Open the Information Window of Your SAS® Data Warehouse
A WEB Information Center Powered by SAS®

Di Meng
Cigna Healthcare, Chattanooga TN

ABSTRACT

This paper describes how to use SAS's new WEB Tools along with the HTML and PERL languages to build a information center which provides information on our SAS data warehouse to our SAS users. The Information Center provides public access on our company Intranet and delivers real time information to our SAS users in a user-friendly environment.

Three major components will be discussed in detail in this paper.

1. HTML pages as front-ends for our SAS users.

2. CGI programs as application brokers.

3. SAS entries with SAS WEB Publishing Tools as a driving engine for this application.

INTRODUCTION

Our SAS data warehouse is serving over 170 SAS users across the company, which includes 16 healthcare plans. The way that our data warehouse is structured, which is multiple plans and multiple business subjects, complicates our means of communication. The WEB pages provide real time information which includes a introduction to our SAS data warehouse, the schedule and actual dates of DW update, current table layouts and field attributes, on-line validation reports, etc.. An on-line support page let users send questions and feedback, report problems and submit special requests.
The application consists of HTML pages, CGI Perl scripts and SAS programs to generate real time WEB reports.

METHODOLOGY

WEB applications reach end users by providing WEB pages through WEB browser such as Netscape browser. WEB pages are written in HTML, the HyperText Markup language. They are static and isolated, which means that once a HTML page pops up on screen, it has lost connections with any other pages and the program that generated it. To obtain any further information through WEB application, we have to activate the WEB pages by using dynamic HTML that generated by Perl scripts and SAS applications.

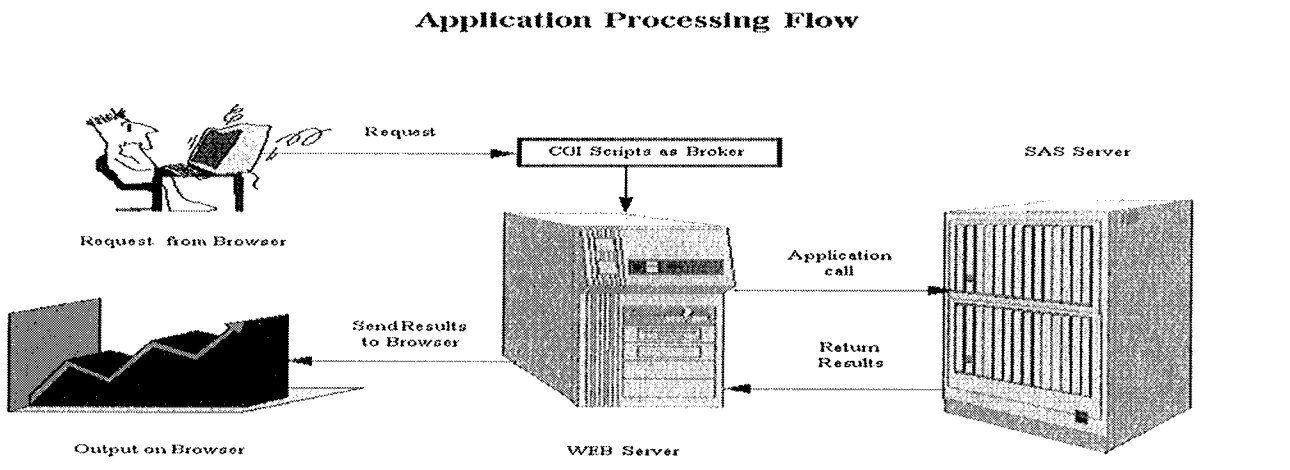

Figure 1.

Figure 1 describes the processing flow of our Implemented methodology.

1. WEB Pages for User's Requests

Even though HTML pages are static, they are where a WEB application starts and ends. HTML pages collect user's requests through user's inputs, then pass the information to the application broker and finally take the results and present them to users(see Figure 2 for a sample WEB page).

