Paper 6960-2016

CREATING AMAZING VISUALISATIONS WITH SAS® STORED PROCESSES AND JAVASCRIPT LIBRARIES

Phil Mason, Wood Street Consultants

ABSTRACT

I have been using SAS for over 30 years and developing applications with SAS for over 20 years now. In the last 10 years I have been developing these SAS applications so that they run in a web browser so that the applications can be delivered more easily to users everywhere. I have used HTML and JavaScript to make the user interface in the browser while SAS feeds data and images to it from the background. I have used Stored Processes to prepare the data and graphs that are needed in the right format needed to be consumed by the HTML and JavaScript. The SAS Stored Process Web Application provides the technology that enables the Stored Process to be run from the web browser and for the results to be streamed to the web browser. In this paper I will be describing how Stored Processes work, how to use them with HTML/JavaScript and techniques that can be used to do this effectively to make Web Applications.

Stored Processes

Stored Processes were introduced in SAS 9 and are similar to a SAS macro, except they have some extra information attached. There are 2 parts to a stored process: the SAS code, which is run when the stored process is executed & the metadata for the stored process which holds information about it like where it will run, who can run it, parameters that can be used, etc.

When a stored process is run, it is actually run on behalf of a user by a special user id. If you have configured SAS in the recommended way then Stored Processes will usually be run under the SASSRV user-id. So if a user called PMASON tried to run a stored process, it would check whether that user was allowed to run that stored process and if so it would be run on the requested server (probably a stored process server) using the SASSRV user-id. This is an important fact to be aware of when designing applications particularly for UNIX systems which are very fussy about permissions.

Creating Stored Processes

When creating a stored process it is often easiest to use Enterprise Guide, since you can use wizards to create code or write your own, test it out and then save the code as a stored process. A wizard will guide you through the process and allow you to specify everything in an easy way.

Another way is to create the metadata for the stored process using the SAS Management Console. This allows everything to be specified, including where the source code is located. You then need to write the source code for the stored process separately and ensure that it is in place when you try to use the stored process. If doing this, then there are a few things you will need to know about the structure of stored processes.

The SAS code for a stored process can be as simple as a normal everyday SAS program. For instance I could have a data step and a Proc Print in a file called test.sas, and that would be all that was required. In my stored process metadata I would need to point to that code so that when the stored process was run it would load that SAS code in and execute it. However by making use of 2 other lines of code you can get a lot more power out of a stored process.

```
%stpbegin;
```

This macro initializes the Output Delivery System for use from a stored process. By setting various macro variables you can affect what this macro does. For example, by setting the _ODSDEST macro to RTF will cause the macro to produce RTF output.

```
%stpend;
```

This macro finalizes the ODS output. For example if we had been writing HTML, it would write the final HTML tags such as </body> and </html>.

Of these 2 things, %stpbegin is the most complex to understand since it can make use of over 40 reserved macro variables to control what it does. Some of the more useful of these and ways to use them will be explained later.

Stored Process Web Application

The SAS Stored Process Web Application enables Stored Processes to be run from a web browser and then can stream the results back to the browser. This is the single most useful new facility that SAS have provided in the last decade – in my opinion. This is because it means that SAS code can be run from almost any place. For example, I can go into Microsoft EXCEL and enter a URL which runs the web application and produces a table – that table will then be imported into EXCEL automatically. Another example, at a previous client of mine we

built a java application which simply constructed URLs to run the web application and then read the results that were streamed back.

How to create a simple Web Application using Stored Processes

This section will describe how you can create a simple web application. It will take you through a series of simple steps which show how to create a report in Enterprise Guide, make into a Stored Process, run it in various ways, modify it and finally build a simple web application using it. You can skip some of these steps, but I wanted to show how someone with almost no knowledge of SAS could actually make a web application using Stored Processes.

