

More Tricks and Tips With SAS/IntrNet®

Dan Hammarstrom

Keith Cranford

Attorney General of Texas, Child Support Division

Abstract

Have you wanted to include tabs in your SAS/IntrNet applications? Would you like an easier way to construct dynamic HTML forms? Have you wanted more control of your inline help? Taken from real world experiences, these techniques, as well as others, will be presented to help speed up your SAS/IntrNet application development. The paper includes SAS code, Javascript and Cascading Style Sheets.

Introduction

This paper is a continuation of a previous paper in which various techniques used in developing SAS/IntrNet applications were discussed. The goal of the papers remains unchanged, to share some of our experiences and how we learned to address different needs. The techniques shown here still only scratch the surface of the creative ways that you can approach application development within SAS/IntrNet. There are three essential tools used in the paper – SAS code, javascript, and cascading style sheets.

Each technique is presented with a general discussion of the need and how to address the need. SAS code, javascript and cascading style sheets are shown and discussed, as necessary. Complete program code for each technique is included in the Appendix.

Add Navigation HTML Output

There are times when you would like to include navigation and other browser functions to your reports and not rely on the browser. There are several ways to add navigation to SAS/IntrNet output. The method shown here uses buttons created using the SAS ODS headtext option, SAS macros, JavaScript, and Cascading Style Sheets (css). This section walks you through an example showing the steps needed to quickly add navigation to any html document.

Here is the program we will modify.

```
options missing = 0;

proc format;
  value $qtr 'qtr1' = "Qtr 1"
           'qtr2' = "Qtr 2"
           'qtr3' = "Qtr 3"
           'qtr4' = "Qtr 4";
run;

filename scsug "D:\data2\scsug\scsug1.htm";

ods listing close;
ods html
  body = scsug
  style = sasweb2;

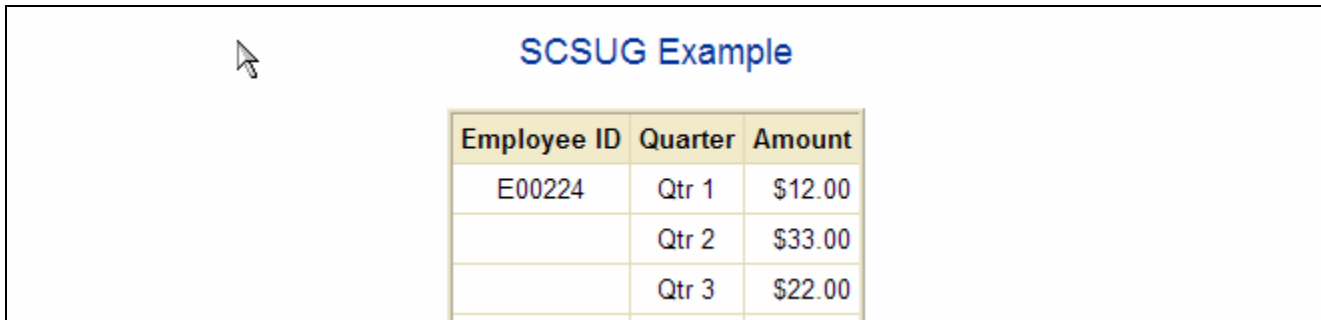
title1 "SCSUG Example";

proc report data = sasuser.contrib nowd split="~"
  style(header) = {vjust=bottom};
  column empid qtrnum amount;

  define empid / "Employee ID" center group;
  define qtrnum / "Quarter" center f=$qtr.;
  define amount / "Amount" f=dollar12.2 ;
run;

ods _all_ close;
```

ods listing;



Employee ID	Quarter	Amount
E00224	Qtr 1	\$12.00
	Qtr 2	\$33.00
	Qtr 3	\$22.00

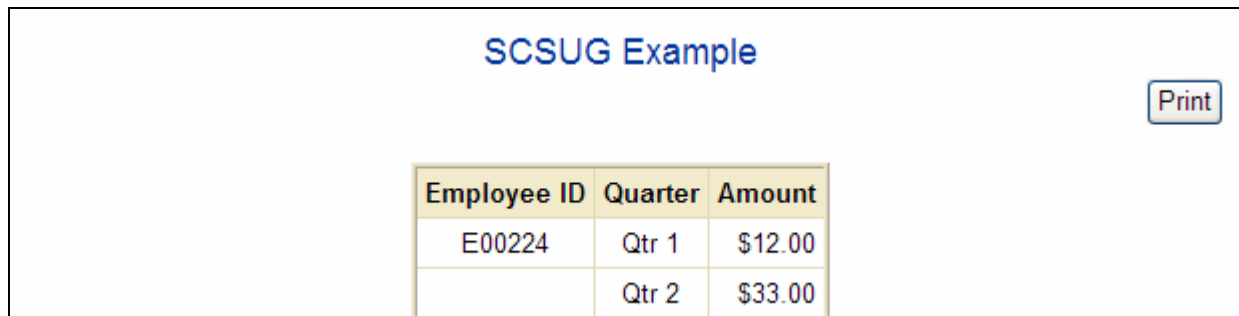
The first thing we want to add is a print button. To accomplish this add three lines of code to the program above.

%let printb = <script>PB()</script>; - This line creates a macro variable that will be called in a title and/or footnote statement. It can be located anywhere in your code but must be initiated before you use the macro. If you are going to use this macro and the ones to follow, it would be a good idea to create a macro library and access the macro variables from there. It would have been fine to just enter the html code right in the title statement. The only issue that you may run into is that you may run out of room if you want to add additional things to the same title statement. By using a JavaScript function to create the button for us, we get around that limitation. It's the PB() function that actually creates the button. And that is done at the time the html page is created.

headtext = '<script language="JavaScript1.2" src="javascript\scsug.js"></script>'; - Add this line to your ODS html statement in the headtext option. This line references a JavaScript file that contains the functions we will be calling to create appropriate buttons.

title2 j=r "&printb"; - This line references the printb macro.

Here is what you will see after adding these lines and submitting the program.



Employee ID	Quarter	Amount
E00224	Qtr 1	\$12.00
	Qtr 2	\$33.00

Print

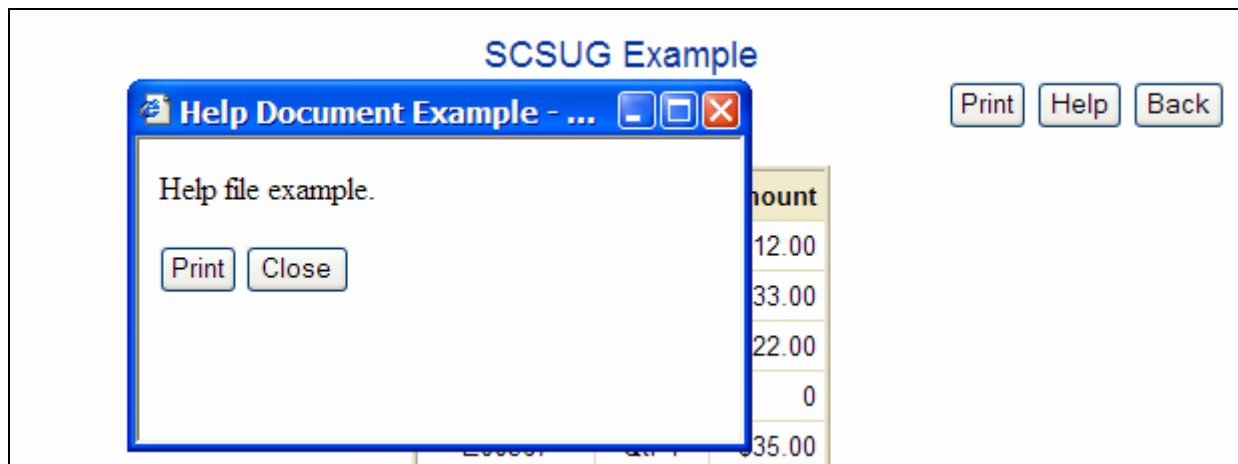
Now we need to add a back and help button. The back button works the same as the browser's back button. The help button opens a secondary window that contains information about report output.

