Experience Learned in Deploying htmSQL® Based Application

Hsiwei Yu (Michael), Digital Intelligence Systems, Centreville, VA
Chapman Gleason, U.S. Environmental Protection Agency, Washington, DC

ABSTRACT

This paper discusses techniques for dynamic drill-down of web pages using SAS htmSQL and JavaScript language elements. Possible future enhancements to htmSQL, encountered during our development, are also addressed.

INTRODUCTION

Problems: The user wants a web page showing all customers who made a purchase yesterday, and then click on a customer to see the detail invoice. Or, the user wants to select a product, and on the ***same*** page, seeing a list of stores having sold this item yesterday, then pick a store to see all activities for that store.

In the first scenario, we can use htmSQL, an integral part of SAS/IntrNet®, to query SAS or ***another DBMS*** and dynamically produce the list of customers. For the second one, since the list of stores is dependent upon the chosen product, we have to use additional JavaScript programming to produce the list of eligible stores. This one-level drill down can be expanded to two- or three-levels, though the JavaScript programming logic becomes increasingly complex. We have recognized patterns of the required JavaScript code, in the future we can automate by writing SAS Macro programs to generate the desired HTML file based on the application developer's input parameters.

SET UP SAS/SHARE SERVER

```
Libname libref1 'OS-file-name';
Libname libref2 'OS-directory';

/* Make formats known to this server */
Options fmtsearch= ( libref1.formats );

Proc server server=SAS_Share_server;
Run;
```

htmSQL requires a SAS/Share server. Modify the fmtsearch system option to allow your custom formats to be visible to htmSQL. Note only htmSQL can make use of a remote format, whereas local SAS session, using remote library services, cannot access format catalogs residing on a remote host. For example,

```
/* A client SAS session */
libname libref1 server= host.SAS_Share_server;
options fmtsearch= ( libref1.formats ) fmterr;
proc print data= ..;
formats var $a_format.;
run;
```

This client SAS would still fail because it can't make use of a format catalog residing on a remote host. This must be a future enhancement for the SAS System.

OUTLINES OF A HTMSQL INPUT FILE

A file used as input to htmSQL always has a file extension of hsql. Its content is a mixture of the regular HTML language elements and htmSQL directives.

```
<HTML><HEAD> .. <BODY> ..
{query datasrc="htmSQL-ds" .. }
  {sql}
    .. SQL query ..
  {/sql}
  {eachrow}
     .. html element to display a row ..
  {/eachrow}
  {norows}
     .. html element to show message ..
  {/norows}
{/query}
.. </BODY></HTML>
```

Since in our development we are only concerned with data retrieval, this paper will not discuss the {update} section available in htmSQL and its associated {success} and {error} directives.

IMPORTANT HTML AND JAVASCRIPT ELEMENTS

JavaScript can be embedded in an HTML file, and it is executed on the client's machine. Some frequently used features:

```
<FORM .. METHOD=POST
ACTION='htmSQL.exe/..hsql'>
<INPUT TYPE=HIDDEN NAME= VALUE= >
document.location.pathname
document.location.href
onchange( )
onclick( )
submit( )
```

To implement a dynamic drill down, we can concatenate the document.location.pathname property with the user's choice to construct the entire URL and then assign it to document.location.href property, effecting a new dynamic query result page. Alternatively, with post method and form's submit method, the newly constructed URL can be hidden from view.

Event handlers, such as onchange(), onclick(), are key functions for reacting to user actions.