1) In Enterprise Guide start the Query Builder
2) Now you can open some data to start building a query from. I picked a standard sample dataset from our SAS 9.4 installation - sashelp.orsales.
3) Now you can add tables, variables & join tables. I just added them all in.
4) You can also filter data, sort data, computed columns, etc. You can also click on Preview to see the SQL code that was produced. So if you know how to code in SAS then you could skip this wizard and just create the code yourself.
5) Now having made the query, we can convert it into a Stored Process. Right click on Query Builder and select Create Stored Process.
6) Now use the wizard to create a stored process and give it a name. You can fill in the other fields although you can leave them to default.
7) Press next to see the SAS code of the Stored Process being created.
8) Press next to see the execution options, which you probably just want to let default. You might need to modify the source code repository field so that you can ensure the SAS code is stored where you want it to be.
9) Hit next and you will see what librefs were used in the code and whether you need to create references for any of them. In this case we have just used some built-in ones which are automatically defined, and so need to do nothing else.
10) Hit next and you see a screen where you can define prompts, which can be used to prompt the user for values when a stored process is run. The values can then be passed through to the stored process code as macro variables. We are not defining any prompts at this stage.
11) Hit next and you see the page where we can define input & output streams for the stored process. Our simple stored process won't need any of these.
12) Finally hit next and you will see a summary page showing key information about the Stored Process you have created.
13) Hit finish and the Stored Process is created. This stored process can then be run and it creates a dataset based on the query that we built. However we want to see that dataset on the screen, so we will modify the stored process to do that. Now we need to right click on the stored process and modify it. We can add a proc print or similar o show the data at the end. Save it and run it to test.
14) Now you will need to find how to access your Stored Process Web Application. To open the Stored Process Web Application at my site we use this link http://my_server/SASStoredProcess/. This will show us the Stored Process Web application home page,
15) Select "List Available Stored Processes and Reports". Then drill down through tree to show your stored process from the location in the metadata that you saved it.
16) Click on your stored process to run it. The results show up on the page.
17) Right click on your stored process and copy the link address. This link will let us run the stored process from a number of other places.
18) Paste the link into the URL box in the browser, and hit return to run it. You now have the complete URL that can be used to call your Stored Process from anywhere.
19) To show how flexible this is we will run the stored process from EXCEL. The assumes you have the Office Add-in installed. Open EXCEL. Select SAS menu item, then click on Reports.
20) Navigate to your stored process and open it.
21) The stored process will run. A little progress bar is displayed while it runs.
22) When the stored process finishes running then the table that it produces will be imported into EXCEL. You now have the results of the Stored Process in EXCEL.
23) Now that we can create a stored process and run it from various places (more ways to run it are listed later), we will add a graph to it. Go back to Enterprise Guide and add a graph to the Stored Process. You can use something like a simple Proc Gchart, or you could use a wizard in Enterprise Guide to help you with this.
24) Once you save your new code, you can run it from the web browser and see the graph & table produced there.
25) We might like to add a filter to our stored process, so lets go back to EG. We will add a parameter to let us filter on a variable (e.g. product_line). You can then use a macro variable for the value of the that

variable (e.g. where product_line="&product_line";). This means that by changing the value of the macro variable we can apply a different filter.

26) Now we can add a prompt for this macro variable. The wizard will search our code for macro variables and allow us to define them as prompts. We can just use the defaults.
27) Run the stored process again. You will be prompted for a value, so enter one. Make sure your value matches one of the values from the data, otherwise you won't find anything. Then click on Run to execute the stored process using the value you entered.
28) You now see the stored process with your parameter applied.
29) If you want to use a URL to pass your parameter to your stored process, you can do so by making use of one of the key features of the Stored Process Web Application. Any parameter/value pairs like &parameter=value will be passed into the SAS code as macro variables. They don't even have to be pre-defined in the SAS code. So that means that I can call our stored process using the following URL to pass the value in, e.g. http://my_server/SASStoredProcess/do?_program=%2FShared+Data%2FSASTesting%2FTest3&product_line=Children
30) We can make a simple HTML file which allows us to select the report we want to run from a menu. So the following code simply calls our stored process and passes a different value for product_line each time.

```
<html>
<body>
<h1>Pick a report to run</h1>
<a href="http://d351tq92/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Ftest&product_line=Children"> Children</a><p>
<a href="http://d351tq92/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Ftest&product_line=Clothes+%26+Shoes">Clothes & Shoes</a><p>
<a href="http://d351tq92/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Ftest&product_line=Outdoors">Outdoors</a><p>
<a href="http://d351tq92/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Ftest&product_line=Sports">Sports</a><p>
</body>
</html>
```

