Three Ways to Utilize the SAS/IntrNet™ Application Dispatcher

Steven L. Willhoite, Litton/PRC, San Antonio, TX
John P. Campbell, Litton/PRC, San Antonio, TX
George R. Orr, Litton/PRC, San Antonio, TX

ABSTRACT

This paper provides examples showing how we have utilized the SAS/IntrNet Application Dispatcher along with the Output Formatter macro on three different types of projects. The first project selects a few parameters (macro values) from a web page, submits the request to the SAS® System, and returns standardized output to the user's web browser. The second project collects survey data and stores the data in a permanent SAS data set using multiple dynamic HTML pages and SAS programs. The final project demonstrates a method of submitting an ad hoc query to SAS/SCL® and returning the results of the query to the user's web browser. We also demonstrate a method to restrict access to these applications using the SAS System and the Application Dispatcher. The programs were tested using Netscape Navigator™ 3.x, Microsoft Internet Explorer™ 4.x and SAS 6.12 for Solaris™.

INTRODUCTION

The U.S. Army's Center for Healthcare Education and Studies (CHES), Analysis Branch, Fort Sam Houston, Texas, performs analysis of large medical databases and studies of specialized medical areas. Directed by Dr. Scott A. Optenberg, Ph.D., large volumes of reports are produced annually and distributed world wide. The task of putting a completed report together and delivering it to its recipients is expensive and time consuming. Additionally, the CHES provides remote users access to these large databases to request standardized reports and execute ad hoc queries.

The emerging technology of the web offers a more practical means of providing these reports to the intended targets at a much lower cost and allowing access by remote users with little or no additional software expenses. Until the SAS Institute developed the SAS/IntrNet Application Dispatcher, we had no viable tools to web-enable these reports. Remote users required the SAS System for the PC to access our data.

Litton/PRC personnel, under contract to the CHES, have taken the Institute's web tools and developed the CHES' first web applications. Initially, this was done as an evaluation of the SAS/IntrNet software but once the three applications discussed in this paper were demonstrated to Dr. Optenberg, he saw the potential power of using the web to disseminate information to our users and directed further development take place.

Using just the HTML Formatting Tools, the massive reports can easily be made web-ready, requiring little or no knowledge of HTML. Developing a working application from which a user can interactively produce output does require an understanding of HTML, web browser capabilities, the Internet, and secure data transmission. Working around our normal workload, we developed three simple, but useful applications we wish to share with you.

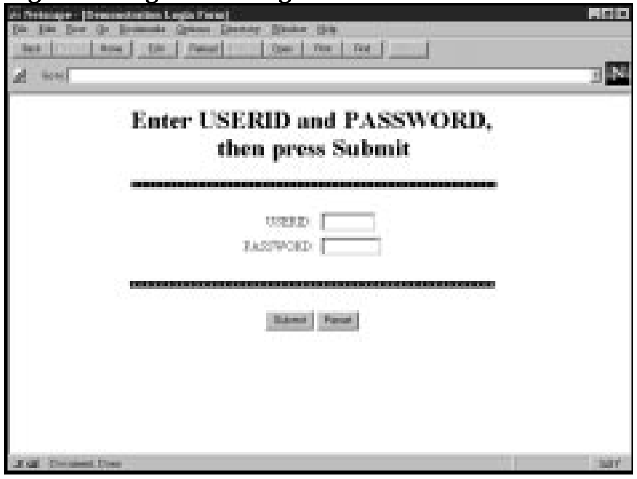
SECURITY

Due to the nature of our projects and the sensitivity of our data, security quickly became a major concern in the development of our web applications. Initially, our application development resided on a web server that we had no administrative control over, forcing us to develop our own SAS security program that performed all user authentication and access control based on information contained in a SAS data set. This is basically how the security program works:

1. The user opens the login page with their browser (Figure 1).
2. The user submits his user id and password to the SAS System via the Application Dispatcher.
3. The SAS program authenticates the user id and password using a SAS data set (Figure 2).
   a. If the authentication fails, an error page is returned to the user's browser.
   b. If the user is successfully authenticated, the first page of the application is returned.

