloc1 '/public_html/aeimage1.gif';  
filename imgloc2 '/public_html/aeimage2.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;
```

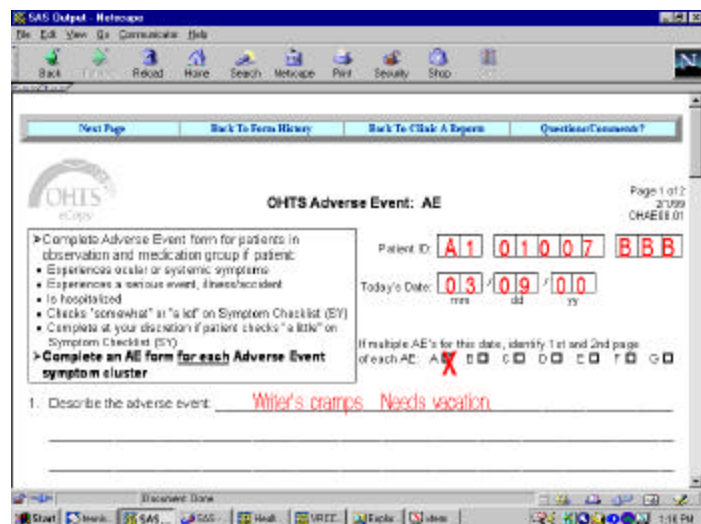
/** Aepage1.gif and aepage2.gif are the two graphics files we use as the backgrounds of the respective AE pages. They were created by scanning in the pages of the CRF (AE, in our example) and saving as GIF files. These files are used by specifying *iback=imgloc&page* in the *goptions* statement. For each *ganno*, SAS creates a gif that is a combination of the AE data and one of the background image files. **/

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

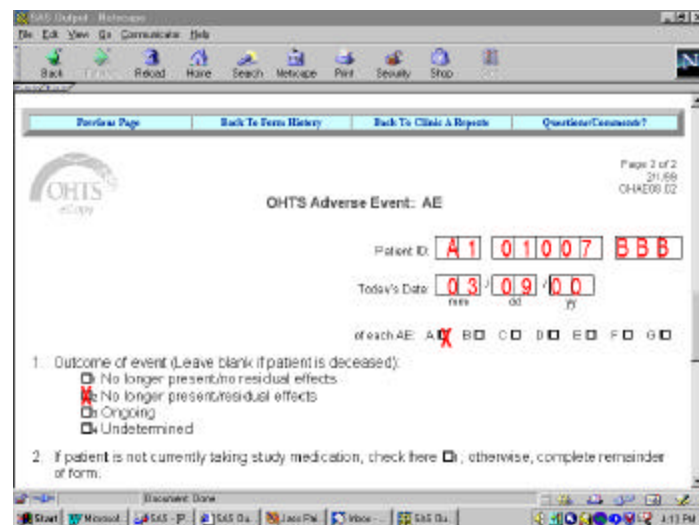
/** The annotate data sets are work.pthist1 and work.pthist2 that were created prior to invoking the macro sendinfo. **/

```
%mend sendinfo;
```

The results are returned 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 program *htaccess* 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 clinic has its own directory and is not allowed to access any other clinic's directory. This is easily managed through *.htaccess* files. We use a cgi program called *user_manage* to maintain the *.htaccess* files. *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 all these security measures in one dataset in the SAS program:

```
data work.ptfound;  
  set unixdata.ae;  
  if id="&theid" and  
    examdate=input("&thedate",mmdyy10.)  
    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, an error page is sent to the browser window. This is implemented via the following code:

```
data _null_;  
  if howmany=0 then call  
    symput('howmany',howmany);
```

```
else call symput('howmany','1');  
set work.ptfound nobs=howmany;  
run;
```

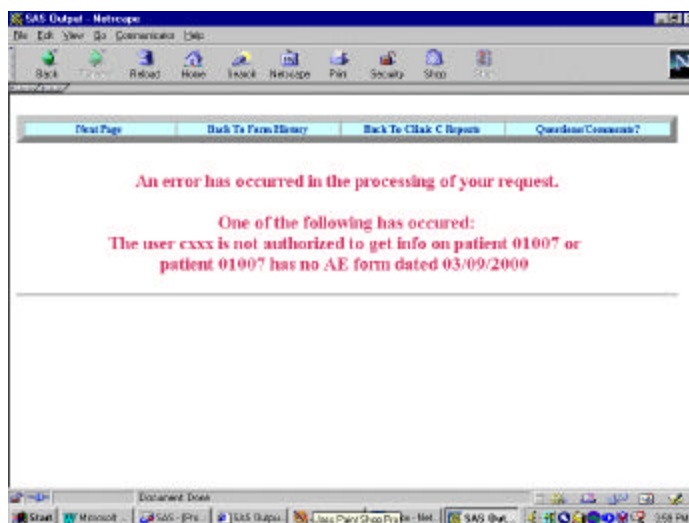
/**The null dataset gives the macro variable `howmany` the value of 0 if `work.ptfound` is empty; otherwise, a value of 1.**/

