PAPER 38-27

ASK DSS…USING HTML, JAVASCRIPT, HTMSQL, AND SAS® TO CREATE DYNAMIC WEB APPLICATIONS

Karen L. Wyland, Decision Support Systems, Sodexho, San Bruno, CA

ABSTRACT

The Decision Support Systems web site promises to deliver information on demand. How do you do that? More specifically, how do you meet the requests of many users who span all levels of the organization and need different levels of data? How do you optimize the user experience that is dialing over a 28.8 modem?

"Ask DSS", partially modeled after "Ask Jeeves" is one such application. All users choose from a prearranged set of questions. Each question has several pull-down boxes to let users tailor it for their particular inquiry. We term this a "predefined where clause with variable values." It's easily maintained because each question has its own SAS program called from the SAS/IntrNet® broker. Macro code manages the dynamic decision making of which user and what data, and JavaScript within the web page manages the questions submitted and the necessary parameters to pass.

INTRODUCTION

The Decision Support Systems (DSS) web site presents primarily financial and demographic data for Sodexho to a large variety of users including financial analysts, operational management, and top-level executives. The data represents ~7500 operational stores. Demographic information, market segmentation, monthly general ledger, and various summarized or categorized datasets detail each operational store. The data is subset into nine divisions within the company. Some queries go against the entire company. These datasets can be as large as 750,000 observations.

Because we are restricted by dial-up connection speeds, anywhere from 28-56 kps., we need to optimize web applications so that the amount of data displayed for selections or returned as output is small enough to produce an acceptable wait time for the user.

Further, we're restricted in that we don't have a company Intranet networked for national access. This means encrypting all transmissions and competing with traffic over the World Wide Web. We're also limited in implementing complicated security access rules. We use logon authentication and access rights by our web server, Netscape Enterprise Server. We also use SAS 6.12 and can't take advantage of some of the new security features in version 8.

Lastly, our team is very small -- one developer/administrator and two support staff. In addition, we administer the server that houses all the applications remotely which can be quite challenging (another paper to come on that one).

In summary, we had to find creative ways in which to use the existing tools of the SAS/IntrNet. Many of the tools provided work fine "out of the box". Other SAS/IntrNet users out there know you always customize for your specific data and needs. Since much of the SAS/IntrNet code is compiled scl and macro code, it is often difficult to edit and test. We can maintain and revise our own code much easier, so we coupled existing tools with JavaScript and SAS macros. The result is an efficient and effective experience for the end user and maintainable code.

DEVELOPMENT METHODOLOGY

There are four development tasks to this application. They are distinct in purpose and coding and require development skills in all areas. Yet, most importantly, remember each one is dependent upon the other for the success of the application.

- Design/layout of the web page.
- htmSQL coding within the web page.
- JavaScript coding within the web page.
- SAS programming code to handle the submission of parameters (name-value pairs) to the SAS broker and return output to the browser.

The following subheadings contain some overall comments and helpful hints about these four tasks. Following that, we'll take you through how we built and coded the web page for one of the questions described by detailed and commented code examples.

Design/Layout

We designed the page using FrontPage 2000. We wanted to see how to fit appropriate questions on the page as well as develop the form and object attributes that we needed. It was truly a work in progress. We started with one question and got the htmSQL code to return values into the select boxes. Then we had to figure out what parameters to pass to a SAS program. Of course, this means knowing what the program will do and that means coding one. We could not fully complete one of the four tasks without the others being involved. We bounced back and forth between them all, tweaking and editing as we went.

We used FrontPage because we are familiar with using it for the initial layout and editing the HTML code to meet our needs. We do not use the server extensions or any FrontPage add-ins products. Any HTML editor will do.

htmSQL Coding

Once we designed the form, we coded the SQL that dynamically returns the values of the variables in the select boxes from SAS/IntrNet. We've hard coded some selections because the data is not available in our warehouse.

The htmSQL coding took quite a bit of trial and error since you cannot see the results until you move the page to the server and test it. We suggest testing the SQL code in SAS to check syntax and expected output. Another helpful tip is to use the automatic sql variable &sys.query that produces the text of the last SQL query processed. The output shows in the source of the web page.

JavaScript Coding

The next and most difficult step was writing the JavaScript to handle multiple queries under one form. We could have

used multiple forms, but would then have to manage which form is being submitted. That seemed more difficult. We checked the form, made of several questions, by an in-page JavaScript function (see Code Samples). When the user clicks on the "Ask" button next to the question they desire, it is a submit function and sends the parameters of the form to the JavaScript function. Each question has its own "if" logic to decide the correct URL and parms to pass to the SAS Broker based on the question requested.