We also restricted access by making it possible to bookmark only the login page and none of the subsequent application pages.

Figure 1. Login Web Page



Enter USERID and PASSWORD, then press Submit

USERID

PASSWORD

Figure 2. SAS Security Program

```
/*---------------------------------------------
+ Compare user id and password to values in the
+ master SAS data set. The macro variables user
+ and pwd are passed to this program from the
+ web page by the SAS/IntrNet application
+ broker. The proc sql will create a local macro
+ variable called sqlobs.
---------------------------------------------*/
proc sql noprint;
   select userid, password
   from libname.sas_master_data_set
   where userid = "&user" and password = "&pwd";
quit;
%macro check;
   /*-------------------------------------------
   + If sqlobs = 0 then no match was found. Send
   + an error message back to the user's browser
   -------------------------------------------*/
   %if &sqlobs eq 0 %then %do;
      data _null_;
         file _webout;
         put 'Content-type: text/html';
         put;
         put '<h1> Login Failed </h1>';
         put '<p>';
         put 'Use the back button on your';
         put 'browser to return to the Login';
         put 'Page';
         put '</html>';
      run;
   %end;
   /*-------------------------------------------
   + A match was found so print out the first
   + web page of the application.
   -------------------------------------------*/
   %else %do;
      /* --- more SAS statements --- */
   %end;
%mend check;
%check;
```

Upon installation of our own independent secure web server, the SAS security program was replaced by "server-level" authentication routines and use of the Secure Socket Layer (SSL), which encrypts and authenticates messages sent between a browser and Sun™ WebServer™ (SWS). We used Sun's bundled routines for creating the private- and public-keys needed to encypt/decrypt data between the user's browser and our web server. The user must use the "https://" command on their browser to communicate through the secure port on our web server, the only port actually available for their use. Once connected, the server will prompt the user to download the public-key and give them options to either download it only for the current session, for the duration of the key pair, or not at all.

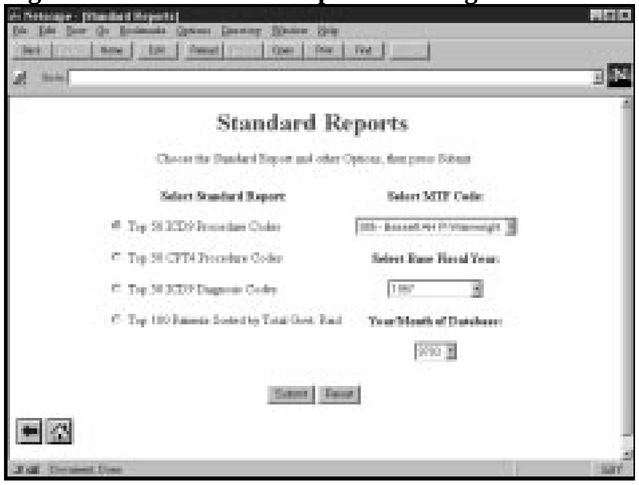
LAUNCH SERVICE vs. SOCKET SERVICE

We elected to use the launch service capabilities of SAS/IntrNet in all of our web applications. This eliminates any possibility of the socket service not being available due to a long running or "hung" process and provides each web application user with their own SAS session. The negative side to using the launch service is that we lose any temporary and log files which would remain during a socket service connection. We also created a separate launch service for each web application by modifying the broker configuration file and adding the appropriate lines for each service. This also necessitates having individual application server start up files (srvauto.sas, reset.sas, and permdata.sas) for each service and gives each programmer control of their respective project.