%let backb = <script>BB()</script>; - Creates the backb macro variable.

%let helpb = %nrstr(<script>HB('&help_url.')</script>); - Creates the helpb macro variable.

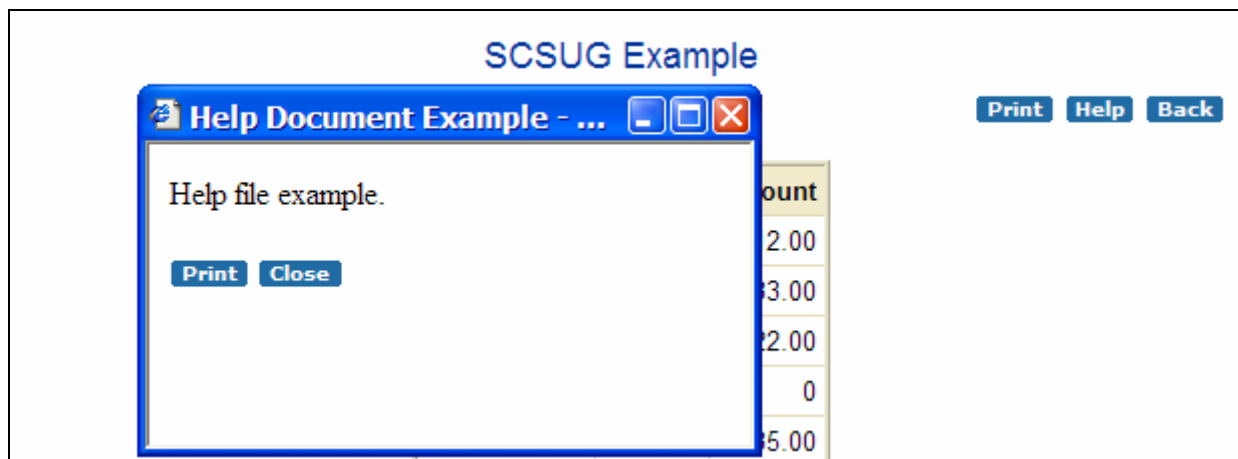
%let help_url = scsug_help1.htm ; - Creates the help_url macro variable. This is the reference to the help file that we want to open in the secondary browser window.

title2 j=r "&printb &helpb &backb"; - A help button and a back button will now be displayed.



These buttons work just fine but we need to change the appearance of the buttons so they fit into the color scheme we have selected and do not take up so much valuable real estate in the browser window. We will use a Cascading Style Sheet to change the look of the buttons.

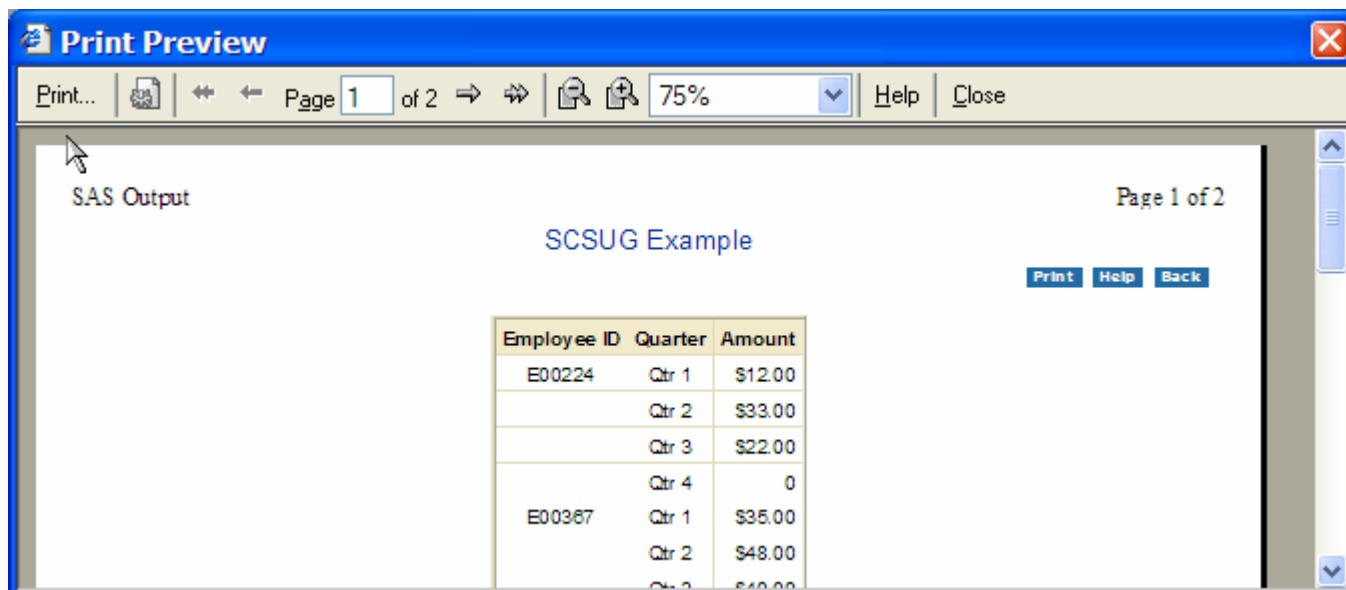
<link rel="stylesheet" type="text/css" href="css\scsug.css"> - Add this line to your ODS html statement. The headtext option adds the text you specified between the <HEAD> section of the html document. This line creates a reference to a Cascading Style Sheet that controls the appearance of the buttons.



