Paper 38-28

"The California Template" or "How to keep from reinventing the wheel using SAS/IntrNet, JavaScript and process reengineering"

Blake R. Sanders, U.S. Census Bureau, Washington, DC

ABSTRACT

Creating SAS/IntrNet applications is straightforward, but creating multiple programs with different datasets can be time consuming. In our experience, this is especially true when the application is extremely flexible and customized. This takes time, and that's not always available. By reengineering the process used in the applications, however, it is possible to have one set of flexible scripts to service multiple applications. This collection provides the basic tools for data analysis and output as well as an opportunity for customization. The result is simple, clean and easily maintained. It is a template on which applications can grow with virtually no start-up time and from which they can expand once the users' true needs come to light.

INTRODUCTION

Internet based applications are great tools to solve most of our data dissemination problems. Once you know the purpose of the application, they're fairly easy to organize and implement. The trick, however, is getting the process of the application organized correctly.

In order to get more Internet based application in the hands of our users in a timely fashion, we needed an option that did not require customizing every piece of the puzzle to meet that exact situation. Customization would be nice, but it would be best as an "add-on" -- something that could be plugged in after the application is up and running. This way, we could avoid having to reinvent the wheel each time someone wants a new program.

BACKGROUND

The Foreign Trade Division (FTD) of the U.S. Census Bureau has had several years of experience in creating Internet based applications – the majority of which have used SAS/IntrNet. Each application has been used to simply view. Each program follows the same basic process: 1.) Define the scope of the request, 2.) Define the output. Both steps could get more granular, but those details fall under the same general umbrella.

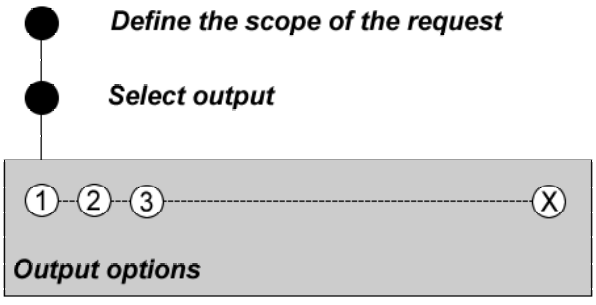

Figure 1 Basic structure of most applications

Initially, "define the scope of the request" meant identifying a specific dataset and then identifying the criteria on which to subset the data. Since then, users have needed to identify multiple datasets to cover a greater span of time. Additionally, they have needed to break their requests into stages that may change depending on the results of the previous stage. For example, users no longer need to view just data for February 2002. They'd rather access data for January through June 2002. Also, while, initially, they may have wanted to only look at data for Canada, once they'd seen what's traded with Canada, they'll want to focus on what's traded with Canada through specific ports.

When users would "define the output", they would select from multiple customized options. Canned reports, of course, needed to be created for the specific datasets. However, even our "dynamic" output options (where specific menus would appear based on previous selections) needed to be customized to the dataset.

Each approach, while suitable to their situation, required time to set up and modify when creating a new application.

What was needed was a solution that was easy to modify and easily implemented over a wide range of situations. If we were tasked with giving Internet access to three different sets of time series data, how could we give the users basic access as soon as possible while still having the option of customized solutions later?

SOLUTION

The solution was to modify our basic process to be flexible and adaptable up to a certain point and then provide a structure for the customized needs of the users.

In modifying "defining the scope of the request", we realized there were two things holding us back: 1.) Communicating early selections to the program, 2.) Defining the WHERE clause.

SESSION VARIABLES

In our initial applications, we embedded early selections in hidden HTML fields on subsequent pages. So, if a user needed to select a product and a country on separate pages, they would first select the product on the first page. When the second page, where the country was selected, was constructed, the selection from the first page would be put into a hidden field. That way, when the output was created (on the third page), the SAS program would receive both selections.