31) This displays a menu.
32) Selecting a value (e.g. Sports) runs the stored process with the appropriate parameter to display the required report.
33) With a little basic HTML knowledge we can modify the HTML to make a better menu. This introduces another useful technique of using HTML forms to run stored processes. The key points in using this technique are:
    - The form tag has
        - Action element which defines the start of the URL to use when calling your stored process.
        - Method element
    - The form uses other tags which are form elements which define what will be on the form. These include:
        - Input tags which define name/value pairs which will be passed to the stored process as parameters. Some of simply define a field where the user can type in a value, but others have special characteristics
            - Input tags with a type of hidden won't be displayed on the form but will be passed to the Stored Process as a parameter. In the example below we are passing the name of the stored process with its parameter _program. You should always have this pointing to your stored process when using this technique.
            - Input tags with a type of submit will display a submit button which can be pressed to run the stored process and pass any values from the form to it.
        - Select tags will create a drop down box of options. This allows the user to choose an option and then the selected value will be passed to the stored process.
34) Our menu now has a drop down menu of choices. You select one and click on run which then adds your selection onto the URL as a parameter.

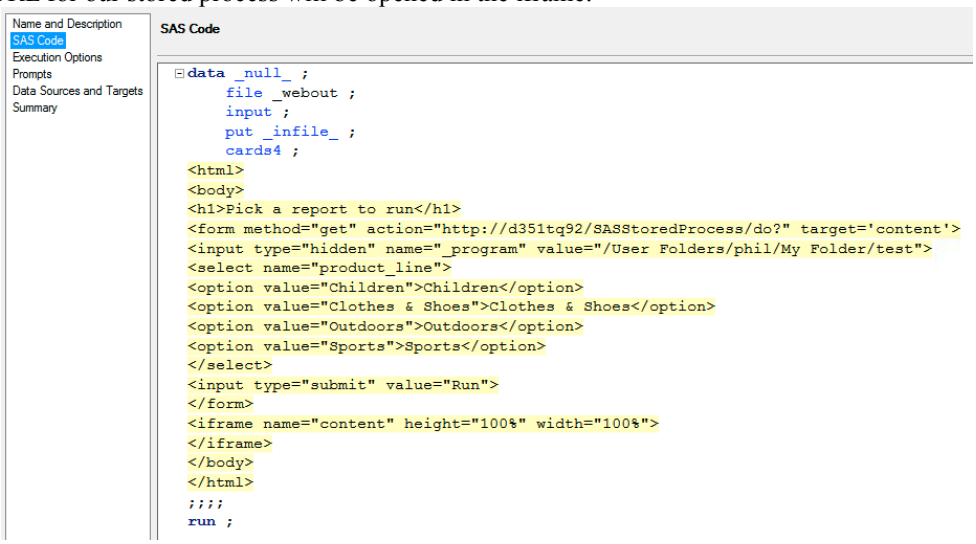
```
<html>
<body>
<h1>Pick a report to run</h1>
<form method="get" action="http://d351tq92/SASStoredProcess/do?">
<input type="hidden" name="_program" value="/User Folders/phil/My Folder/test">
<select name="product_line">
<option value="Children">Children</option>
<option value="Clothes & Shoes">Clothes & Shoes</option>
<option value="Outdoors">Outdoors</option>
<option value="Sports">Sports</option>
</select>
<input type="submit" value="Run">
</form>
</body>
</html>
```

35) To automate this application a little more, we can automatically generate the drop down list of options.