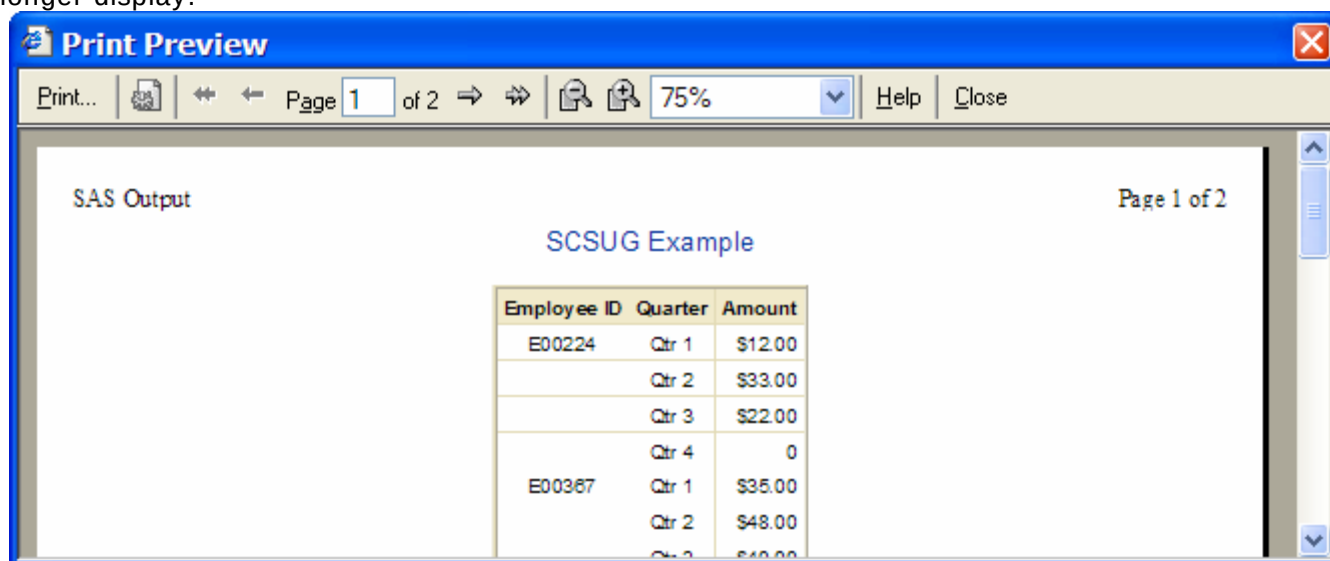
Now let's make sure that when I print the html document the new buttons don't print. There is a way to accomplish this by referencing a Cascading Style Sheet and telling the browser to use this style sheet only when a document is printed. This line is also added using the ODS headtext option.

<link rel="stylesheet" type="text/css" href="css\scsug_noprint.css" media="print">

Here is what the printed output will look like if you don't add the code.



And here is what the printed output will look like after you have added this code. Notice that the buttons no longer display.



Create Links to Other Reports

You may need to add links within a report to another report. SAS provides a way to create links to other document or reports. In this case we will create two temporary columns, populate them with graphics but only display those graphics and create the link to another document if certain criteria have been met. Here is the proc report where the new code has been bolded.

```
proc report data = sasuser.contrib nowd split="~"
  style(report) = {protectspecialchars=off}
  style(header) = {vjust=bottom};
  column empid qtrnum amount html excel

  define empid / "Employee ID" center group;
  define qtrnum / "Quarter" center f=$qtr.;
  define amount / "Amount" f=dollar12.2;
  define html / "HTML~Output" computed center ;
  define excel / "Excel~Output" computed center ;

  compute html / character length=195;
    if _c3_ > 20 then html = " ";
    else do;
```

```

html = '';
call define(_col_,"style","style={url = 'scsug_E00224_qtr1.htm' }");
end;
endcomp;

compute excel / character length=195;
excel = '';
call define(_col_,"style","style={url = 'E001224_quarter1.xls' }");
endcomp;

```

style(report) = {protectspecialchars=off} - This style option interprets how <, >, and & characters will be interpreted by the output destination. By setting this option to "off" we are ensuring that SAS will not protect these characters and the browser will interpret them as html tags.

Compute html / character length=195; - In this compute block we are creating a temporary column for this report called html. The variable type will be character with a length of 195. We are then checking the value of the third column which is the amount variable. If amount is greater than 20 then we are setting the value of html to missing. If amount is < than 20 we set the value of html to an html image tag, which will display a small Internet Explorer graphic. Using the call define statement and the style={url=} option we are telling SAS to wrap the graphic with hyperlink tag and set the destination of the link to another html page. In this case it's another html page.

Compute excel / character length=195; - In this compute block we are creating a temporary column for this report called excel. The variable type will be character with a length of 195. Regardless of the value of amount, we set the value of excel to an html image tag which will display a small Excel graphic. Using the call define statement and the style={url=} option we are telling SAS to wrap the graphic with a hyperlink tag and set the destination of the link to another html page. In this case it is an Excel Workbook.

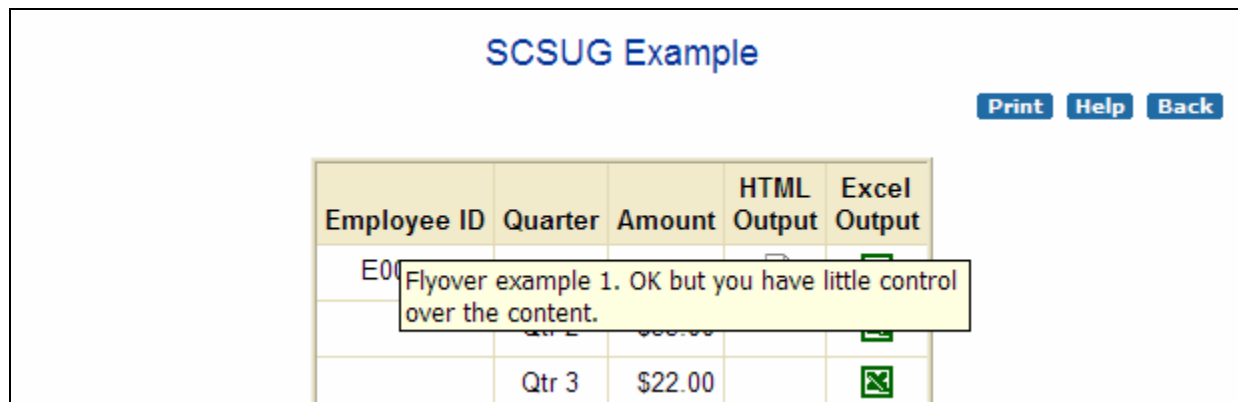
SCSUG Example				
			Print	Help Back
Employee ID	Quarter	Amount	HTML Output	Excel Output
E00224	Qtr 1	\$12.00		
	Qtr 2	\$33.00		
	Qtr 3	\$22.00		
	Qtr 4	0		

Tool Tips / Flyovers

SAS provides an easy way to add flyovers. However, you may need more capability within this flyover than the simple technique provides. We will consider both the simple method and a method that provides more flexibility.

Adding tool tips, popup, or flyovers when the user passes the mouse over the header of a column is easily accomplished using the flyover style option. Here is the code need to add a flyover to Employee ID.

style(header) = [flyover = "Flyover example 1. OK but you have little control over the content." htmlstyle = "cursor:help"]; - This code is added to the define statement for the Employee ID and will produce a tool tip when the user places their cursor anywhere in the header cell for this variable. The cursor will also change.



This will work well if you do not have a lot of information you want to relay to the users of this report. There is a limit to the amount of text you can use and you have no control over appearance. But there is an alternative that will give you almost unlimited space and control over the layout.

