ABSTRACT
Creating SAS/IntrNet applications is straightforward, but custom coding programs to access different data sets can be time consuming. By re-engineering the process it is possible to have one set of scripts service multiple applications. This collection of programs provide for data analysis and output as well as a platform for customization. The result is simple, clean and easily maintained -- 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.

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 program follows the same basic process: 1.) Define the scope of the request and 2.) Select the output. See Figure 1.

SESSION VARIABLES
In our early applications, we preserved users' initial selections by embedding them in hidden HTML fields on subsequent web pages. For example, if we required a user to select a product and a country, they would select the product on the first page and the country on a second. When the second page was constructed, the selection from the first page would be placed in a hidden field. That way, when the output options were presented on a third page, the SAS program would have access to both 'forwarded' selections. See Figure 2.

More recently we’ve begun using SAS session variables to perform this function. Now the users’ selections are saved as temporary macro variables. While the “Session ID” must still be forwarded from page to page, it is only one item no matter how complex the system becomes. See Figure 3.
WHERE CLAUSE
To define WHERE clauses, we 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 different variables?

The WHERE clause was a perfect tool to for this situation, allowing us to flexibly subset based on the user's needs. We created a WHERE clause generator that prompted the user for the variables to include in the clause from a menu based on the variables of previously selected data sets. It asks for variables and for the corresponding values of those variables, then places both in the clause. Additional pieces are strung together with the usual Boolean operators: AND, OR and NOT, also obtained from the user.

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 resulting WHERE clause would then be "where (country='1220') and (district in (10,11,13))".

The WHERE clause is also saved to a session macro 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 – a difficult assignment when you don’t know the data set before it’s selected. See Figure 4.
We've created our WHERE clause and given the users flexible output options, but what about the additional customizations that have been mentioned?

On the first web page of an application — the page where users select the data set(s) they want to query, we have hidden a field that records the type of data requested. If the request was for “import data via ports”, the identifier would be “IMPORT-PORT”, for example. This ID is a program identifier, one of many to be found in a data set that lists all of the programs available on the server. In addition, this registry data set contains the output options specific to each listed program. When our user’s “IMPORT-PORT” output selection page is presented they will have access both to the application’s basic output options in addition to those specific to the “IMPORT-PORT” selection. Were the identifier “EXPORT-PORT”, only the basic and “EXPORT-PORT” options would be available under what we've labeled our Customized Menu. See Figures 9 and 10.

To create a report, the users select several formatting options (e.g. font, font size) 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, will they be able to read the results? We decided to limit users to three DOWN variables, two ACROSS variables, and two ANALYSIS variables.

This report generator also gives users the ability to enter variable labels if they don’t like the default labels. In addition, they can turn on and off the row and column totals. For ANALYSIS variables, FORMAT functionality is also available. Because we don’t know all of the data sets that will be available, we can’t provide FORMAT for the DOWN and ACROSS variables in this without writing custom code. See Figures 7 and 8.

CUSTOMIZED OUTPUT

We've created our WHERE clause and given the users flexible output options, but what about the additional customizations that have been mentioned?

On the first web page of an application — the page where users select the data set(s) they want to query, we have hidden a field that records the type of data requested. If the request was for “import data via ports”, the identifier would be “IMPORT-PORT”, for example. This ID is a program identifier, one of many to be found in a data set that lists all of the programs available on the server. In addition, this registry data set contains the output options specific to each listed program. When our user’s “IMPORT-PORT” output selection page is presented they will have access both to the application’s basic output options in addition to those specific to the “IMPORT-PORT” selection. Were the identifier “EXPORT-PORT”, only the basic and “EXPORT-PORT” options would be available under what we've labeled our Customized Menu. See Figures 9 and 10.
DRAWBACKS
The FTD’s California template is great for quickly granting access to data over the web. However, depending on the situation, a custom coded application may provide a better solution. Even so, such solutions can be based on the template. By copying the template and modifying its code -- perhaps because the WHERE clause isn’t adequate, for instance -- we can still save significant development time.

Since a common thread services all programs using the template, attention must be paid to workload on both the web service and the server. If the workload gets too heavy for the service, we can define a second service on the front pages of some applications. A session variable will propagate this value 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, we will need to ensure that all components of the copied template are revised appropriately.

CONCLUSION
The California template gives us the ability to quickly give our users access to a great deal of data. The template provides a means by which users can help customize the experience for their specific data sets. This, in turn, will help promote interest in the maintenance of the program.

While custom coded programs contribute greatly to the productivity of an organization, it’s good to have a reusable tool available to quickly put new reports into production. That way, the development staff does not have to move Heaven and Earth all in the name of reinventing the wheel.

REFERENCES

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/

ACKNOWLEDGEMENTS
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.
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 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
    }
    // 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
            break
        }
    }
    // 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.
    if (!found) {
        addToList(leftText,leftValue,rightMenu)
    }
}

// ---------------------------------------------
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) {
        if (isNav) {
            history.go(0)
        }
    }
}

// ---------------------------------------------
// 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)'>