HTMSQL VARIABLE REFERENCE AND CGI NAME AND VALUE PAIR

The name and value pairs passed by common gateway interface (CGI) is recognized by htmSQL and their values are resolved inside a hsql file, then the resulting file is sent to the web browser. For example, a URL like

```
../htmSQL.exe/aaa.hsql?NAME1=VALUE1& ..
```

And the corresponding hsql file has this content:

```
<SELECT NAME=any_name ONCHANGE=" .. ">
{sql}
{* This is the aaa.hsql file *}
select column1
  from a_SAS_dataset
 where {&NAME1} = "a_constant"
{/sql}
{eachrow}
<OPTION>{&column1}
{/eachrow}
</SELECT>
```

After resolution by htmSQL, the SQL becomes

```
..
{sql}
{* This is the aaa.hsql file *}
select column1
  from a_SAS_dataset
```

```
 where VALUE1 = “a_constant”
{/sql}
..
```

In the aforementioned SAS SQL context, VALUE1 must be a valid SAS column name in the *a_SAS_dataset*. If VALUE1 is not a valid column, say longer than eight characters, in the *a_SAS_dataset*, a SAS SQL syntax error occurs.

We can divide htmSQL variables into two groups based on their origins, those derived from CGI name and value pairs, and those purely from SAS SQL. In previous example, {&NAME1} is derived from CGI name and value pair, whereas {&COLUMN1} is purely from SAS SQL. It is best to give htmSQL variable distinctive name so as to avoid any confusion about a htmSQL variable's origin. For example, use more than 8 characters for htmSQL variable name derived from CGI input parameters. A practical use of htmSQL resolving CGI input parameters is you can set up hsql input file having htmSQL variable reference from CGI parameters only, without any query or update sections at all. A web page's content can be dynamically changed without writing a SAS/IntrNet program just to do the CGI parameter resolutions. It is a very handy feature.

APPLICATION 1: ONE-LEVEL DRILL DOWN

In this scenario, we use SASUSER.HOUSES data set for illustration. User is presented with a list of columns as radio boxes for sorting the list of houses. Clicking on a radio box, then the list of houses is dynamically sorted by that column, shown in Fig 1. The URL is something like:

..htmSQL.exe/one-level.hsql?order_column=STYLE

Outline for this main query section:

```
{sql}
    select style, bedrooms, sqfeet, street
      from sasuser.houses
  order by {&order_column} descending
{/sql}

<TABLE .. ><CAPTION>Sorted by ..
{&order_column}</CAPTION>
  <TR><TH .. >Style</TH> ..
{eachrow}
  <TR><TD .. >{&style}</TD>
{/eachrow}
```

Note the very first invocation of this page, the entire URL is plainly visible, a user can maliciously change its content, such as from 'order_column=STYLE' to 'order_column=ABCXYZ'. We added an extra query to check the existence of 'order_column' in SASUSER.HOUSES data set. If a non-existent column is requested, a norows directive can show message to the user. For a valid column, then eachrow directive can bring in the main query section to generate the desired house listing.

This trick is needed because

```
    select style, bedrooms, sqfeet, street
      from sasuser.houses
  order by non-existent-column descending
```

would produce syntax error for htmSQL. HTML programmer cannot trap these error messages from htmSQL. It would be nice if htmSQL provides a special directive for syntax error so that one can re-format these messages at will.

Finally additional JavaScript programming are needed, one when the user clicks on a different radio box, and one for making sure the radio box representing current selection is properly selected.

APPLICATION 2: TWO-LEVEL DRILL DOWN

In this scenario, user is first presented with a list of style of houses. Choose a style, then a list of number of bedrooms becomes available. Choose the desired bedrooms, all houses satisfying these criteria are shown (Fig 2).

THREE QUERY SECTIONS

```
   {sql}
       select distinct style
         from sasuser.houses
     order by style
   {/sql}
   {* .. .. }
   <!-- HTML elements not shown -->
   {sql}
   {* STYLE drives bedroom selection *}
      select distinct bedrooms
        from sasuser.houses
       where style =
   "%sysfunc(upcase({&style_x}))"
   {/sql}
   {* .. .. }
   <!-- HTML elements not shown -->
   {sql}
   {* STYLE and BEDROOMS drive eligible
   houses *}
   {* The eligible houses query *}
         select sqfeet, price, street
           from sasuser.houses
          where style =
   "%sysfunc(upcase({&style_x}))"
           and bedrooms = {&bedrooms_x}
   {/sql}
```

Note the selection criteria for eligible houses query is dependent on user input. Also note the use of %SYSFUNC in the where condition to translate user input into all capital-lettered constant. Since htmSQL is executed first on the web server machine and JavaScript later on client machine, we can't use JavaScript to do the capitalization because it would have been too late for htmSQL.