Create a new stored process which will create our HTML menu for us. This can be done by using a stored process that will write the HTML directly into the web browser. To do this you need to write to a fileref called _webout which is predefined for the stored process to use. You also need to turn off the automatically generated Stored Process Macros by using the Include Code For button. These macros usually allocate the _webout fileref for their own use which means that we can't use it from a data step.

- The program below first runs some SQL code which gets the different values of product_line and puts them into option tags, then concatenates them together and puts the result into a macro variable called options. (See screenshot below)
- The program then runs a data step which basically just gets lines from the cards4 area and writes them out to the _webout file ref. After reading a line in we then run it through a resolve() function, which is very important. The resolve() function will resolve any macro language in the line that was read, which means that our options macro variable is resolved and the option lines that were made by our PROC SQL are inserted.
- This generates the same web page that we made before, but now it is data driven and flexible. So if we added another product_line to our data, then we would get another option in our drop down list.

36) We can further improve this by combining the menu and output onto a single page. To do this we add an iframe to our web page specifying the size so it doesn't default to something too small. Then we must add target= to the form tag, specifying a name which matches the one for the iframe. This means the URL for our stored process will be opened in the iframe.



37) Now we have a simple web application that takes some input and updates the page with output based on that.

Enhancing Stored Process

Note: This section had many screen shots which could not fit in this paper due to space. However if you email me I can send you the longer version of this paper which has those screen shots & the presentation.

1. We can use _odsdest to produce output in various formats, rather than the default HTML format. We could add a dropdown for _odsdest to our web page. You can use the HTML select tag to make this. Then we could run it select RTF for example. That would call the stored process passing _odsdest=rtf to it. This makes an RTF file for us which we can open in Microsoft Word.

2. We can enhance our web app again using _odsstyle to choose different ODS styles which control colours, fonts and so on. So we can add a dropdown with a select tag for _odsstyle. Then if we select seaside and run it, then it will produce output using that style. This is because it will have passed _odsstyle=seaside to the stored process. Running it and selecting statistical will produce output using a that style.

3. We can further enhance our web app by using the _debug parameter to get various debug information. So we add checkboxes for each debug option, since there are several that can be specified concurrently, such as log & time. Now if we select the check boxes it will pass parameters for those selected, e.g. _Debug=log&_debug=time. This will let us see the log and time (which is how long it took for the stored process to run).

4. Its interesting to look at the URL that has been generated by our web app to run this. It is :

```
http://my_server/SASStoredProcess/do?_program=%2FShared+Data%2FSASTesting%2FTest3&product_line=Children&_odsdest=html&_odsstyle=meadow&_debug=log&_debug=time
```

5. You can break this URL up into sections to understand what the HTML has generated:

```
http://my_server/SASStoredProcess/do?
_program=%2FShared+Data%2FSASTesting%2FTest3
&product_line=Children
&_Odsdest=html
&_Odsstyle=meadow
&_Debug=log
&_debug=time
```

6. An interesting thing we can see from the log of the stored process is that there are various macro variables which could be used to reconstruct the URL of the stored process call, such as _program, _srvname, _srvport & _url. Using these automatically generated macro variables we can change the hard-coded URL in the stored process to use them. This will mean that if the stored process name changes or the stored process is moved to another place in the metadata then it will still work as expected. So http://my_server/SASStoredProcess/do? Would become http://&_srvname.:&_srvport/&_url.?

7. When the stored process runs, the resolve function will resolve these macro variables. So looking at the HTML code that the stored process has generated we can see how it has substituted the right values to create the HTML.

8. Another very useful parameter which we can pass to the Stored Process Web Application is _result. It can be used to determine how complex your HTML generated will be. For instance, using _result=stream (which is the default) for our current example we would generate 3998 lines of HTML, including almost 2000 lines of CSS code. This is quite a lot for a quite simple report. Using _result=streamfragment would generate 1989 lines of HTML, with no CSS code - and the lines are shorter as they don't use the CSS.