THE STANDARDIZED REPORT

For our first attempt at producing a web application, we chose to see if we could duplicate the functionality of a SAS System for PCs v6.04 AF program still being utilized by our organization. This turned out to be a relatively simple process once we determined how SAS/IntrNet passed macro variables from HTML to SAS via "name/value" pairs. This application uses a static web page with selection boxes and radio buttons which allow the user to choose the standard report desired, the SAS data set(s) required for the report, and the time period desired (see Figure 3). The relevant part of the HTML code is given below in Figure 4. Note that the macro variable name is assigned in the *name="varname"* statement, and the value assigned to the macro variable is given in the *value="something"* part of the HTML code.

Figure 3. The Standard Report Web Page



Standard Reports

Choose the Standard Report and other Options, then press Submit

Select Standard Report:

- Top 50 ICD9 Procedure Codes
- Top 50 CPT4 Procedure Codes
- Top 50 ICD9 Diagnosis Codes
- Top 100 Patients Sorted by Total Govt. Paid

Select MTF Code: 005 - Bassett AH Ft Wainwright

Select Base Fiscal Year: 1997

Year/Month of Database: 9710

Figure 4. The Standard Report HTML code

```
<HTML>
  <!- -- more HTML code -- ->
<FORM ACTION="/cgi-bin/broker.cgi">
<input type="hidden" name="_service"
       value="stdrpt">
<input type="hidden" name="_program"
       value="libname.sasmacr.stdrpt.source">
<table cellpadding=6>  <!- Table begins ->
  <tr>                 <!- First row    ->
    <th> Select Standard Report: </th>
    <th valign=bottom> Select MTF Code: </th>
  </tr>
  <tr>                 <!- Second row   ->
    <td> <input type="radio" name="report"
                value="t50proc" checked>
         Top 50 ICD9 Procedure Codes </td>
    <td align=center valign=top>
      <select name="cat">
        <option value="005"> 005 - Bassett AH Ft
                             Wainwright
        <option value="008"> 008 - Bliss AH Ft
                             Huachuca
        <option value="031"> 031 - Fitzsimons
                             AMC
        <option value="037"> 037 - Walter Reed
                             AMC
      </select> </td>
  </tr>
  <tr>                 <!- Third Row    ->
    <td> <input type="radio" name="report"
                value="t50cpt4">
         Top 50 CPT4 Procedure Codes </td>
    <th valign=bottom> Select Base Fiscal Year:
    </th>
  </tr>
  <tr>                 <!- Fourth Row   ->
    <td> <input type="radio" name="report"
                value="t50diag">
         Top 50 ICD9 Diagnosis Codes </td>
    <td align=center valign=top>
      <select name="bfy">
        <option value="1997"> 1997
        <option value="1996"> 1996
        <option value="1995"> 1995
        <option value="1994"> 1994 (CPR only)
        <option value="1993"> 1993
      </select> </td>
  </tr>
  <!- -- more HTML code -- ->
</table>               <!- Table Ends   ->
  <!- -- more HTML code -- ->
</FORM>
</BODY>
</HTML>
```

One of the nice things about this particular project is that each of the SAS programs used to produce the standard reports are actually macro programs. This enabled us to store the programs as compiled macros in a macro catalog with one exception. When the above page is submitted, it calls the program STDRPT.SOURCE. This program declares all the global macro variables and then calls the first compiled macro. The macro used is determined by the report requested. Two print macros are also stored in the same macro catalog. One of these produces a negative report, and the other returns the data requested by the user to the user's browser. The code for this macro is given in Figure 5 and uses the SAS supplied Output Formatter macro (out2htm) as well as macro variables created by the web page and the compiled standard report macros.