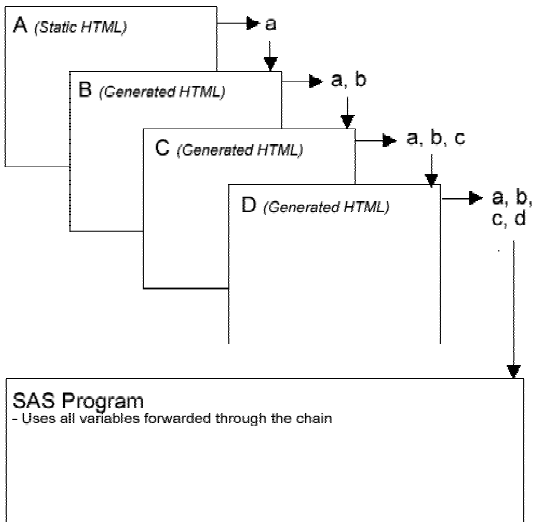

Figure 2 Forwarding macro variables over many generated pages

This has been solved by using SAS session variables. The selections are saved as temporary macro variables. There is still administration on our part in that the "session ID" needs to be forwarded from page to page. However, one it is only one item no matter how complex the system becomes.

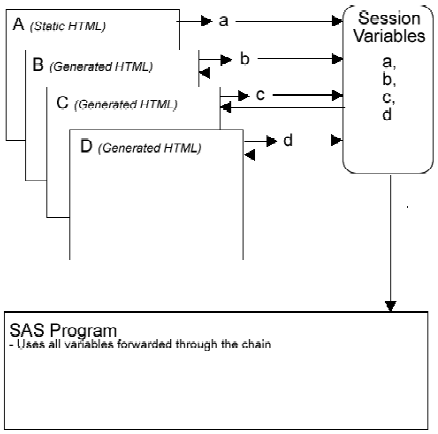

Figure 3 Forwarding the same variables using session variables

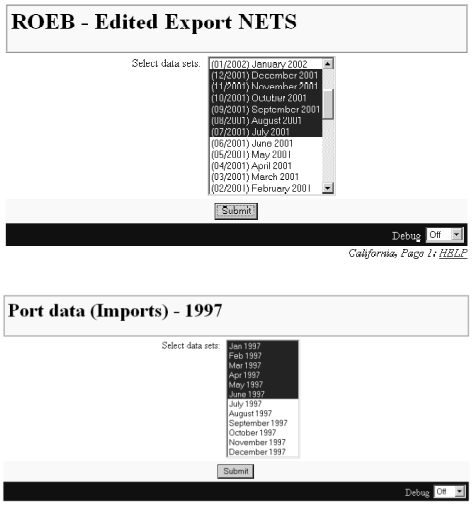

Figure 4 The front pages of two different applications that use the template

WHERE CLAUSE

To define WHERE clauses, we've assumed we knew the limits of a user's request. The assumption has always been about dimensions. "If they want data," we said, "odds are that the request will be on one or more of three dimensions: A x B x C." In our case, we assumed those dimensions would be COMMODITY (i.e. product), COUNTRY and DISTRICT. But, what if a user really needed to subset on a different variable?

The WHERE clause was a perfect tool to for this situation. It gives the most flexibility to sub setting based on the user's needs. The trick was making that easy.

So, we created a WHERE clause generator that prompted the user for the variables on which the clause was to be based from a menu generated from the structure of the selected datasets. It asked for the corresponding values of those variables and placed it in the clause. Additional pieces would be strung together with the usual Boolean operators: AND, OR and NOT. For example, if the user needed data for all imports from Canada that entered through the districts of New York City, Philadelphia or Baltimore, they would select "Country", "=" and type "1220" for Canada, click "ADD", click "AND", then select "DISTRICT", "IN" and type the codes for "New York City" (10), "Philadelphia" (11) and "Baltimore" (13), and click "ADD". The WHERE clause would then be "where (country='1220') and (district in (10','11','13'))".