To accomplish this modify the program by adding the following:

```
define qtrnum / '<span onmouseover="TipOn(1);" onmouseout="TipOff();" class="tip">Quarter</span>'
               center f=$qtr.;
define amount / '<span onmouseover="TipOn(2);" onmouseout="TipOff();" class="tip">Amount</span>'
               f=dollar12.2;
define html / '<span onmouseover="TipOn(3);" onmouseout="TipOff();" class="tip">HTML<br>Output</span>'
              computed center;
define excel / '<span onmouseover="TipOn(4);" onmouseout="TipOff();" class="tip">Excel<br>Output</span>'
              computed center;
```

And the following at the bottom of the program.

```
data _null_;
  file scsug mod;
  put '<span id="tip_text" class="tool_Tip"></span>';
  put '<script>';
  put 'function TipOn(tip) {';
  put '  var tip_text = document.getElementById("tip_text");';
  put '  ';
  put '  if(tip == 1) {';
  put '    tip_text.innerHTML = "Flyover example 2.";';
  put '    tip_text.className = "tool_Tip";';
  put '  }';
  put '  ';
  put '  if(tip == 2) {';
  put '    tip_text.innerHTML = "Flyover example 3.<br><br>Make them<br>as big<br>as you want.";';
  put '    tip_text.className = "tool_Tip";';
  put '  }';
  put '  ';
  put '  if(tip == 3) {';
  put '    tip_text.innerHTML = "Flyover example 4.<br><br>Change the fonts.";';
  put '    tip_text.className = "tool_Tip2";';
  put '  }';
  put '  ';
  put '  if(tip == 4) {';
  put '    tip_text.innerHTML = "Flyover example 5.<br><br>Add a graphic.<br><br>" + ';
  put '      "<img src=' "'images/scsug.jpg" border='0'" ";';
  put '    tip_text.className = "tool_Tip";';
  put '  }';
  put '  ';
  put '  tip_text.style.display = "inline";';
  put '  tip_text.style.left = event.x ;';
  put '  tip_text.style.pixelTop = event.y + 17 ;';
  put '  ';
  put '  function TipOff() {';
  put '    var tip_text = document.getElementById("tip_text");';
  put '    tip_text.style.display = "none";';
  put '  }';
```

```
put '</script>';
run;
```

This code

1. Surrounds the labels of each of the four variables with html span tags, which have a class of "tip".
2. Writes the TipOn JavaScript function, which contains a single parameter for tool tip number which indicates the content of the tool tip.
3. Writes the TipOff JavaScript function, which turns off the tool tip.
4. Adds the JavaScript code to the bottom of the html output where the TipOn and TipOff functions are listed.

When the user places the cursor over the header label,

1. The TipOn function is run and a variable is passed to the function.
2. The TipOn function determines which tip needs to be displayed by evaluating the value variable passed when the function was accessed.
3. A span tag written to the html page in the data _null_ is populated with the appropriate text by using the innerHTML function.
4. The span tag display: style is then set to "inline" which will allow the tag to be viewed.
5. The location of the span tag is determined by the location of the mouse when it passes over the header label.
6. When the user removes the cursor the span tag display style is set to "none" and the tip is no longer visible.

Here are the tool tips this code produces.

SCSUG Example

Employee ID	Quarter	Amount	HTML Output	Excel Output
E00224	Qtr 1	\$12.00		
	Qtr 2	\$33.00		

Flyover example 2.

SCSUG Example

Employee ID	Quarter	Amount	HTML Output	Excel Output
E00224	Qtr 1	\$12.00		
	Qtr 2			
	Qtr 3			
	Qtr 4			
E00227	Qtr 1	\$35.00		

Flyover example 3.
Make them as big as you want.

SCSUG Example

Employee ID	Quarter	Amount	HTML Output	Excel Output
E00224	Qtr 1	\$12.00		
	Qtr 2	\$33.00		
	Qtr 3	\$22.00		
	Qtr 4	0		
E00367	Qtr 1	\$35.00		
	Qtr 2	\$48.00		
	Qtr 3	\$40.00		
	Qtr 4	\$30.00		

Flyover example 4.

Change the fonts.

SCSUG Example

Employee ID	Quarter	Amount	HTML Output	Excel Output
E00224	Qtr 1	\$12.00		
	Qtr 2	\$33.00		
	Qtr 3	\$22.00		
	Qtr 4	0		
E00367	Qtr 1	\$35.00		
	Qtr 2	\$48.00		
	Qtr 3	\$40.00		
	Qtr 4	\$30.00		
E00441	Qtr 1	0		
	Qtr 2	\$63.00		
	Qtr 3	\$89.00		
	Qtr 4	\$99.00		

Flyover example 5.

Add a graphic.



So as you can see with a little html, JavaScript, Cascading Style Sheet, and SAS ODS understanding, your html output can be more than just a simple proc print.

Displaying Multiple Output with Tabs

Many times when you display a graph a user would also like to see the data behind the graph. You could include a table on the same page with the graph, but this clutters the page and takes up valuable real estate. An alternative would be to use tabs where you can view the graph, but with a single click immediately see the data associated with the graph.

With a little javascript, cascading style sheets (CSS) and HTML code you can display multiple reports on the same HTML page and navigate to the reports by clicking on a tab. The general idea is to:

1. Display tabs at the top of the page using CSS classes and `<span>` tags
2. Divide your reports into sections using `<div>` tags
3. Navigate with the tabs calling a javascript function

First, the tabs are created using CSS classes and `<span>` tags. A `DATA _NULL_` data step is used to output this to the HTML file.

```
put '<span class="tabspace">&nbsp;&nbsp;&nbsp;'
' </span><span id="tab" class="tabon_output" '
    'onClick="tabs_output(this,' "'2'" ');">Graph&nbsp;&nbsp;&nbsp;'
' </span><span class="tabspace">'
' </span><span id="tab" class="taboff_output" '
'onClick="tabs_output(this,' "'2'" ');">&nbsp;&nbsp;&Data Table&nbsp;&nbsp;&nbsp;'
' </span><span class="tabspace">'
' </span><span class="tabline30"></span>'
' </span><span class="tabline30"></span>'
' </span><span class="tabline10"></span>'
'</span>';
```

This requires the following CSS classes, which are used in the `<span>` tags.

Class	Description
Tabon_output	What the tab will look like when it is the focus
Taboff_output	What the tab will look like when it is not the focus
Tabspace	Adds a little spacing between tabs
Tabline10/30	Adds a line 10% or 30% of the page width

In the example above there is a space, the “Graph” tab, another space, the “Data Table” tab, and finally a series of lines to stretch across the rest of the page. The result will look like this.

Graph **Data Table**

The “Graph” tab is the focus (using the `tabon_output` class), and the “Data Table” tab is in the background and not the focus (using the `taboff_output` class). Notice also the `onClick` option on the tab span tags. This indicates the javascript function that will be called to change the output, which will be discussed later.

<div> tags are used to layer the output. This is done by simply adding these tags with DATA _NULL_ steps before and after your SAS code that produces output.

```
put '<div id="data_tab" class="tabarea_output" style="display:inline">';

    <graph code>

put '    </div>';
put '<div id="data_tab" class="tabarea_output" style="display:none">';

    <data table report code>

put '    </div>';
```

The `tabarea_output` CSS class specifies what the report output area will look like, such as the background color. The `style` option indicates whether to display that layer – `display:inline` indicates to display, `display:none` indicates to hide the layer. Note the `id` option is the same for both layers. This will create a named array that will be referenced in the javascript.