9. Another useful thing to mention is that if we don't specify a target on our form, then when it runs the stored process it will produce a new page in the current window. But if we specify a target and it is an iFrame on our current web page, then that points the output from the stored process into that iFrame on the same page.

```
<form method="get" action="http://&_srvname.:&_srvport./&_url.?" target="content">
```

More ways to enhance your web app

There are many other reserved macro parameters that we can use for the Stored Process Web Application. Some are used to pass information in, and some are just automatically set by the Stored Process Web Application and provide useful information for you to use. I don't have space within my 20 pages to show much about them, but here is a link to material you should make sure you understand - http://support.sas.com/documentation/cdl/en/stpug/68399/HTML/default/viewer.htm#p184mqqbi9w6qjn1q0619x19eg02.htm

Doing more complex things with Stored Processes

The key to this is knowing that you can write raw HTML code to the web page that the web app is creating from your stored process as it runs. This is done by writing the HTML code to the fileref called _webout. One trap to avoid is that you cant write to _webout unless it is free. It won't be free if %stpbegin has run, since it will be being used by ODS. So I usually only use %stpbegin when I want to use ODS to produce some kind of report, and then I turn it off with %stpend when I am finished. If you produce a stored process in Enterprise Guide then SAS helpfully puts and %stpbegin at the start and an %stpend at the end. I usually then remove these so I can just put them where I want them to be. That means that I can drop into a data step anytime and write some HTML or JavaScript to do something. For example:

```
Data _null_ ;
  File _webout ;
  Put '<h1>Make your choices and press submit to continue</h1>' ;
Run ;
```

Another very important thing to point out is that if you use %stpbegin to start writing to HTML, then use %stpend to stop so you can write some custom HTML, then the HTML produced by default is quite interesting and verbose. It will start with an <html> tag for instance, and end with an </html> tag. This is fine if you are just producing one report in a single lump. However if you want to nip in and out of writing custom HTML and have SAS produce reports around what you do, then I have found an incredibly useful undocumented result type. You will usually be using a result type of stream which streams the results to your web browser. However if you use a result type of streamfragment, then SAS will just produce the HTML for the reports and none of the extra tags required. That gives us much more control over what goes on in our HTML. To use this you just need to set the macro variable _result=streamfragment prior to running the stpbegin macro.

Passing cookies to Stored Processes

We can set cookies in JavaScript to pass information between pages and stored processes. For example we could get the screen size and pass to stored process via a cookie. This can let you build output of the right size and quality because you have queried the size of the users screen and passed that information to your SAS code in the Stored Process. You can also use the info in your JavaScript to set the width and height of your objects in your dashboard. The following code is part of a data step that would create code to do this.

```
  put "<script type='text/JavaScript'>" ;
  put "  var x=window.screen.availWidth ;" ;
  put "  var y=window.screen.availHeight ;" ;
  put "  document.cookie = '_gopt_xpixels=' + x ;" ;
  put "  document.cookie = '_gopt_ypixels=' + y ;" ;
  put '<form name="myform"' ;
  put '  method="get"' ;
  put "  action='http://&_srvname:&_srvport&_url'>" ;
  put "<input type='hidden' name='_program' value='&_program'>" ;
  put "<input type='hidden' name='_debug' value='log'>" ;
```

Linking different stored processes

Another thing that you will often want to do in a web app, is to run one stored process and then have it automatically run another. The best method that I have discovered for doing this is to use some custom HTML. You can use the onLoad method on the body tag in an HTML page which will run some JavaScript after the current web page has fully loaded. This is exactly what we need to link stored processes. An example of this is in an application I have which links several stored processes together. This first produces a web page to choose a study and the user clicks on submit – that runs another which saves the selection to a parameter file – that runs another which loads a list of subjects in the study – which runs another that loads a list of favourites for that study – and so on. Here is some HTML taken from an application which will call refresh the contents of another HTML iFrame, which runs another stored process to update it.

```
data _null_ ;
  file _webout ;
  put '</head>';
  put '<body' ;
  put " var x = window.parent.document.frames.main.location.href.indexOf('cookie_save') ; if (x==-1) "  ;
  put " { window.parent.document.frames.main.location.reload() ; } ;"  ;;
  put '" class="panel">' ;
run ;
```