Another difficulty is handling whether the user wants to query against the entire company data or subset it by division. Because our JavaScript skills are limited and we have to nest this selection with the other functions, we simply pass the parameter to the SAS broker and make the determination in a SAS program.

SAS Programs

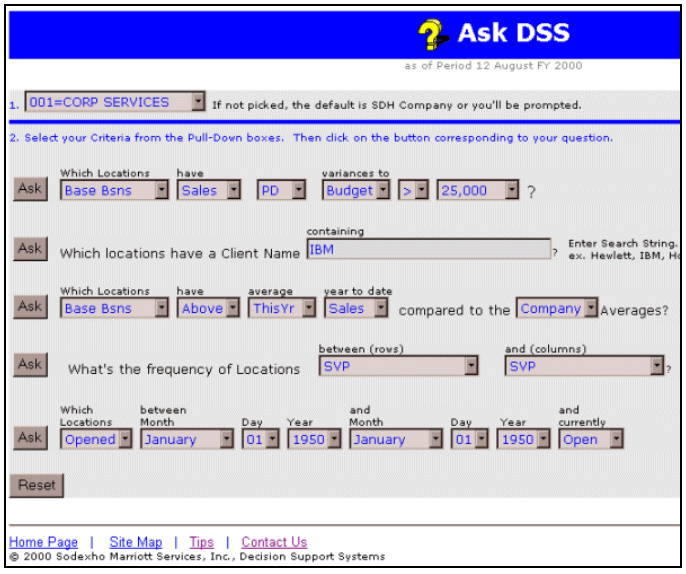
We had a lot of fun with the SAS programs since we were in our element. If you're comfortable with SAS/Macro code then you'll find it easy to take a prescribed set of parameters and work your way through the logic to get the desired output. We used Proc SQL and SAS/Macro to accomplish most of it. We used the SAS/IntrNet formatters in their default states. This paper only covers one of the questions. We'll be glad to share others.

LET'S GET TO IT

The screen capture below shows the web page as a user sees it. Notice there are two sections. Section 1 is for the user to select the level of data to query. Section 2 asks the user to choose their criteria from the drop down boxes and then click on "Ask" next to that question. Take a moment to familiarize yourself with the page.

User Interface - Selection Page

What the User Sees in the Browser:



Components / Features

- Current "as of date" of data displayed.
- Section 1. Global select box applies to all questions on form. User queries entire company or subsets by division.
- Section 2. Five unrelated questions with select boxes of dataset/macro variables so user can customize query. Possibilities are limited only by available data and programming skills.
- Multiple form submissions work on one page. JavaScript handles this portion.
- You can add unlimited number of questions.
- Each question requires a JavaScript function, a SAS program, and maybe htmSQL code.

Refer to the graphic you've just examined, User Interface - Selection Page. We're going to describe the various code parts of Question #1 as well as the dynamic date at the top of the page and the Global Selection Box. It'll follow in order of the page layout as we describe the code and indicate what kind of coding is occurring at each step.

What we don't show or discuss is installing/configuring the SAS/IntrNet software, SAS/SHARE® Server, HTML codes (head, body, fonts, etc.), or how to program in SAS. We assume that the user has experience with SAS/IntrNet, basic HTML and JavaScript, and base SAS.

SAMPLE CODE AND EXPLANATIONS

Please note:

Bolded text within the sample code is there to emphasize the use or syntax of a particular code bit.

Connection to Server

To use htmSQL all pages must have a query to the SAS/SHARE server. We place it at the top of the page to make sure its active for all queries in the page.

```
<!--{query server="xyz.com:6913"}-->
```

(:6913 is the port no. assigned to the SAS/SHARE server.)

Dynamic Date

We wanted to include the “as of date” dynamically at the top of the web page to indicate the period of time. We have a dataset that stores various environment and system parameters specific to our data and users. This allows us to retrieve the values with htmSQL. We also use this data to create SAS Macros within SAS programs when needed.

```
<!-- {sql}
select period, put
(pd_mo_1,worddate9.) as month, fy_t
from sasdl.consol ;
{/sql}-->
```

What the User Sees in the Browser:

as of Period 9 May FY 2001

Global Selection Box

Since users can choose to query the entire company or subset the data by division, we needed what we termed a "global" selection box. It affects each question, but we didn't want it to appear as a selection, repeatedly, for each question. It's more of a design issue for the user's benefit. Rather than do the checking in the JavaScript function, we pass it along, missing or not and test for conditions in SAS.