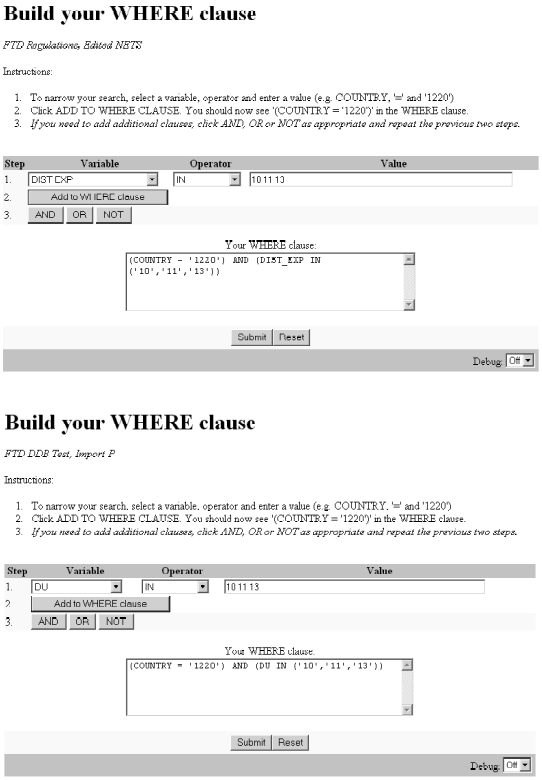

Figure 5 The WHERE pages of those two applications.

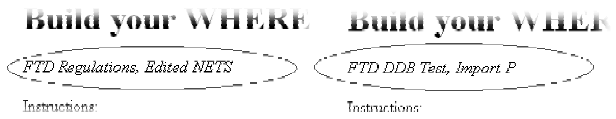

Figure 6 The difference between them is the label

The WHERE clause is also saved to a session variable.

Not only does this approach save us from having to forward multiple values from page to page, it spares us from having to create lists of possible values prior to using the programs (which is difficult when you don't know the dataset before it's used).

OUTPUT

When defining output, we realized that our users need at least one of three basic options: export to a file, browse or create a report. All options needed to be as flexible as possible.

EXPORT

To export a file, users have three options: Excel, CSV or SAS dataset.

BROWSE

"Browse" uses an option we call FlexBrowse where the user selects the variables they wish to see from the dataset (from a menu

generated based on the structure of the dataset), selects the format they wish to view (HTML, PDF or RTF) and selects the range of records they wish to view.

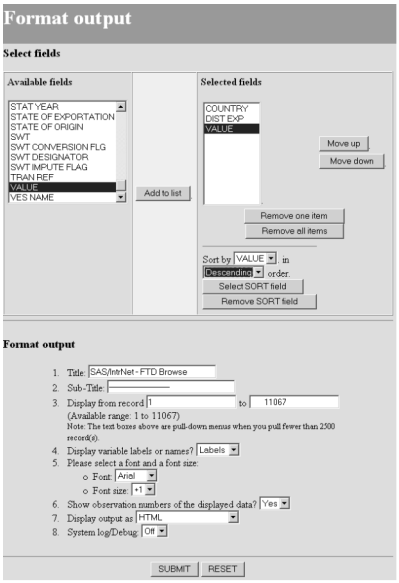

Figure 7 Browse

SAS/IntrNet - FTD Browse

Generated on 16DEC2002
SORTED on VALUE (descending)

Obs	COUNTRY	DIST EXP	VALUE
1	1220	10	9267097
2	1220	10	7578902
3	1220	10	6430386
4	1220	10	6384075
5	1220	10	5989496
6	1220	10	5369562
7	1220	10	3986379
8	1220	10	3983657
9	1220	13	3266542
10	1220	13	3250117
11	1220	10	3212231
12	1220	10	3109597
13	1220	13	3068199
14	1220	13	2999771
15	1220	10	2962768

Figure 8 Browse report

REPORT

To create a report, the users select several formatting options (e.g. font, font size, etc.) and fields they want to feed into a PROC TABULATE script. While we wanted to be as flexible as possible, we had to build in certain restrictions since giving users unrestricted access to PROC TABULATE in this environment is unrealistic. Do they NEED to be able to put as many variables DOWN or ACROSS as they desire? If they do, can they read the results? So, much like the TIMEOUT setting on Sas/IntrNet (where you select how long a request can run so it's not crowding out all of the other requests), we decided to limit users to three DOWN variables, two ACROSS variables, and two ANALYSIS variables.