Finally, the output is navigated by clicking on the tab you want to display. When this is done, the `onClick` event is triggered and the javascript function `tabs_output` is called.

```

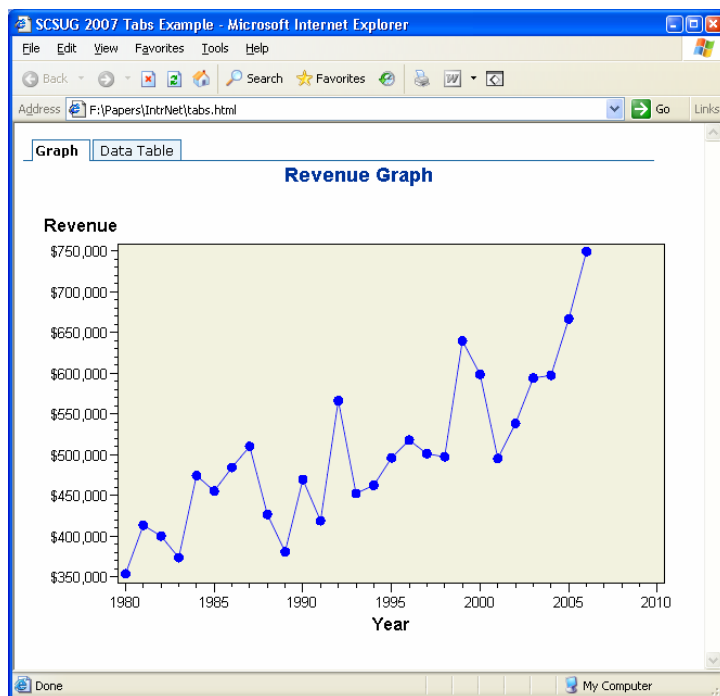
function tabs_output(tab_picked,cnt) {

    for (var i = 0; i < cnt; i++) {
        tab[i].className = "taboff_output";
        data_tab[i].style.display = "none";
    }
    for (var i = 0; i < cnt; i++) {
        if(tab_picked == tab[i]) {
            tab[i].className = 'tabon_output';
            data_tab[i].style.display = "inline";
        }
    }
}

```

The function call is `tabs_output(this, '2')`. The argument “this” indicates the particular tab that was clicked, and the argument “2” says that there are two tabs. The function first turns off all the tabs and layers. Then for the picked tab, it turns the tab on and the corresponding layer to inline display.

Initially, the final result looks like the first page, and when you click on the Data Table tab, you get the second.



The screenshot shows the same web browser window, but the "Data Table" tab is now selected. The main content area displays a table titled "Revenue Data". The table has 6 columns: "Year", "Revenue", "Year", "Revenue", "Year", "Revenue". It contains 17 rows of data, representing the years 1980 through 2006.

Year	Revenue	Year	Revenue	Year	Revenue
1980	\$354,046.29	1989	\$380,981.92	1998	\$497,373.74
1981	\$413,481.47	1990	\$469,602.34	1999	\$639,687.54
1982	\$400,313.17	1991	\$418,990.08	2000	\$598,321.22
1983	\$373,795.53	1992	\$566,233.04	2001	\$495,406.12
1984	\$474,410.25	1993	\$452,519.65	2002	\$538,348.82
1985	\$455,509.96	1994	\$462,452.86	2003	\$593,974.94
1986	\$484,300.18	1995	\$496,043.23	2004	\$597,187.64
1987	\$510,176.21	1996	\$517,847.46	2005	\$686,387.90
1988	\$426,720.05	1997	\$501,130.20	2006	\$749,036.40

Using PROC REPORT to Create Forms

SAS code can be used to create dynamic forms in SAS/IntrNet. The traditional approach has been to build these with `DATA _NULL_` steps using HTML tables to format the form. This results in you having to keep track of all the `<td>` and `<tr>` tags, which can be tricky if you have a complicated form. PROC REPORT provides an alternative to this approach. You will still have to include form item code, but you use PROC REPORT to format the table. This can save a lot of hassle and coding.

To illustrate this approach, consider an application where you want to view and update data from a user database. The form will contain the information for a particular user, and you want the ability to update that information. You need the following four steps.

1. HTML code to initiate the form
2. DATA step to reformat the data to input into PROC REPORT – most intensive step which requires formatting your data into columns

3. PROC REPORT step to display the form
4. HTML code to add submit button and close the form

The first and last steps could be done with `DATA_NULL_steps`, but a nifty way to get around this (since we are trying to eliminate those steps in the first place) is to use `TITLE` and `FOOTNOTE` statements. You can include the `<form>` tag HTML code in a `TITLE` statement.

```
title '<form name="Ideas" action="/cgi-bin/broker" method="pre">'
'<input type="hidden" name="_service" value="bkc"> '
'<input type="hidden" name="_program" value="oagprogd.scsug_update.sas">'
'<input type="hidden" name="_debug" value="0"> ';
```

SAS includes this when the PROC REPORT is executed, before the form items that are imbedded in the report output, which is just what you need. Similarly, a FOOTNOTE can be used to close off the form.

```
footnote j=left '<p></p>'
          '<input type="submit" name="Submit" value="Update">&nbsp;&nbsp; ';
          '<input name="Cancel" type="button" value="Cancel" onClick="history.back();">';
          '</form>';
```

This adds a Submit and Cancel buttons and closes out the form.

The real work is in the DATA step. This can be done by creating generic columns that contain either descriptive text or form items. By doing this, you line up the form elements nicely. The example attempts to show the different form elements, since each has to be handled slightly differently.

You should first set the length of the column variables. It is important that you set the length of the variable containing the form elements to be high, because some elements can be very long. In this example, COL1 contains the descriptive text, and COL2 contains the form elements.

```
data form ;
    set subset ;
    **-- Set lengths for columns --**;
    length col1 $20 col2 $ 2000 ;
```

First, COL1 is assigned the value 'Title,' which provides some descriptive text. The radio buttons are a bit tricky, since you need to define each one, plus set the "checked" option on the one that corresponds to the data value. In this example, there are four buttons for the possible values on title. To shorten the code, arrays can be combined with DO loops. Here the titles (what will actually be stored in the data set) and labels (what you want displayed on the form) are set into temporary arrays. A DO loop is executed for each button assigning the proper VALUE option and displaying the corresponding label. Included is a check for the current value of title, where if it matches the value of the button, the 'checked' option is added.

```

**-- Radio Button: Title --**;
col1 = 'Title';
array titles {4} $ 4 _temporary_ ('Mr.', 'Ms.', 'Dr.', '');
array labels {4} $ 4 _temporary_ ('Mr.', 'Ms.', 'Dr.', 'None');
col2 = '';
**-- Loop through radio buttons --**;
do i=1 to 4;
    col2 = strip(col2) || '<input type="radio" name="title" value="" || titles[i] || '""';
    if title=titles[i] then
        col2 = strip(col2) || ' checked';    **-- Check if title value same as radio button --**;
    col2=strip(col2) || '>' || labels[i] || '&nbsp;&nbsp;&nbsp;';
end;
output;

```

Text fields are perhaps the simplest form item to generate. You just need to assign the current value in the data set to the VALUE option. Then this value will be displayed. You can see this in the example for the NAME and CITY fields.

Using PROC REPORT to Create HTML Form
SCSUG SAS Educational Forum 2007

Title	☒ Mr. ☐ Ms. ☐ Dr. ☐ None
Name	Dan Hammarstrom
City	Tucson
State	AZ
☒ Add to e-mail list	

The form is interactive, so it can be changed. For example you can change the text or the drop-down selection value.

Using PROC REPORT to Create HTML Form
SCSUG SAS Educational Forum 2007

Title	☒ Mr. ☐ Ms. ☐ Dr. ☐ None
Name	Dan Hammarstrom
City	Austin
State	AZ AZ LA NM TX
☒ Add to e-mail list	

Once the changes have been made, clicking on the Update button calls the referenced program from the form and will update the original data set.