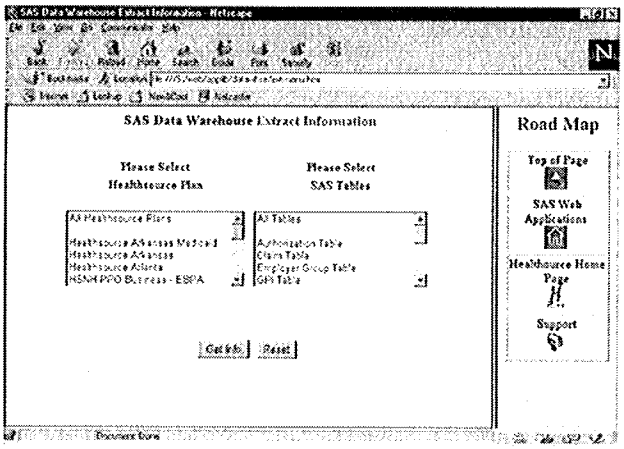

Figure 2.

This page is one of our information request pages. It askes users to select the data warehouse and the tables that they are interested in. After all the 'behind-the-scene' work, the next page the browser presents to the users is supposed to be a output report.

Once the user click the submit(Get Info) button, it will pass all the parameters it collected to the CGI application broker, and the job for this page is done.

2. The CGI Application Broker

The CGI program takes the requests from the user and sends back valid and useful responses to the user by presenting the results on a HTML page. It can understand the information passed by a WEB page, organize the requests in your WEB server environment and contact the application server to get the results for the requests. The following PERL script (Example program 1) shows how the CGI program takes the requests we collected from Figure 1 and gets results back.

```
#!/usr/local/bin/perl
#
# datawhse.pl
# Submit Job to write a response html page to the calling browser
##################################################################
do "/usr/local/bin/cgi-lib.pl"        || die "Could not load cgi library";
do "/web_users/medecon/cgi-bin/utility.pl"
 || die "Could not load UTILITY Library";

#####################################################
# Use Read Parse to Create Array of Parameters
#####################################################
&ReadParse;

#################################################################
# Retrieve Necessary Paramaters from Array
#################################################################
$query    = $in{'query'};
$plan     = $in{'plan'};
more statements;

#################################################################
# Write a SAS program to start SAS and a script to call it
#################################################################
$saspath = "/sas/sas612";
more statements for preparation;

open(sasfile, ">$outfile") || die ("Unable to open output file $outfile
\n");

print sasfile ("/*******************************************/ \n");
print sasfile ("/* SAS program to convert shell vars to   */ \n");
print sasfile ("/* macro vars and start sas process       */ \n");
print sasfile ("/*******************************************/
\n\n");

if ($action eq "Extrinfo") {
        $line = "filename runprog '" . $progpath . "/extrinfo.sas';";
  }
more elseif etc.;
elsif ($action eq "Validate") {
        $line = "filename runprog '" . $progpath . "/validchk.sas';";
  }
print sasfile ("$line  \n\n");

more statements;

$line = "%include runprog;";
print sasfile ("$line  \n\n");

close (sasfile);

$statement="su webuser -c \"rcp $outfile sasuser\@namov:$spath\" ";
system($statement);

more statements;

$statement="su namov -c \"remsh sasbox -l me $saspath/sas -sysin
                $rempath/$sasprog $opts $autoexec $config $log $rpt\"
";
system($statement);
```

Program 1

First part of this script is to load the CGI library and to read and parse the parameters passed by the WEB page just like all the other CGI programs do.

The core of this script is to organize the information, to generate the call programs to the application server and to dispatch the application jobs.
It prepares all necessary parameters such as SAS path, options for SAS session and so on.
Then it writes a SAS "call" program which will assign necessary SAS variables, for example macro variables, based on the requests and includes the existing SAS application program into the process.

The final part of the CGI program is to issue WEB server system commands to send the "call" program over to SAS server and to start a SAS session which will run the included SAS application program and return the results back to the WEB server.

3. SAS Application Programs

These SAS programs, which have been prepared and stored in SAS libraries, take the commands from the CGI application broker and perform requested processing. They generate the results and write the results out as internet contents which are sent back to the browser in a HTML page via WEB server.

All our SAS programs have similar structures like the one in Example Program 2 which has three sections: Prepare output data; Write out HTTP header; and Call SAS WEB publishing tools to create HTML contents.

3.1 Data Preparation

This section of codes works no different from a regular SAS program. In this section, the program manipulates SAS data to generate the results based on the requests that it takes from the CGI broker. The results we get by the end of this section are still in SAS format. So they are not publishable on a WEB page. We need some tool to convert the results into WEB publishable contents.