ANOTHER {NOROWS} TRICK

We decided to show eligible houses only after user has specified style and bedroom selections. In order to suppress the eligible houses query's effect when no bed room selection has been made, we use an auxiliary query section and embed it inside this auxiliary's eachrow directive.

```
{sql}
     select count( * ) as counter
       from sasuser.houses
      where style =
"%sysfunc(upcase({&style_x}))"
        and bedrooms = input(
"{&bedrooms_x}", 8. )
     having counter > 0
{/sql}
{eachrow}
{* Now include the eligible houses query *}
..
```

It's worth noting this extra query elongated the response time because the SAS/Share server must process four instead of three queries.

A REAL LIFE APPLICATION

Imagine a database storing yearly toxic chemical release information from each factory. A simplistic view of these records has the following dimensional columns:

```
State, County, Facility, Year, Chemical,
Media (through which a chemical is released
into environment, such as air, water, etc.)
```

The only numeric column is Amount (release weight in lbs.) Can we dynamically query this database from Internet?

The HTML interface, not using htmSQL, is in Fig 3. There can be some 600 chemicals required for report, but in a particular county and year combination, only a handful of chemicals were released. Similarly, not all counties in a state have reported toxic releases. Without htmSQL, we're obligated to show all 600 chemicals in the 'CHEMICAL' list and all counties in a state in the 'COUNTY' list. With htmSQL, we can show only those chemicals that were released, and only those counties that had reported releases, see Fig 4.

This application allows the user to select one of thirteen reports. Some reports, say a county-level report, the user must choose a county in order to see generated report; for a state-level report, the user must choose a state but not county. The content of the chemical list is dynamically determined by user's selection of an U.S. national-, state-, or county-level report. To control interactions between these selectable items, TYPE-OF-REPORT, STATE, COUNTY, YEAR, and CHEMICAL, JavaScript is required. Utilizing corroborative processing between htmSQL and JavaScript, we can produce dynamic content changes for not only the CHEMICAL, but also the COUNTY and STATE lists.

Here's a brief description of how to generate the list of chemicals for a given state and year combination, in response to a user clicking on the 'type of report' drop down list and choosing a state-level report. In order to optimize the web query's response time, we preprocessed the base SAS table to create a `chem_STATE_YEAR_dimension_dataset`, using

```
Proc sql;
/* Chemical found in particular state and year */
Create table chem_STATE_YEAR_dimension_dataset as
Select distinct state, year, chemical
From base_table
Group by state, year;
```

The input for htmSQL to create the chemical list has these elements in it,

```
<SELECT NAME=CHEMICAL>
<!--
http://../htmSQL.exe?Chemical_File=chemical-STATE-
YEAR&.. & ..
-->
{include file="..{&Chemical_File}.hsql"}
</SELECT>
```

Note the JavaScript variable 'Chemical_File' is set by the event handler associated with the 'type of report' menu option. In that event, noting a state-level report has been chosen by the user it does something like:

```
/* Snippet of JavaScript event handler */
Chemical_File="chemical_STATE_YEAR";

/* Also setting other selection values, such as
state and year values chosen by user */

/* Construct additional URL requirements */
/* Finally set to location.href */
document.location.href= ".. /htmSQL.exe" +
"?Chemical_File=chemical_STATE_YEAR" +
"&state_value=.." +
"&year=.." + ..;
```

Finally, the chemical_STATE_YEAR.hsql file has this content:

```
{sql}
{* This is chemical-STATE-YEAR.hsql file *}
select chemical
from chem_STATE_YEAR_dimension_dataset
where state = "{&state_value}"
  and year = {&year}
{/sql}
{eachrow}
<OPTION>{&chemical}
{/eachrow}
```

Note htmSQL variables, {&state_value}, {&year} are derived from CGI's name and value pairs.