With just a little DATA step code, you can take full advantage of the formatting capabilities of PROC REPORT for laying out your data-driven forms. Although this is a fairly simple example, it provides the building blocks for more extensive forms that you could construct.

Automatic Creation of Javascript Code

Javascript is a handy tool in SAS/IntrNet. However, many times there are values within the functions that change over time, so that the code needs to be more dynamic. SAS can help with this issue. Since typically your data for SAS/IntrNet applications are already in SAS data sets, you can read these data sets to capture the current possible values and dynamically build the javascript functions. Usually this will be done at the time data is refreshed.

In the above example, the state drop-down selection list can be made dynamic. This is done by moving the selection lists options out of the program code and into a separate javascript program that can be automatically created by SAS. Two steps are needed.

1. Produce a list of possible values – PROC SQL or PROC FREQ work nicely for this
2. Use a DATA _NULL_ step to generate the javascript program, using the list

Either PROC SQL or PROC FREQ can be used to produce a data set of a list of possible values for a variable. In the example above, you are interested in the possible values of state.

```
proc sql ;
    create table state_list as
    select distinct state
    from info ;
quit ;
```

This PROC SQL creates a data set containing the unique values of state from the input data set. With this list, you can generate the javascript code fairly easily.

```
filename js "c:\temp\scsug07\states.js";
data _null_;
    set state_list nobs=last;
    file js ;

    if _n_ = 1 then do;
        put '// this file is generated by scsug_create_js.sas';
        put 'function loadstate(nameobj) {';
        put '    with (nameobj) {';
        put '        options.length=" last +(-1)";';
        end;

    n = _n_-1;

    put '    options[' n +(-1) '].text="" state +(-1)";';
    put '    options[' n +(-1) '].value="" state +(-1)";';

    if last = _n_ then do;
        put '    }';
        put '};';

        put 'function setstate(currval) {';
        put '    for (var i=0; i < document.ideas.state.length; i++) {';
        put '        if (document.ideas.state.options[i].value == currval) {';
        put '            var sel=i ;';
        put '        }';
        put '    }';
        put '    if (sel >= 0 ) {';
        put '        document.ideas.state.options[sel].selected = true;';
        put '    }';
        put '    else {';
        put '        document.ideas.state.options[0].selected = true;';
        put '    }';
        put '};';
    end;
run;
```

First, the initial part of the function is written. The function statement with the name and parameters are written, as well as the statements for identifying the selection list to be used and setting the options

length. Then the values of state are subsequently used for setting the VALUE and TEXT for the options. Finally, at the end the function is closed. In this example, a second function is also written to select the current value of state in the selection list when the form is loaded.

This program can then be run after the input data is updated, either on a regular schedule or as a part of a dynamic update. Unless the input data set is very large, this program should run very quickly.

With a few changes to the report form program discussed earlier, the state selection list becomes dynamic. Since the selection list is loaded by the javascript code, you do not include the <option> tags directly in the HTML form selection element. This simplifies the DATA step.

```
***-- Selection list (drop-down): State --**;  
col1 = 'State' ;  
col2 = '<select name="state" id="state" size="1"></select>' ;  
output ;
```

However, you still have to load the selection list and set the current value appropriately. This is accomplished with a few javascript function calls, which are made after the initial form is loaded. You may recall a FOOTNOTE was used to display the submit button and close the form. Subsequent FOOTNOTE statements can be used to load the selection list (FOOTNOTE2 below), set the current value (FOOTNOTE3), and place the cursor where you want it initially (FOOTNOTE4).

```
footnote2 '<script>loadstate(document.ideas.state);</script>' ;  
footnote3 '<script>setstate("'"&currval" '"');</script>' ;  
footnote4 '<script>document.ideas.name.focus();'  
          'document.ideas.name.select();</script>' ;
```

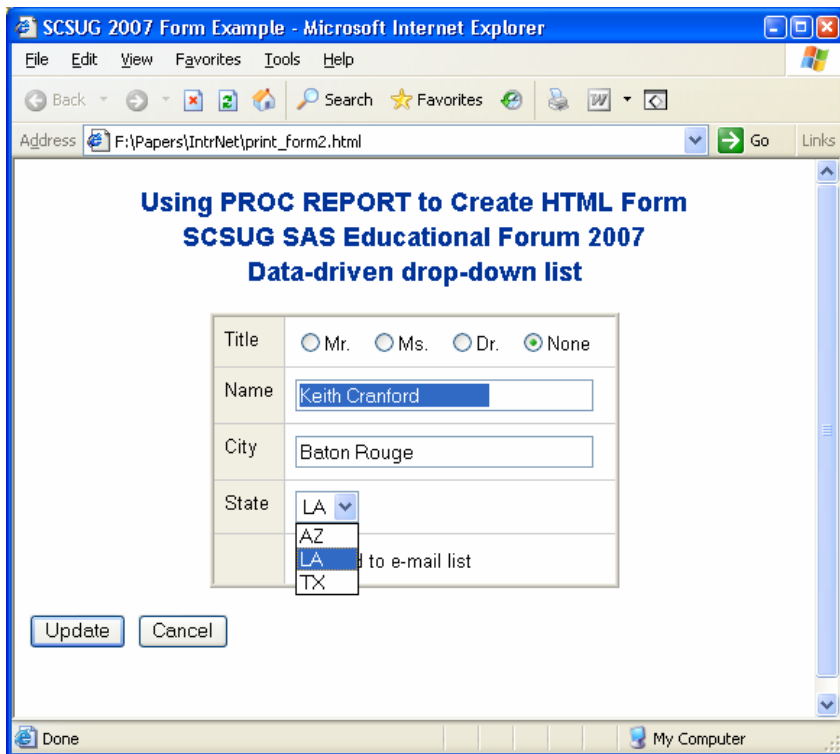
The argument of the loadstate function is the form item of the selection list. In this example this references the current document, the 'ideas' form, and the 'state' form item. Setting the current value of the selection list requires you specify the current value somehow. This can be done using a macro variable, which can be set when the data is initially read.

```
call symputx('currval',state) ;
```

Finally, since this requires the javascript functions created above, you need to reference this javascript program. This is done in the HEADTEXT option of ODS HTML statement.

```
ods html  
  body = outhtml (title='SCSUG 2007 Form Example')  
  headtext='<script language="JavaScript1.2" src="c:\temp\scsug07\states.js"></script>'<br>  
  style = sasweb;
```

The resulting form looks like this.



This looks similar to the previous example, except that now the selection list is being created by the javascript that was generated dynamically from the actual values in the data set.

Conclusion

Developing SAS/IntrNet application development always presents new challenges. With handy tools such as javascript, CSS, and, of course, SAS (especially ODS) you can meet these challenges and have a chance to use your creativity. Feel free to use the code provided here to meet some of your needs and enhance it to make it even better!

If you have questions, please feel free to contact us at:

Dan Hammarstrom
Systems Analyst
Attorney General of Texas, Child Support Division
dan.hammarstrom@cs.oag.state.tx.us

Keith Cranford
Systems Analyst
Attorney General of Texas, Child Support Division
keith.cranford@cs.oag.state.tx.us

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.