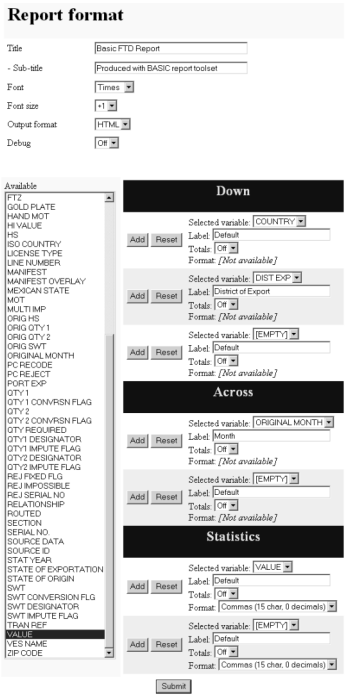

Figure 9 Report

Basic FTD Report
Produced with BASIC report toolset

		Month					
		07	08	09	10	11	12
		VALUE	VALUE	VALUE	VALUE	VALUE	VALUE
		Sum	Sum	Sum	Sum	Sum	Sum
COUNTRY	District of Export						
1220	10	44,206,263	40,474,809	32,124,211	56,539,094	40,028,222	26,476,871
	11	13,549,282	10,557,781	11,716,497	13,162,133	14,256,970	11,341,617
	13	2,532,030	4,570,035	3,929,496	4,507,287	87,713	5,364,438

Figure 10 Report after formatting

This report creator also gives users the ability to enter variable labels if they don't like the default labels and turn on and off totals. For ANALYSIS variables, FORMAT is also available. Again, considering we don't know all of the datasets which will be available, we can't provide FORMAT for the DOWN and ACROSS variables in this basic report generator.

CUSTOMZIED OUTPUT

Now that we've created our WHERE clause and given the users flexible output options, what about the customizable options that have been mentioned?

The first page of the chain, the static HTML where users select the datasets they want to query, has a hidden field that identifies the type of request originating from that page. If the page was for "import data via ports", the identifier might be "IMPORT-PORT". This identifier is registered in a dataset that lists all of the customized programs available on the server. When the output page is generated, if the original page was "IMPORT-PORT", the user will have access to the basic output options in addition to the customized "IMPORT-PORT" output options. Were the identifier "EXPORT-PORT", only the "EXPORT-PORT" options would be available under the Customized menu.