3.2 The HTTP Header

The HTTP Header determains how your WEB (HTTP) Server works with your WEB Browser and the CGI programs. When users send a request from a WEB page, the browser gathers the information and passes it to WEB server through a HTTP request header.

On the other hand, after the CGI broker calls the application(SAS) programs to perform some function and return the results, we need to tell the WEB server what type of response is being sent back to the server. Since we are going to send a HTML document back to the WEB server in order to publish it on the user's browser, we tell the server that the content type is HTML text before we deliver the actual document.

```
*****************************************
* Data Preparation Section                *
*****************************************
PROC SQL;
  create table work.data as
  select  some variables
  from sample table;
QUIT;

more data preparation steps;

*****************************************
* Write out the HTML header               *
*****************************************
data _null_;
  file STDOUT;
  put "Content-type: text/html";
  put ;
run;

*****************************************
* Use SAS WEB Tools to Generate the Contents  *
* of the results                          *
*****************************************
title '<CENTER><FONT FACE=ARIAL
            SIZE=5 COLOR=BLUE><B>'
      Data Warehouse Extract Information for sample table
      '</B></CENTER><FONT>';

 %ds2htm(data=work.data
            var=sample variables,
            vhalign=left,
            bgtype=color,
            bg=cyan,
            cspace=4,
            id=id_var,
            ihalign=left,
            icolor=blue,
            isize=3,
            clhalign=left,
            clcolor=blue,
            clsize=3,
            encode=n,
            vsize=2,
            vwrap=n,
            htmlfref=STDOUT,
            formats=Y,
            labels=Y,
            obsnum=N,
            border=n,
            openmode=replace)
            ;
```

Program 2

3.3 The SAS WEB Publishing Tools

Everything is ready up to this point. We now need to write all the results line by line into a HTML document. SAS has developed very powerful WEB publishing tools to help us to accomplish this task. The macro call in the example program is one of them. It converts a SAS data set into a HTML document.

There two ways that you can create a dynamic HTML.

One is to send the HTML document to Standard Output which will be sent back to the WEB server directly. This is the case in our example program. The final thing this SAS program does is to convert the SAS data into a HTML document and send it to STDOUT.

Another method is to create HTML files and then send another HTML document to Standard Output to include them. For example, in our application, when we try to combine several pages into a frameset , we have to take this approach.

It is so easy to switch from one mode to another. The only thing needs to be changed is the macro parameter 'htmlfref'. When it is set to equal 'STDOUT', the HTML document will be sent to Standard Output. If it is set to a file reference, then the document will be written out to the file.

4. Final Results

Figure 3 shows the final WEB page which contains the results returned from the request user submitted from the WEB page in Figure 2.

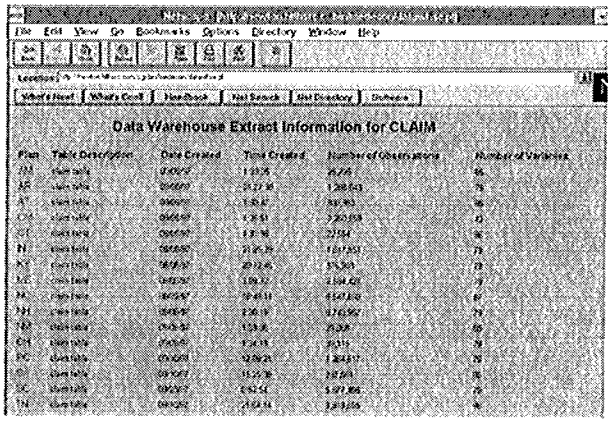

Figure 3.

From Figure 2 to Figure 3, it takes only 2-3 seconds to the users. The work is done by powerful SAS engine and convenient SAS WEB publication tools.

CONCLUSIONS

The WEB application that we have discussed here serves as an open window of our SAS Data Warehouse to our SAS user community.
A data warehouse is essentially a central database that is loaded with information from multiple data sources for the purpose of end-user access. A well established communication channel between the Data Warehouse Administrator and our SAS users is fundamental to us. The methodology we developed here provides us a powerful administration tool to better maintain our SAS Data Warehouse and serve our SAS user community.

ACKNOWLEDGEMENTS

I would like to acknowledge Charlie Bastnagel who initiated SAS WEB application development in our company.

For Further Information Contact:

Di Meng
Cigna Healthcare
2 Fountain Square 5-N
Chattanooga, TN 37402

mengd@hlthsrc.com