Appendix

scsug_final.sas

```
%let printb = <script>PB()</script>;
%let backb = <script>BB()</script>;
%let helpb = %nrstr(<script>HB('&help_url.')</script>);

%let help_url = scsug_help2.htm ;

proc format;
  value $qtr 'qtr1' = "Qtr 1"
    'qtr2' = "Qtr 2"
    'qtr3' = "Qtr 3"
    'qtr4' = "Qtr 4";
run;

filename scsug "D:\data2\scsug\scsug_final.htm";

ods listing close;
ods html
  body = scsug
  style = sasweb2
  headtext =
'<script language="JavaScript1.2" src="javascript\scsug.js"></script>
<link rel="stylesheet" type="text/css" href="css\scsug.css">
<link rel="stylesheet" type="text/css" href="css\scsug_noprint.css" media="print">';

  title1 "SCSUG Example";
  title2 j=r "&printb &helpb &backb";

proc report data = sasuser.contrib nowd split="~"
  style(report) = {protectspecialchars=off}
  style(header) = {vjust=bottom};
  column empid qtrnum amount html excel;

  define empid / "Employee ID" center group style(header) =
    [flyover = "Flyover example 1. OK but you have little control over the content."
    htmlstyle = "cursor:help"];
  define qtrnum / '<span onMouseOver="TipOn(1);" onMouseOut="TipOff();" class="tip">Quarter</span>'
    center f=$qtr.;
  define amount / '<span onMouseOver="TipOn(2);" onMouseOut="TipOff();" class="tip">Amount</span>'
    f=dollar12.2;
  define html / '<span onMouseOver="TipOn(3);" onMouseOut="TipOff();" class="tip">HTML<br>Output</span>'
    computed center;
  define excel / '<span onMouseOver="TipOn(4);" onMouseOut="TipOff();" class="tip">Excel<br>Output</span>'
    computed center;

  compute html / character length=195;
    if _c3_ > 20 then html = " ";
    else do;
      html = '';
      call define(_col_,"style","style={url = 'scsug_E00224_qtr1.htm' }");
    end;
  endcomp;

  compute excel / character length=195;
    excel = '';
    call define(_col_,"style","style={url = 'E001224_quarter1.xls' }");
  endcomp;

run;

ods _all_ close;
ods listing;

data _null_;
  file scsug mod;
```

```

put '<span id="tip_text" class="tool_Tip"></span>';
put '<script>';
put 'function TipOn(tip) {';
put '  var tip_text = document.getElementById("tip_text");';

put '  if(tip == 1) {';
put '    tip_text.innerHTML = "Flyover example 2.";';
put '    tip_text.className = "tool_Tip";';
put '  }';';
put '  ';
put '  if(tip == 2) {';
put '    tip_text.innerHTML = "Flyover example 3.<br><br>Make them<br>as big<br>as you want.";';
put '    tip_text.className = "tool_Tip";';
put '  }';';
put '  ';
put '  if(tip == 3) {';
put '    tip_text.innerHTML = "Flyover example 4.<br><br>Change the fonts.";';
put '    tip_text.className = "tool_Tip2";';
put '  }';';
put '  ';
put '  if(tip == 4) {';
put '    tip_text.innerHTML = "Flyover example 5.<br><br>Add a graphic.<br><br>" + '
put '      "<img src=' "'images/scsug.jpg' border='0'>" "'";';
put '    tip_text.className = "tool_Tip";';
put '  }';';
put '  ';
put '  tip_text.style.display = "inline";';
put '  tip_text.style.left = event.x ;';
put '  tip_text.style.pixelTop = event.y + 17 ;';
put '  }';';

put 'function TipOff() {';
put '  var tip_text = document.getElementById("tip_text");';
put '  tip_text.style.display = "none";';
put '  }';';
put '</script>';
run;

```

scsug.css

```

/* Allows the header to print at the top of each page */
thead {
  display:table-header-group
}

.buttons {
  border-right:#ffffff 1px solid;
  border-left:#ffffff 1px solid;
  border-top:#ffffff 1px none;
  border-bottom:#ffffff 1px none;
  font-family:verdana;
  font-weight:bold;
  font-size:10px;
  cursor:hand;
  background-color:#206BA4;
  color:#ffffff;
}

.buttonshelp {
  border-right:#ffffff 1px solid;
  border-left:#ffffff 1px solid;
  border-top:#ffffff 1px none;
  border-bottom:#ffffff 1px none;
  font-family:verdana;
  font-weight:bold;
  font-size:10px;
  cursor:help;
  background-color:#206BA4;
  color:#ffffff;
}

```

```

.tip {
    cursor:help;
}

.tool_tip {
    font-family : Arial;
    font-size:10pt;
    color:#000000;
    font-weight:bold;
    background-color:#66FFFF;
    padding:2px;
    z-index:99;
    border: outset;
    border-width: 2px;
    display:none;
    position:absolute;
}

.tool_tip2 {
    font-family : Times New Roman;
    font-size:3em;
    color:#000000;
    font-weight:bold;
    background-color:#66FFFF;
    padding:2px;
    z-index:99;
    border: outset;
    border-width: 2px;
    display:none;
    position:absolute;
}

```

scsug_noprint.css

```

.buttons, .buttonshelp {
    display:none;
}

a:active, a:link, a:visited, a:hover {
    color:#000000;
    text-decoration:none;
}

```

scsug.js

```

// JavaScript Document

function printPage(){
    window.print()
}

function CloseWindow(){
    window.close()
}

function PB() {
    document.writeln("<input type='button' class='buttons' value='Print' onClick='javascript:window.print()'"
    title='Click here to print this page.'>");
}

function BB() {
    document.writeln("<input type='button' class='buttons' value='Back' onClick='history.go(-1)'"
    title='Click here to return to the previous page.'>");
}

function TB() {
    document.writeln("<input type='button' class='buttons' value='Top' onClick='javascript:document.all.top.scrollToView()'"
    title='Click here to return to the top of this page.'>");
}

```



```

data _null_ ;
    file outhtml ;
    put '    </div>' ;
    put '<div id="data_tab" class="tabarea_output" style="display:none">' ;
run ;

ods html3 path='c:\temp\scsug07'
    body = outhtml (title='SCSUG 2007 Tabs Example' no_top_matter no_bottom_matter)
    style = sasweb;

options missing=' ' ;
    title 'Revenue Data' ;
    proc report data=tabs_data nowd ;
        column year revenue year2 revenue2 year3 revenue3 ;
        define year / 'Year' display style(column)=[background=#F1EFE2 font_weight=bold] ;
        define revenue / 'Revenue' format=dollar12.2 ;
        define year2 / 'Year' display style(column)=[background=#F1EFE2 font_weight=bold];
        define revenue2 / 'Revenue' format=dollar12.2 ;
        define year3 / 'Year' display style(column)=[background=#F1EFE2 font_weight=bold] ;
        define revenue3 / 'Revenue' format=dollar12.2 ;
    run ;

ods html3 close ;

data _null_ ;
    file outhtml ;
    put '    </div>' ;
run ;

ods html3
    body = outhtml (title='SCSUG 2007 Tabs Example' no_top_matter) ;
ods html3 close ;