We would repeat this basic process for county- and U.S. national-level chemical list, and so on. The final product is several files with mixed contents of HTML, htmSQL, and JavaScript language elements. As for the generated reports themselves, SAS/IntrNet programs are written for back-end processing.

CONCLUSION

We note that SAS/IntrNet application can have multiple servers for responding to a request, for example,
`http:// ..cgi-bin/broker?_SERVICE=default& ..`
This _SERVICE can have multiple SAS daemons ready to respond to an Internet client.

However, for htmSQL,
`{query DATASRC=".."}`
A DATASRC is essentially a ***single*** SAS/Share server, meaning a htmSQL-based HTML file can only be served by a single SAS/Share server. If the application has lots of users, a single Share server might produce unacceptable response time, despite the best optimization efforts. It is for this reason that we didn't actually deploy the htmSQL-based application, but chose instead a pure JavaScript-based version. We recommend SAS Institute enhance htmSQL to utilize several SAS/Share servers for responding to an Internet client

htmSQL is still very useful for generating HTML page with dynamic contents, such as the samples we have shown. From the samples in htmSQL installation package, we have developed an application capable of displaying the structures and contents of any Oracle database instance, including sample SAS code to extract Oracle data into SAS. Though the htmSQL-based TRI Explorer is not adopted for deployment at this time, we are looking into other ways for allowing users to dynamically query complex databases on the Internet.

ACKNOWLEDGMENTS

Authors like to thank Dr. William P. Smith of US Environmental Protection Agency who originally developed the TRI Explorer prototype. Jay Jacob Wind of American Environmental Institute provided SAS/IntrNet programming support. Also thanks to Steve Hufford, Nathan Wilkes, and Rashmi Lal of US EPA and Myles Powers of Logicon for providing requirement specifications and reviews for the TRI Explorer project. Special thanks to David Barron of SAS for helping to review this paper.

CONTACT INFORMATION

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

Hsiwei Yu (Michael)
Digital Intelligence Systems
Centreville, Virginia
http://www.disys.com
Work Phone: 202-260-5312
Email: vhyu@netkonnect.com

Chapman Gleason
U.S. Environmental Protection Agency
Washington, DC
Work Phone: 202-260-9006
Email: gleason.chapmane@epa.govl