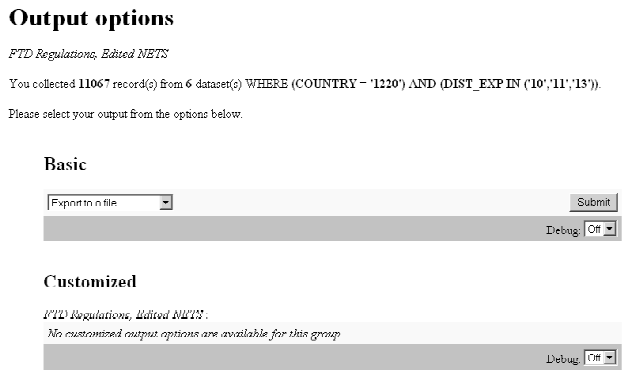

Figure 11 This application has no customized output options

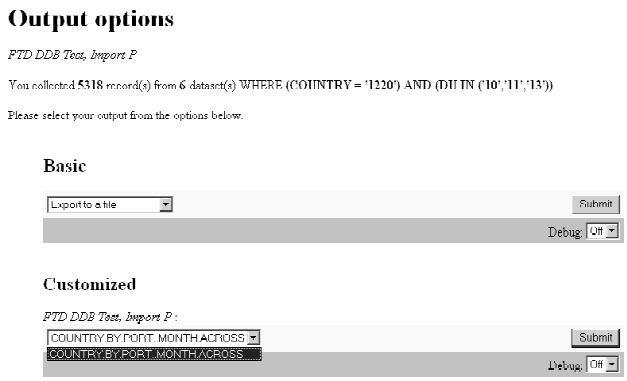

Figure 12 This one, however, does.

When a new customized output option is created, we need to add the location of that script to the registry under the first page of the application.

Since adding customized options is so simple, this gives us an opportunity to leverage one of the best assets we have in our data dissemination operation: users. Many enterprising people write SAS programs for their specific situation. If modified correctly, we can take those programs and add them to the list of customized output options.

In the end, we have a collection of documents and SAS programs that can serve a great many needs with a minimum amount of maintenance from us and a maximum amount of flexibility for the users.

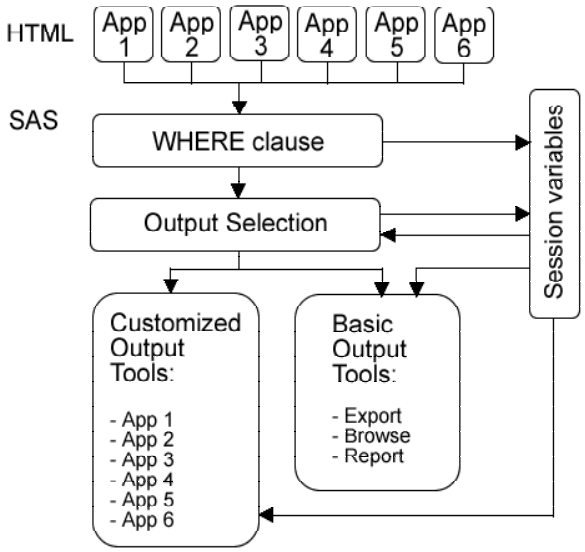

Figure 13 Multiple applications can use the same structure

DRAWBACKS

The FTD California template is great for quickly granting access to data over the web. However, depending on the situation, the completely customized approach may provide a better solution. Even so, those customized solutions can be based on this template. By copying the template and changing the beginning (because maybe the WHERE CLAUSE solution isn't for everyone), we can still save development time since the template is known and its reactions are predictable.

On a different note, since a common thread services all programs using the template, more attention will need to be paid to workload on the server and the service. If the workload gets too heavy for the service, define a different service on the front page. Session variables will propagate that through the rest of the application. If the workload on the server is too heavy, the template can be copied to an additional server. In this case, more administrative work will need to be done to make sure that all components of the template are up to date.

CONCLUSION

The California template gives us the ability to give quick access to any series of datasets. As long as the structure is consistent across the series (as is often the case in a time series), users will have access to the data as well as a basic set of tools for accessing it. The template provides a means by which users can help customize the experience for their specific datasets. This, in turn, will help promote interest in the maintenance of the program.

While completely customized programs contribute greatly to the productivity of an organization, it's good to have a useful, reusable tool available to quickly put something into production . That way, the development staff does not have to move Heaven and Earth all in the name of reinventing the wheel.

CODE

SAS Code to start a session, set a session variable and retrieve the value of a session variable:

```
1.) Create a session
        a.  %let
            rc=%sysfunc(appsrv_session(create
```

```
            ));
```

2.) Set a session variable

```
        a.  %global SAVE_ID;
            %let SAVE_ID = "&id";
```

i. Session variables always start with "SAVE_"

3.) Retrieve a session variable

```
        a.  put '<INPUT TYPE="HIDDEN"
            NAME="identification" VALUE="'
            &SAVE_ID '">';
```

i. Refer to it like a regular variable and not a macro variable.