```

tabs.css

```

.tabon_output {
    font-family: Verdana;
    font-size: 12px;
    font-weight: bold;
    background-color: #ffffff;
    padding: 2;
    height: 20px;
    border-right: #206ba4 1px solid;
    border-left: #206ba4 1px solid;
    border-top: #206ba4 1px solid;
    cursor:hand;
}

.taboff_output {
    font-family: Verdana;
    font-size : 12px;
    background-color: #ebf4fa;
    padding: 2;
    height: 20px;
    border: #206ba4 1px solid;
    cursor:hand;
}

.tabline10 {
    font-family: Verdana;
    font-size : 12px;
    background-color: #ffffff;
    padding: 2;
    border-bottom: #206ba4 1px solid;
    width:10%;
}

.tabline30 {
    font-family: Verdana;

```

```

        font-size : 12px;
        background-color: #ffffff;
        padding: 2;
        border-bottom: #206ba4 1px solid;
        width:30%;
    }

    .tabspace {
        font-family: Verdana;
        font-size : 12px;
        background-color: #ffffff;
        border-bottom: #206ba4 1px solid;
        width:2px;
    }

    .tabarea_output {
        text-align: left;
        font-family: Verdana;
        font-size : 12px;
        background-color: #ffffff;
        padding: 2;
    }
}

```

tabs.js

```

// used with the html output from IDEAS
function tabs_output(tab_picked,cnt) {

    for (var i = 0; i < cnt; i++) {
        tab[i].className = "taboff_output";
        data_tab[i].style.display = "none";
    }
    for (var i = 0; i < cnt; i++) {
        if(tab_picked == tab[i]) {
            tab[i].className = 'tabon_output';
            data_tab[i].style.display = "inline";
        }
    }
}
}

```

scsug_report_form.sas

```

*-- Change to see other records --**;
%let partname=Hamm ;

data info ;
    length title $ 3 name city $ 30 state $ 2 email $ 1 ;
    title= 'Mr.' ;
    name= 'Dan Hammarstrom' ;
    City= 'Tucson' ;
    State='AZ' ;
    email='Y' ;
    output ;
    title= ' ' ;
    name= 'Keith Cranford' ;
    City= 'Baton Rouge' ;
    State='LA' ;
    email='N' ;
    output ;
    title= 'Ms.' ;
    name= 'Paula Linderman' ;
    City= 'Victoria' ;
    State='TX' ;
    email='N' ;
    output ;
run ;

```

```

data subset ;
    set info ;
    where name contains "&partname" ;
run ;

filename outhtml 'c:\temp\scsug07\print_form.html' ;
ods listing close;
ods html
    body = outhtml (title='SCSUG 2007 Form Example')
    style = sasweb;

    **-- Reformat data set with one column of descriptive text and  --**;
    **-- one column with HTML INPUT fields  --**;
data form ;
    set subset ;
    ***-- Set lengths for columns --**;
    length col1 $20 col2 $ 2000 ;

    ***-- Radio Button: Title --**;
    col1 = 'Title' ;
    array titles {4} $ 4 _temporary_ ('Mr.', 'Ms.', 'Dr.', ' ');
    array labels {4} $ 4 _temporary_ ('Mr.', 'Ms.', 'Dr.', 'None') ;
    col2 = ' ' ;
    ***-- Loop through radio buttons --**;
    do i=1 to 4 ;
        col2 = strip(col2) || '<input type="radio" name="title" value="" ||
            titles[i] || '""';
        if title=titles[i] then
            col2 = strip(col2) || ' checked' ;
            ***-- Check if title value same as radio button --**;
            col2=strip(col2) || '>' || labels[i] || '&nbsp;&nbsp;&nbsp;';
    end ;
    output ;

    ***-- Text Field: Name --**;
    col1= 'Name' ;
    col2= '<input name="name" type="text" id="name" size="30" ' ||
        'maxlength="30" value="" || name || "">&nbsp;&nbsp;&nbsp;';
    output ;

    ***-- Text Field: City --**;
    col1= 'City' ;
    col2= '<input name="city" type="text" id="city" size="30" ' ||
        'maxlength="30" value="" || city || "">&nbsp;&nbsp;&nbsp;';
    output ;

    ***-- Selection list (drop-down): State --**;
    col1 = 'State' ;
    col2 = '<select name="state" id="state" size="1">' ;
    ***-- Add options for each state --**;
    do st='AZ', 'LA', 'NM', 'TX' ;
        col2 = strip(col2) || '<option value="" || st || '""';
        if state=st then
            col2 = strip(col2) || ' selected' ;
            ***-- Select if state same as option --**;
            col2 = strip(col2) || '>' || st || '</option>';
    end ;
    col2 = strip(col2) || '</select>';
    output ;

    ***-- Checkbox: e-mail --**;
    col1 = ' ' ;
    col2 = '<input name="email" type="checkbox" id="email"' ;
    if email='Y' then
        col2= strip(col2) || ' checked' ; ***-- Check if email=Y --**;
    col2 = strip(col2) || '> Add to e-mail list' ;
    output ;
run ;

title '<form name="ideas" action="/cgi-bin/broker" method="pre">'
    '<input type="hidden" name="_service" value="bkc"> '

```



```

        put '    for (var i=0; i < document.ideas.state.length; i++) {';
        put '        if (document.ideas.state.options[i].value == currval) {';
        put '            var sel=i;';
        put '        }';
        put '    }';
        put '    if (sel >= 0 ) {';
        put '        document.ideas.state.options[sel].selected = true;';
        put '    }';
        put '    else {';
        put '        document.ideas.state.options[0].selected = true;';
        put '    }';

        put '}' ;

    end;
run;

```

states.js (resulting javascript program)

```

// this file is generated by scsug_create_js.sas
function loadstate(nameobj) {
    with (nameobj) {
        options.length=3;
        options[0].text="AZ";
        options[0].value="AZ";
        options[1].text="LA";
        options[1].value="LA";
        options[2].text="TX";
        options[2].value="TX";
    }
}
function setstate(currval) {
    for (var i=0; i < document.ideas.state.length; i++) {
        if (document.ideas.state.options[i].value == currval) {
            var sel=i;
        }
    }
    if (sel >= 0 ) {
        document.ideas.state.options[sel].selected = true;
    }
    else {
        document.ideas.state.options[0].selected = true;
    }
}
}

```

scsug_report_form2.sas

```

*-- Change to see other records --*;
%let partname=Cran ;

data info ;
    length title $ 3 name city $ 30 state $ 2 email $ 1 ;
    title= 'Mr.' ;
    name= 'Dan Hammarstrom' ;
    City= 'Tucson' ;
    State='AZ' ;
    email='Y' ;
    output ;
    title= ' ' ;
    name= 'Keith Cranford' ;
    City= 'Baton Rouge' ;
    State='LA' ;
    email='N' ;
    output ;
    title= 'Ms.' ;
    name= 'Paula Linderman' ;
    City= 'Victoria' ;
    State='TX' ;
    email='N' ;
    output ;

```

[illegible]

```

footnote2 '<script>loadstate(document.ideas.state);</script>' ;
footnote3 '<script>setstate("'" "&currval" '"');</script>' ;
footnote4 '<script>document.ideas.name.focus();'
           'document.ideas.name.select();</script>' ;

title2 'Using PROC REPORT to Create HTML Form';
title3 'SCSUG SAS Educational Forum 2007';
title4 'Data-driven drop-down list' ;
proc report data=form nowd ;
    column col1 col2 ;
        define col1 / ' ' style(column)=[background=#F1EFE2] ;
        define col2 / ' ' style(column)=[protectspecialchars=No] ;

run ;

ods html close;

```