Fig 1, One-level drill down, CGI input parameters visible

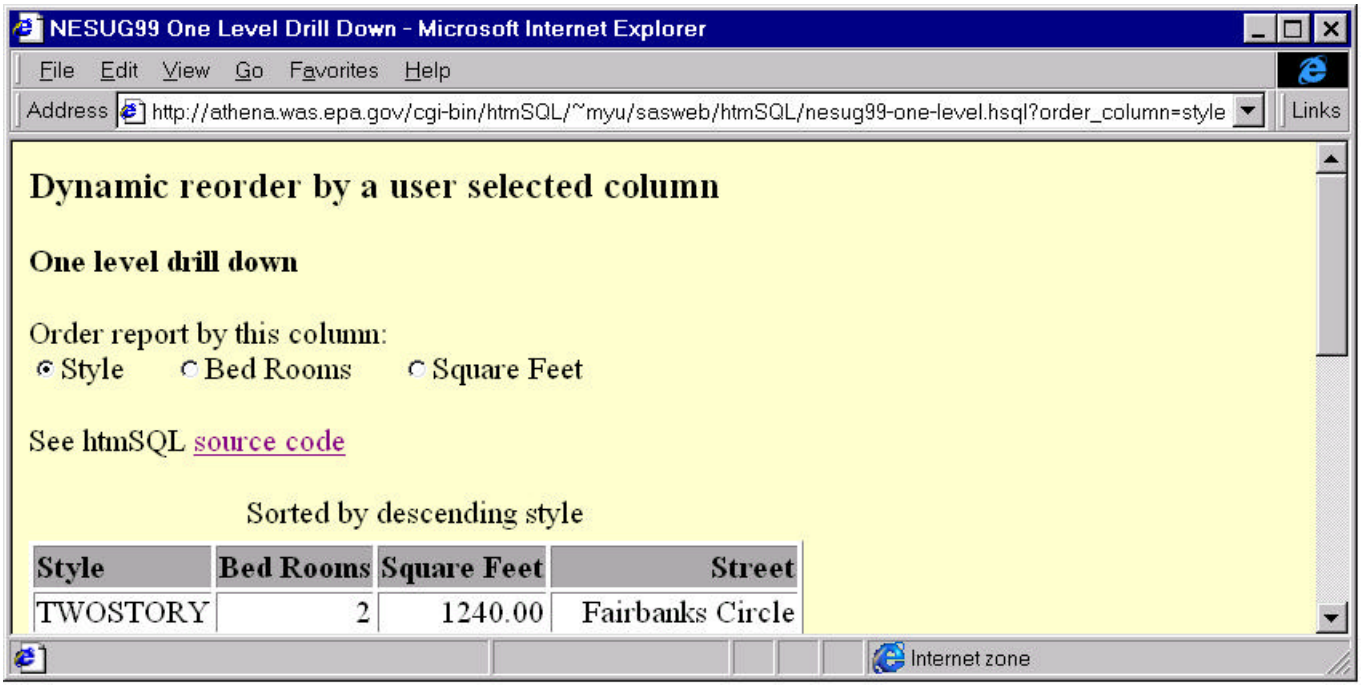

Fig 2, Two-level drill down, CGI input parameters hidden

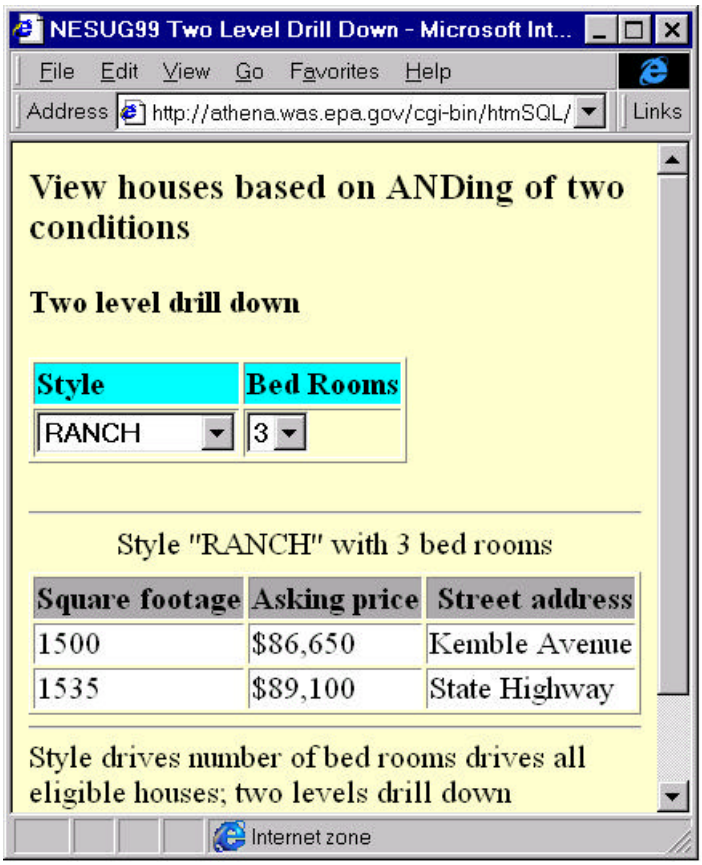

Fig 3, Without htmSQL, note many chemicals are not ever released in Wake county, North Carolina