Figure 5. Macro Code to Print Standard Report

```
%macro printit;
  %out2htm(capture=on);
  proc print data = %superq(datset)
             %superq(datopt) noobs split='*';
    options missing = '0';
    id %superq(idvar);
    var %superq(printvar);
    sum %superq(sumvar);
    title1 "U.S. Army Managed Care Program";
    title2 "%superq(titl2)";
    title3 "%superq(titl3) %superq(srtid)";
    title4 "Fiscal Year %superq(bfy)";
    title5 %qcmpres(&camname.);
    footnote1 "Source: Tri-Service CHAMPUS
         Statistical Database";
    footnote2 "Medical Analysis Support System
         (MASS)";
    footnote3 "Center for Healthcare Education
         and Studies";
    footnote4 "Ft. Sam Houston, TX 78234-6125";
    footnote5 "Report: %superq(report), Nov 97,
         SLW";
  %out2htm(capture=off, htmlfref=_WEBOUT,
         runmode=s, openmode=%superq(opmode));
  run;
%mend printit;
```

THE SURVEY PROJECT

The survey project presented some interesting challenges. Since the actual survey we wanted to place on the internet has over a hundred questions, we decided to break the survey down into smaller sections. This meant we had to find a way to update the SAS data set which stores the survey data in pieces, i.e., we needed to track the user who was entering the data and the patient id for the observation being entered. Tracking the user was simple. When a person logs onto our web server using the built-in user authentication program, the SAS/IntrNet application broker stores the user's name in a macro variable called _RMTUSER. This variable can be referenced by any SAS program called during the user's session.

Tracking the patient id required a little more thought since we are using launch service to run our applications (see Launch Service vs. Socket Service). The patient id is entered on the first page of the survey along with other demographic information for that patient. When that page is submitted to the SAS program, the master SAS data set is updated, and the second page of the survey is sent back to the user's browser. Rather than require the user to enter the patient id on every page, we chose to keep track of it ourselves by sending the patient id back to the user's browser in a "hidden" type of input field. Figure 6 shows an excerpt of the code we developed to handle this situation. In this example, we are using SSN as the patient id.

Figure 6. Passing a Value to the User's Browser

```
/*-------------------------------------------------
+ This data step stores the userid and ssn
+ entered on the web page into SAS variables
-----------------------------------------------*/
data temp;
  length userid $ 8 ssn $ 11;
  userid  = upcase(symget('_rmtuser'));
  ssn     = symget('wssn1') || '-' ||
            symget('wssn2') || '-' ||
            symget('wssn3');
  /* --- more SAS statements --- */
run;
/* --- more SAS statements --- */
/*-------------------------------------------------
+ This data _null_ outputs page 2 of the survey
+ back to the web, passing ssn in a hidden field
-----------------------------------------------*/
data _null_;
  set temp (keep = ssn);
  file _webout;
  put 'Content-type: text/html';
  put;
  /* --- more SAS statements --- */
  put '<FORM ACTION="/cgi-bin/broker.cgi">';
  put '<input type="hidden" name="wssn"
            value="' ssn +(-1) '">';
  /* --- more SAS statements --- */
run;
```

When the second page of the survey is submitted to the SAS program, the patient id (ssn) will be stored in the macro variable WSSN, and the user's id will still be in the automatic macro variable _RMTUSER. All subsequent SAS programs can now update the master SAS data set by userid and ssn.

This application does not produce any standard SAS output. It is a data entry, on-line transaction processing application and as such, SAS/SHARE® software is required for concurrent access and to avoid corruption of the data. How to run SAS/SHARE is beyond the scope of this paper, and it is mentioned here simply to inform the reader of its necessity for this type of application.

AD HOC REPORTING

The Ad Hoc Report program is one of several modules from a larger web application which includes both standard reports and graphs. To invoke the Ad Hoc Report module, the user enters the web site through an HTML menu selection page. Upon selecting the Ad Hoc Report option, a SAS/SCL program creates the Ad Hoc Report HTML page. While this action requires additional processing time and subsequently a longer wait time for the returning page, we chose this method because we wanted the SAS System to generate the HTML page dynamically and the data set contents are subject to change.