What the User Sees in the Browser:

Place the htmSQL code prior to the select box HTML code. This will query the server for the unique value of each level of data (**SVP**) into the option list of the selection box when the page is loaded.

```
<!--{sql}
select distinct compbl
(put (svp,z3.) || '=' || put (svp,_nsvp.))
as svp from sasdl.consol;
{/sql}-->
```

Now create the select box in HTML code and incorporate the htmSQL macros that will resolve and populate the option values in the select box. Notice how we use the **SVP** variable from the select statement above in the eachrow section.

```
<select name="svp" size="1">
<option>Select a Division
{eachrow}
<option value= "{&svp}" > {&svp}
{/eachrow}
</select>
```

Question #1

The first question lets a user request output for each operational store for the entire company or subset by division. The user will customize the output by selecting values in each drop-down or select box. We term this a "predefined where clause with variable values". Any level of complexity can be achieved by "knowing your data" and efficient and effective SAS programming.

What the User Sees in the Browser:

A user may select any number of values for each variable. The written words above the select boxes help the user to verbalize the correct question to return the output desired.

To put the query into words, this question lets a user ask for a listing of operational stores which are categorized as base business (opened more than 2 years) where Sales for the Period are over Budget by more than $25,000. That's the default if the user just clicks on the "Ask" button. Notice in the sample code below that the options in the select box have a "selected" value so that we can pass default parms to the SAS program.

All code for the select boxes is within a <form> object.

The _onClick = "getQ1()" is the action performed when a user clicks on the "Ask" button and passes the parms to the JavaScript functions.

Sample Code:

```
<input type="button" value="Ask"
_onClick = "getQ1()">
```

Place the htmSQL code before the select box to return unique values as options. We demonstrate for **rank**.

```
<!--Rank------------------------>

<!--{sql}
select distinct put (rank,$rank.)) as
rank from sasdl.rank;
{/sql}-->

Which Locations<br>
<select name="rank" size="1">
<option>
{eachrow}
<option value= "{&rank}" > {&rank}
{/eachrow}
</select>

<!--Cat ---------- cat=s ---------->

have<br>
<select size="1" name="cat">
<option selected value="s">Sales
</option>
<option value="f">Food</option>
<option value="l">Labor</option>
<option value="c">Cntl</option>
<option value="n">N-Cntl</option>
<option value="o">OPC</option>
</select>

<!--Time --------- time=pd ------->

<select size="1" name="time">
<option selected value="PD">PD
</option>
<option value="YTD">YTD</option>
</select>

<!--Level---------- level=b ------>

variances to<br>
<select size="1" name="level">
<option selected value="b">Budget
</option>
<option value="l">LastYr</option>
</select>

<!--Sign --------- sign=gt ------->

<select size="1" name="sign">
<option selected
value="gt">&gt;</option>
<option value="lt">&lt;</option>
<option value="eq">=</option>
</select>

<!--Amt ---------- amt=25000---------
>

<select size="1" name="amt">
```

```
<option value="0">0</option>
<option value="10000">10,000</option>
<option selected value="25000">25,000
</option>
<option value="50000">50,000</option>
<option value="-10000">-10,000</option>
  . . . . more options.
</select>

<!--End of Select Boxes for Q1 -->
```

Form Handling - JavaScript Functions

Now that we've created the select boxes and where appropriate populated with values using htmSQL, we need to focus on the JavaScript functions to handle passing the user's choices to the SAS/IntrNet broker. There is a JavaScript function for each question because each one will call a different SAS program and pass the appropriate name-value pairs.

We placed the code in the <head> section of the web page. Review the code and notice that its sole purpose is to get the values selected from the select boxes (document.myform.xxx.value) and join that value to the string needed by the URL to pass the name-value pairs with the correct syntax. The final statement in the function joins all the vars created together in one long string to pass to the URL or broker, (parent.location.href = url).