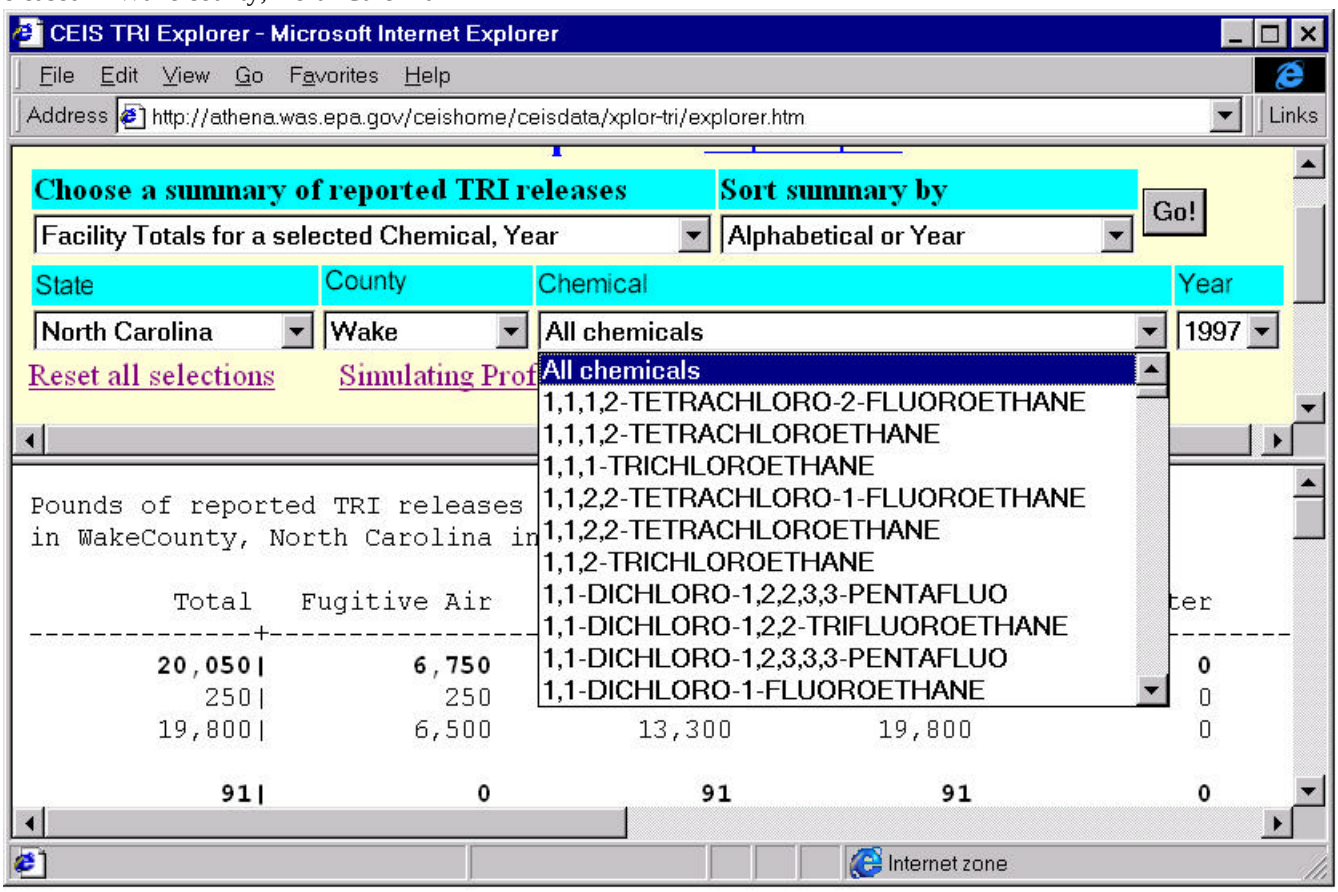

Fig 4, With htmSQL, chemicals can be dynamically changed