Get vs. Post method

HTML forms use one of two methods to pass parameters: get or post. I usually use the get method, since when the next page has been loaded you can see the entire URL in the address bar or properties, whereas if you use post then you cant see any of the parameters. When using an HTML form with a lot of parameters you may encounter a limit at which the get method can no longer pass parameters since it has a limit of 2083 characters. I encountered this when I wrote a stored process to build filters. After adding about 10 lines of filters it all stopped working. I eventually discovered that this was because I had hit the limit, and so parameters were just being truncated which produced unpredictable results. By switching to the post method the problem instantly went away.

What happens when you pass many parameters of the same name? With the web application if you pass in a single parameter, then it becomes available to the stored process as a macro variable of the same name. e.g. “&name=phil” on the URL is equivalent to “%let name=phil ;”. However if you pass two or more parameters in of the same name, then you get a series of macro variables created. One has a suffix of 0, and provides a count of how many parameters there are. Then the first one has a suffix of 1, the second a suffix of 2, and so on. For instance, “&name=phil&name=mike” is equivalent to “%let name0=2 ; %let name1=phil ; %let name2=mike ;”. The following macro takes a list of HTML parameters and puts them into a macro variable where they can be used with the in operator and a where clause.

```
%macro html_parms_to_list(
                  in,
                  out,
                  default=_:,   /* optional value for default */
                  sep=%str( ),  /* optional one char separator */
                  quote=0,      /* 1=quote values, 0=dont */
                  partstmt=0    /* 1=make part of where, 0=dont */
                    ) ;
 %global &out ;
 %let &out= ;
 %if &quote %then
```

```
    %let _q_=%str(%') ;
  %else
    %let _q_= ;
  %if %symexist(&in.0) %then
    %do ;
      %do j=1 %to &&&in.0 ;
        %let &out=&&&out..&sep.%superq(_q_)&&&in.&j%superq(_q_) ;
      %end ;
    %end ;
  %else
    %if %symexist(&in) %then
      %let &out=%superq(_q_)&&&in%superq(_q_) ;
    %else
      %let &out=%superq(_q_)&default%superq(_q_) ;
  %if %symexist(&in.0) %then
    %let &out=%qsubstr(%superq(&out),2) ;
  %if &partstmt=1 %then
    %do ;
      %if %symexist(&in) %then
        %do ;
          %if %superq(&in)=_ALL_ %then
            %let &out= ;
          %else
            %let &out=and &in in (%superq(&out)) ;
        %end ;
      %else
        %let &out=and &in in (%superq(&out)) ;
    %end ;
%mend html_parms_to_list ;
```

Persistence

When I started developing web applications I looked at ways that I could have persistence of data, since I needed to be able to make some choices in one stored process and then use those choices in another (for example). I found that there were a range of ways that could be used to achieve this:

1) Passing parameters on URL. When building up a URL to call a stored process using the web application, you can add more and more parameters onto the URL to pass information from the current stored process to the next. If you build a form in the HTML to call the stored process, then you can have hidden values on it which will then pass those values to the next stored process.

2) Sessions. This is a method provided by SAS in order to pass parameters on from one stored process to another. The idea is that you put name all macro variables you want to save starting with "SAVE_", and you put all datasets to save into a libref of SAVE. You then use the function stpsrv_session to create a session. You get two macro variables that identify this session and must be used to make use of the session in another stored process. One major drawback to all this is that a saved session must be used on the same stored process server that it was saved on – this can have performance implications. We find that sometimes a stored process server will hang, and that would mean the saved session would be inaccessible.