Sample Code:

```
<script language=javascript>
   function getQ1()
   {
   var path ="cgi-bin/broker?";
   var prog = "_program=webpgms.askdss1.sas";
   var serv = "&_service=default";

   var svplab = "&svp=";
   var svpval = document.myform.svp.value;
   var svp = svplab + svpval;

   var ranklab = "&rank=";
   var rankval = document.myform.rank.value;
   var rank = ranklab + rankval;

   var catlab = "&cat=";
   var catval = document.myform.cat.value;
   var cat = catlab + catval;

   var levellab = "&level=";
   var levelval= document.myform.level.value;
   var level = levellab + levelval;

   var timelab = "&time=";
   var timeval = document.myform.time.value;
   var time = timelab + timeval;

   var signlab = "&sign=";
   var signval = document.myform.sign.value;
   var sign = signlab + signval;

   var amtlab = "&amt=";
   var amtval = document.myform.amt.value;
   var amt = amtlab + amtval;

   var url = path +  prog + serv  + svp + rank + cat
   + level + time + sign + amt;
   parent.location.href = url;
   }
   </script>
```

Debugging Tip

Add an htmSQL code section at the bottom of your page to return the text of the last query produced into the web page. It will not show to the user if you use the comment tags <!-- and --> to surround it. You can then view the resultant sql query that occurred when the page was loaded. This is necessary for testing.

```
<!--<pre>
This is the query produced by your
selections:{&sys.query}
</pre>-->
```

Saving the Web Page

When you're all done with the web page, save it with an extension of .hsql. The SAS htmSQL cgi program requires this. Move it to the server into an html docs directory that your server can access. Then, to access the page and have the htmSQL resolve, you'll need to use the cgi-bin/htmSQL prefix when calling the page in the browser so that the htmSQL code will resolve in the page when it loads. This does not mean the page is stored physically in that directory. Please review the htmSQL documentation on the SAS web site for more details.

SAS Program

Now it's time to have fun. At last, we're back in SAS! So, we've got many name-value pairs coming to us and some decisions to make about what to give back to the browser. This isn't so difficult, since we've already designed the program at least in concept. Remember -- you had to know what you could handle on the SAS side before you could finish with the web page.

When Question #1 was developed, we knew which dataset to use. We also knew the available variables. This helps us analyze a user's request. We all know that's pretty simply to accomplish with basic SAS concepts. However, to help you focus on the extras we want to provide, we'll list them here. As you review it, examine the screen print of the output in the next section, Returning the Output, so you can see where we're going and why.

1. Display the total number of operational stores for a particular rank of the business as compared to the total number overall for the entire company.
2. Display the total number of operational stores found based on subsetting criteria such as, sales for the period and a variance to budget greater than $25,000.

The complexity of these two tasks is in finding and storing the counts of observations at both the company and division level that meet the criteria. We also wanted to translate the variable names or macro names into words to use for printing in the title of the output.

The program below is in its entirety. You'll notice many comments that we try to include for each step. What you won't be familiar with is our dataset, variable, or macro names. The bolded text is again, for emphasis.