The application's menu page uses a very short SAS/SCL program to call the Ad Hoc Report SAS/SCL program. The Ad Hoc Report program reads the contents of the target data sets, eliminates any variable whose length is greater than 30 bytes, and sorts them according to their position in the data set. This data set, called contents, is used through out the SAS/SCL program to create HTML SELECT lists.

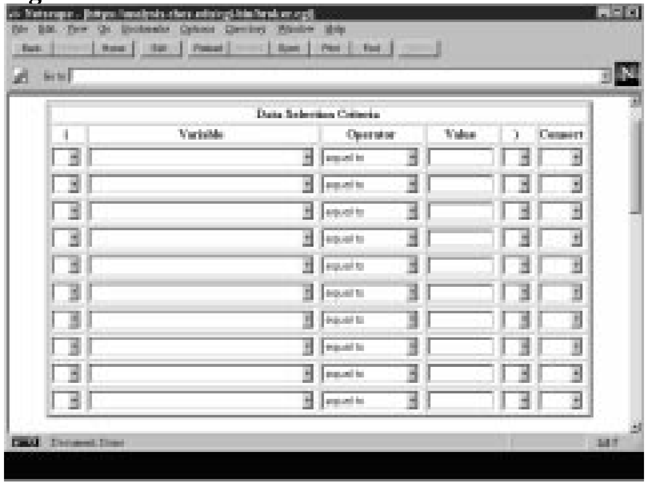
First, it creates 10 lines of WHERE statements. Each WHERE statement consists of an HTML SELECT list for the left parenthesis, variable, logical operator, right parenthesis, and logical connector. The variable's value field is a free text INPUT field. While not all 10 lines of WHERE statements or every element of each line are used on each ad hoc report, they are available for use. The SAS/SCL code in Figure 7 shows how the WHERE clause lines are constructed.

Figure 7. Building the WHERE Clause

```
/*-------------------------------------------------
+ Open the new content data set and validate
+ that it is open and available for use. If it
+ is open, then find  out how many observations
+ it has and the variable numbers of the
+ variables NAME and LABEL.
-----------------------------------------------/
dsid = open('content','I');
if dsid ^= 0 then do;
   numobs = attrn(dsid,'NOBS');
   namevar = varnum(dsid,'name');
   lablvar = varnum(dsid,'label');
   /*-------------------------------------------
   + This loop is repeated 10 times creating the
   + lines for the WHERE clause statements.
   + Again, it is sent only to the PREVIEW
   + buffer and not to SAS for processing.
   -----------------------------------------*/
   do j = 1 to 10;
      submit;
         put '<TR>';
         put '<TD><SELECT NAME="l_par&j">';
         put '<OPTION SELECTED VALUE="None">';
         put '<OPTION VALUE="("> ( ';
         put '<OPTION VALUE="(("> (( ';
         put '</SELECT></TD>';
         put '<TD><SELECT NAME="var&j">';
         put '<OPTION SELECTED VALUE="None">';
      endsubmit;
      /*----------------------------------------
      + Using the number of observations in the
      + content data set, another loop is
      + processed. This loop reads in the NAME
      + of the variables to be used in a HTML
      + <SELECT> tag.
      --------------------------------------*/
      do i = 1 to numobs;
         rc = fetchobs(dsid, i );
         v_name = getvarc(dsid, namevar);
         v_labl = getvarc(dsid, lablvar);
         submit;
            put '<OPTION VALUE="&v_name">
                  &v_labl';
         endsubmit;
      end;
      /*----------------------------------------
      + Loop has ended, time to complete the
      + where clause statement.
      ----------------------------------------*/
      submit;
         put '</SELECT>';
         put '</TD>';
         put '<TD><SELECT NAME="op&j">';
         put '<OPTION VALUE="eq"> equal to';
         put '<OPTION VALUE="gt"> greater than';
         put '<OPTION VALUE="lt"> less than';
         put '<OPTION VALUE="ge">
              greater or equal';
         put '<OPTION VALUE="le">
              less or equal';
         put '<OPTION VALUE="=:"> is like';
         put '<OPTION VALUE="not eq">
              is not equal';
         put '<OPTION VALUE="in"> is in';
         put '</SELECT>';
         put '</TD>';
         put '<TD><INPUT TYPE="text"
              SIZE=10 NAME="val&j">';
         put '</TD>';
         put '<TD><SELECT NAME="r_par&j">';
         put '<OPTION SELECTED VALUE="None">';
         put '<OPTION VALUE=")"> ) ';
         put '<OPTION VALUE="))"> )) ';
         put '</SELECT>';
         put '</TD>';
         put '<TD><SELECT NAME="logop&j">';
         put '<OPTION SELECTED value="None">';
         put '<OPTION VALUE="and"> AND';
         put '<OPTION VALUE="or"> OR';
         put '</SELECT>';
         put '</TD>';
         put '</TR>';
      endsubmit;
   end;
```