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

/**As stated previously, the macro `sendinfo` contains the style definition `pthis`, the ODS statement, and the `ganno` procedure. In this case, the `ganno` part is not run because the annotate data sets are empty. Instead, the following data step sends the error information back to the browser by using the statement `file _webout:`*/

```
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 can not 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 system has been made available only to in-house personnel. We hope to offer this dynamic retrieval of data to all of our centers by the time we present our poster at MWSUG in November 2001. We have programmed data retrieval of only a few CRF types at this time. We plan to offer data retrieval of additional CRFs as we are able to program them. Since we have 20 CRF types and some have as many as nine prior versions that will need to be programmed, this is not expected to be a quick process.

We now use data that is current to the end of the previous day. We are working on a scheme that will allow us to use data that is current to the time that the dynamic SAS is run. This will involve modifying data entry programs that are run on our NT workstations.

Another enhancement we are considering is to give users an option of viewing the average value of each CRF variable for all centers combined in addition to viewing their own center's data.

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.

Other ideas of using dynamic SAS in the future include using HTML forms as the input component and giving users a finite number of variables or parameters that they can manipulate to produce unique reports. This flexibility will help ensure that centers can create reports that are useful for their specific needs.

We also hope to develop a web based data entry system in the future.

CONCLUSIONS

Since most of our users are novice computer users, making the data retrieved on-screen look like a familiar paper form should reduce confusion and assuage fears of "the unknown". Giving the centers glimpses into our database will help us and them 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 83,000 CRFs in our SAS database. The time and space requirements for our dynamic implementation are minimal and the data on-demand character of this report is expected to be well received by our users. If nothing else, it got us to MWSUG.

ACKNOWLEDGMENTS

NIH EY09341, EY09307, Unrestricted grants from Research to Prevent Blindness. We would also like to thank Jack Baty and Betsy Hornbeck for their valuable assistance, as well as SAS Institute's website for its abundance of information.

APPENDIX

```

* program pthistory.sas;
* creates the static patient history report;
* writes each patient's form history to its
  own html file;
* includes links to run SAS dynamically;

libname unixdata '/data/ohts/unixdata';
libname library  '/data/ohts/library';
libname sasuser  '/data/ohts/';

data work.allforms;
  set unixdata.br(keep=id initials form examdate
editdate window) unixdata.cf(keep=id initials
form examdate editdate window)...;
  *read in all observations from all CRF
  data sets and concatenate them into
  work.allforms;
  patient=put(idnum,z5.) || '-' || initials;
run;

proc sort data=work.allforms; by patient; run;

%macro pt_hist (clinic, site_lc, central);

  options nodate source center nonumber nobyline
    ps=250 ls=78;
  title;
  data work.printout;
    set work.allforms;
    if put(id,center.)="&clinic";
    newform='<a href="' ||
compress('http://www2.ohts.wustl.edu/cgi-
bin/broker?_service=default' ||
'&_program=ohtsprog.pthis_' ||
lowercase(substr(form,1,2)) ||
'.sas&thedate=' || put(examdate,mmddyy10.) ||
'&theid=' || id ||
'">' || form || '</a>' ) ;
  run;

  ods listing close;
  %inc '/data/ohts/create_style.sas';
  *the style definition we use for all static
  reports is in the program create_style.sas;

  ods html body='pthistory_body.htm'
    contents='pthistory_content.htm'
    frame='pthistory_frame.htm'
    path="/public_html/ohts&site_lc/"
    (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;
  %mend pt_hist;

  %pt_hist(A, a, pat);
  .
  .
  .
  %pt_hist(W, w , ann);

  ods listing;
  options byline; title;

```

```

* pthis_ae.sas;
* runs dynamically from a hyperlink
  located in the static report
  pthistory_body.htm;
* retrieves all information for a specific AE
  form for a specific OHTS participant;
* sends the information to the user's browser
  in a format that resembles the actual 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;
```

```

data _null_;
  site_lc = (substr("&rmtuser",1,1)) ;
  if upcase(site_lc) in ('A' 'C' 'H' 'J' 'L' 'M'
    'Q' 'R' 'V' 'W') then call
    symput('cc','pat');
  else call symput('cc','ann');
run;

```

```
%macro sendinfo(page);
```

```

data _null_;
  *create two macro variables, prevar and
  postvar, that will 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 use the "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 hsize=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("&thedata",mmdyy10.)
    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;
```

```

ods listing;
proc printto log=log; run;

```