JavaScript code to copy an item from one HTML menu to another HTML menu as is done in our FlexBrowse option:

```
// -----------------------------------------------

function menuProcess(aLeft,aRight) {

// This script is passed two items: two
// SELECT/OPTION menu objects.  It will scroll
// through the
// first menu looking for selected items.  If an
// item is selected, the script will copy it to
// the second menu.

// Scroll through all of the items in the first
// menu

for (var i = 0; i< aLeft.options.length; i++) {

// If the item is selected...

        if (aLeft.options[i].selected) {


                leftProcess(aLeft.options[i].text
                ,aLeft.options[i].value,aRight)


         }

}

// If the user is using Netscape Navigator,
// refresh the screen.
//
// Note: As this code was written, isNav is a
// boolean variable (true/false) set
// outside the function.  If 'navigator.appName
// == "Netscape"', it's set to TRUE.

     if (isNav) {

          history.go(0)

     }

}

// -----------------------------------------------

function
leftProcess(cLeftText,cLeftValue,rightMenu) {

// This function is passed three items: the text
// of the selected item in a SELECT/OPTION
// menu, the value of the same selected item,
// and the SELECT/OPTION menu object to which
// they are supposed to be added.  If there are
// no items in the destination menu, the text
// and value are added without question.  If //
// there are items in the destination menu, that
// menu is scanned to see if the value of the
// selected item already exists.  If so, the
// item is NOT added.

// Note: This edit is based on the VALUE and not
// the TEXT of the selected items.  So, in
// theory, once a list is processed, there could
// be two items on the destination menu with
// the same text label.  However, since they've
// gone through the edit, you know they have
// different values.

// Get the value and text of the selected item

   var leftText = cLeftText
   var leftValue = cLeftValue


// If the right hand menu is empty, add the item
// to the list.

   if (rightMenu.options[0].value == 0000) {

      rightMenu.options[0].value = leftValue
      rightMenu.options[0].text = leftText

// Commented out this line to experiment with
// adding a default option to the right menu.
// This line is appropriate if the right menu is
// completely EMPTY.


      addToList(leftText,leftValue,rightMenu)


   } else {

// Otherwise, find out if the item is already in
// the right hand menu.

      var found = false
      for (var j = 0; j <
              rightMenu.options.length; j++) {

         if (leftValue ==
           rightMenu.options[j].value) {

              found = true


         }

       }

// If not, add it to the right hand menu.

        if (!found) {

          addToList(leftText,leftValue,rightMenu)

        } else {

          alert("The selected item
          '"+leftValue+"' was already on the
          list.")

        }
   }
```

```
}

// ----------------------------------------

function addToList(bLeftText,bLeftValue,bRight)
{

// This function is passed three items: a text
// label for a SELECT/OPTION menu item, a value
// for a SELECT/OPTION menu item, and a
// SELECT/OPTION menu object to which the script
// should attach them.  By this point, it is
// assumed the text and value are correct and
// that the destination menu has been checked to
// make sure the text and value don't already
// exist.

     var newItem = new
        Option(bLeftText,bLeftValue)
     bRight.options[+bRight.options.length] =
        newItem

}
```

In an HTML page with two menus, “fields” and “selectedFields”, these scripts can be triggered by the click of a button whose code is…

```
<input type='button' name='Submit' value='Add to
list' onClick='menuProcess(this.form.fields,
this.form.selectedFields)'>
```

REFERENCES

SAS web site: “Using Sessions: A Sample Web Application”, http://www.sas.com/rnd/web/intrnet/dispatch/sesssamp.html

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact me at:

Blake Sanders
U.S. Census Bureau/Foreign Trade Division
FB-3, Rm-2158
Washington, DC 20233

Phone: 301-763-2234
Email: blake.r.sanders@census.gov
Web: http://www.census.gov/foreign-trade/www/