The code from Figure 7 produces the portion of the web page displayed in Figure 8.

Figure 8. The WHERE Clause



Data Selection Criteria

(	Variable	Operator	Value	)	Connect
		equal to			
		equal to			
		equal to			
		equal to			
		equal to			
		equal to			
		equal to			
		equal to			
		equal to			
		equal to			

Second, it provides an HTML selection list for variables to keep and allows the user to select up to three sort variables as well as different sort options (ascending, descending, nodup, and nodupkey). It allows for free text entry of up to five title and five footnote statements.

Once the user has defined the desired data requirements, the information is sent to the SAS System for processing by a third SAS/SCL program. It is important to note a SAS/SCL program receives the "name/value" pairs created by the Application Dispatcher as a named SCL list and the list elements are treated as character strings.

Before the output can be returned to the browser, extensive validation must be carried out to construct the appropriate WHERE clause. This is especially true when working with parenthesis, the IN operator, and determining data types.

If any errors are found, the program creates a HTML page identifying what the error(s) is/are, sends a dynamic HTML page back to the browser, and halts program execution. Otherwise, the program produces the desired output page in HTML format using the Institute's Output Formatter.

THE OUTPUT FORMATTER vs. THE DATA SET FORMATTER

We used the output formatter to produce the report in the Ad Hoc Query for one simple reason. When we tested the data set formatter with a data set that contained 694 observations and 14 variables, it produced an HTML page that was *eight times* the size of the output generated by the output formatter (452K to 56.5K)! While we preferred the appearance of the output created by the data set formatter over the output formatter, we realized most of our users would have difficulty loading such a large file into their browser. The difference in size can be attributed to all the additional HTML tags the data set formatter uses to create web output.

CONCLUSION

Using the Institute's web formatting tools, the U.S. Army Center for Healthcare Education and Studies will be able to provide easier access to their multitude of reports. In addition, web applications which utilize the SAS/IntrNet Application Dispatcher will provide remote users the ability to create their own unique reports. We have taken advantage of this technology by developing three different types of web applications: Standard Reports, The Survey, and the Ad Hoc Report.

ACKNOWLEDGMENTS

The authors wish to thank the U.S. Army Center for Healthcare Education and Studies, Analysis Branch, Fort Sam Houston, Texas, for purchasing the SAS/IntrNet product and providing us the resources and flexibility to explore its features. We also extend thanks to them for allowing us to take time from our busy schedules to present this paper at SUGI 23.

AUTHORS

Steven L. Willhoite, John P. Campbell, and George R. Orr are employees of:

Litton/PRC Inc
16500 San Pedro Ave, Suite 302
San Antonio, Texas

contracted to the U.S. Army Center for Healthcare Education and Studies, Analysis Branch, Fort Sam Houston, Texas, under GSA Contract #GS35F4340D.

They can be reached at (210) 221-9333 or via e-mail at:

Steve.Willhoite@cs.amedd.army.mil
John.Campbell@cs.amedd.army.mil
George.Orr@cs.amedd.army.mil