Sample Program Code:

```
/*-------------------------------------
     Program:  ASKDSS1.SAS
     Author:   Karen Wyland MAR2000
-------------------------------------/*
```

```
*--Name value pairs passed by broker--;
%global rank svp cat level time timevar sign
amt;

*-- Use 'time' parm passed to determine which
var to use, Current Pd or YTD --;

%macro time;
  %if &time = YTD %then %let timevar = _;
  %if &time = PD %then %let timevar = &zp;
%mend;
%time;

*-- Use 'sign' parm passed to create format to
use in SQL order by statement--;

proc format;
     value $ord
'gt' = 'desc'
'lt' = ' '
'eq' = ' ';

*-- Use 'svp' parm to create libname macro var
for level of data.  If missing, default is
company level--;

data _null_;
  if substr("&svp",1,1) eq . then
   do;
     call symput('svp','000=SDH COMPANY');
     call symput('svplibs','sasdl');
   end;

  else call symput ('svplibs',
put(substr("&svp",1,3), $svplibs.));
  run;

*-- Create macro vars for use in titles--;

data _null_;
  call symput('tvar1',put("&rank",$rank11.));
  call symput('tvar2',put("&cat",$cat.));
  call symput('tvar3',put("&level",$levels.));
  call symput('ord',put("&sign",$ord.));
  run;

*--Turn on output formatter --;

%out2htm(capture=on);
options ls = 195;

*-- Macro to decide dataset --;
%MACRO DECIDE;

%macro eachdiv;
data work.collapse;
   merge &svplibs..collapse (in=m)
         &svplibs..control(keep=location
unitname);
   by location; if m;
   run;
%mend;

%macro company;
data work.collapse;
   set sasdl.collapse;
   run;
%mend;
   %if &svplibs = sasdl %then
     %do; %company; %end;
   %else %eachdiv;
%MEND;
%DECIDE;

*-- Determine Counts of various levels
of the data to use as output --;
proc sql noprint;

/* Total operational stores */
select count(*) into :totnum
   from work.collapse;

/* Total for rank of business */
select count(*) into :totrank
   from work.collapse
   where rank="&rank";

/* No. found based on user criteria */
select count(*) into :totcrit
   from work.collapse
   where rank="&rank" and sum
(t&cat&timevar,-(&level&cat&timevar ))
   &sign &amt;
   reset print;

/* Create table based on criteria */
create table stuff as
   select
      svp label="SVP",
      dvp label="DVP",
      rvp label="RVP",
      put(dm, _ndm13.) as dm,
      mainloc label="Mainloc", unitname,
      t&cat&timevar format=comma11.,
      &level&cat&timevar
format=comma11.,
      sum (t&cat&timevar , -
(&level&cat&timevar )) as var
label="Variance" format=comma11. ,
        dopen label="Opened", dclsd
label="Closed"
   from work.collapse
   where rank="&rank"
   having var &sign &amt
   order by var &ord;

*-- Title statements using counts
determined above and parms passed--;

title "<font color=black>&svp (&totnum)
Total Locations and (&totrank) &tvar1
Locations.<br></font>";

title2 "<font color=blue>Found &totcrit
&tvar1 Locations having &tvar2 &time
variance to &tvar3 &sign
$&amt..</font>";

    select * from stuff;
    quit;

*-- SQL output if nothing found --;

data _null_; file print;
   if &sqlobs = 0 then put "<H4><font
color=red>There were no records matching
your selections.</H2></font>";
   run;

*-- Default output formatter code--;
```

```
%out2htm(capture=off,
         htmlfref=_WEBOUT,
         runmode=s,
         openmode=replace,
         bgtype=color,
         bg=white,
         encode=n,
         tcolor=blue,
         ttag=header 4,
         septype=none,
         brtitle=);

*--- Reset macro vars set by program --;
%let timevar = ;
%let tvar1 = ;
%let tvar2 = ;
%let tvar3 = ;
%let ord = ;
```

Returning the Output

Good job! You've now got output. Moreover, better than that, it is output customized and requested on demand by the user.

Notice on the screen print, the numbers in brackets, which reflect the results of the global sql macros created in the program above.

What the User Sees in the Browser:

002=EDUCATION (1164) Total Locations and (743) A:Base Locations.
Found 88 A:Base Locations having Sales PD variance to Budget gt $25000.

SVP	DVP	RVP	DM	Mainloc	UNIT NAME
002	051	051	V JOHNSON	2202333001	GEORGIA TECH-ADMIN
002	056	070	B JONES	2221799001	EL PASO BASEBALL CLUB LLC
002	053	065	TIM SMITH	2294099001	WILLIAM PATERSON UNIVERSITY
002	052	060	H OCHOCKE	2257340001	OHIO STATE UNIV - CONCESSIONS
002	051	058	PAT TINCH	2201320001	ST MARY'S COLLEGE(CALIF)

CONCLUSION

We had to find creative ways in which to meet the needs of our users who connect over dial-up at very slow speeds. The solution was to allow users to customize the query up front and let the browser and SAS do most of the work in interpreting the criteria and returning a smaller amount of output. We can maintain and revise our own code easily, so we coupled existing SAS/IntrNet tools with JavaScript and SAS programs to create this application. The result is an efficient and effective experience for the end user and maintainable code.

Future Enhancements

The output as shown is pretty simply text from the output formatter. We've found that if we use the %ds2htm formatter instead, we can display the output in an HTML table. Any time there is a table displayed in the web browser, a user can highlight the output with their mouse and simple paste into MS Excel. If the output is in out2htm, then it is just preformatted text and this will paste as one column into Excel. In addition, by using tables, you can begin to apply style sheets to help with formatting of individual elements within the table, which gives you lots of control.

ACKNOWLEDGMENTS

The Technical Support staff at SAS Institute. Of particular mention is Bubba Tally who helped in our early days of htmSQL'ing.

Mr. Nick Thaler, my mentor, friend and boss for his day-to-day support and enthusiasm.

REFERENCES

SAS Institute's Web Site - Web Technology, Communities.

JavaScript: The Definitive Guide O'Reilly & Assoc.

Lots of trial and error. ☺

CONTACT INFORMATION

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

Karen L. Wyland
Sodexho
Decision Support Systems
883 Sneath Lane, Suite 213
San Bruno, CA 94066
Work Phone: 650-742-7630
E-mail Address: kwyland@pacbell.net