3) Cookies. One problem with using cookies is that you cant directly read or write a cookie from SAS. So you end up having to manipulate JavaScript which does the reading and writing for you. Then you have to get that information into SAS. Another problem is that a cookie is limited to 4096 bytes. This became a problem when I allowed users to build filters that returned lists of thousands of items which I then wanted to pass to other stored processes. I then had to split my data into chunks of less than 4096 bytes and stored in a series of cookies, which added more complexity. The final problem I found was that cookies just did not always work 100% of the time (using Internet Explorer 6). There were some cases when strange things would happen, yet my code looked OK – and it would work in a different web browser. This unreliability ultimately made me look at alternatives.

4) Saving data to files/datasets. I found this method to be the most reliable. I can write information to a dataset and then load it back in when I want it. A couple of key points that make this possible is that I save each users parameters in a different SAS dataset named as their userid. i.e. if the userid was U1234 then the dataset is called U123. This eliminates problems of file locking if I used a single dataset for writing everyones parameters to. Where I do have parameters that I want to share between people I do write them all to a single dataset, but I have implemented a locking macro since otherwise I would get locking errors.

Data Store

You could also store data in a JavaScript data store. This can be useful if you are extensively using JavaScript objects, particularly for tabular data that will be displayed in grids or graphs. Such a data store can be written to a file on a server and then used to drive a JavaScript object directly.

Databases

You could write any kind of data to a database which would then be accessible later. Since the SAS stored process is so flexible it means that we can call a URL to write something to a database and then be calling another stored process we can retrieve information from a database and format it in whatever form is required.

Creating a more complex web application

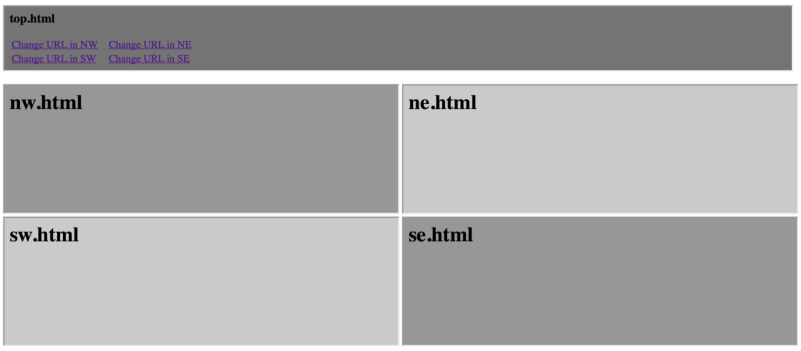
I think the best way to start doing this is to sketch some ideas on paper. A basic design could include a grid of objects and I have used this as a starting point for several clients. You can quickly make a simple prototype using plain HTML and iFrames. The following HTML shows a simple layout made by using an HTML table and putting iFrames in each cell. We have one cell at the top, and then 2x2 cells below.

```
<html> <head> </head>
<body> <table>
<tr> <td colspan='2'> <h1>Dashboard</h1> </td> </tr>
<tr> <td colspan='2'> <IFRAME SRC="top.html" WIDTH=1200 HEIGHT=100></iframe><p> </td> </tr>
<tr> <td> <IFRAME id='nw' name='nw' SRC="nw.html" WIDTH=600 HEIGHT=200></iframe> </td>
     <td> <IFRAME id='ne' SRC="ne.html" WIDTH=600 HEIGHT=200></iframe> </td> </tr>
<tr> <td> <IFRAME id='sw' SRC="sw.html" WIDTH=600 HEIGHT=200></iframe> </td>
     <td> <IFRAME id='se' SRC="se.html" WIDTH=600 HEIGHT=200></iframe> </td> </tr>
</table> </body> </html>
```

The cell at the top of the table covers both colums of the 2x2 cells below. You can create a kind of menu to go in the top cell so that selections made in it will update the content in the other cells. The code below can be used for top.html. It has anchor tags which use the onClick attribute. This runs a simple bit of JavaScript which locates the iframe using its id and then replaces the URL for it. These could be replaced with URLs for stored processes, allowing you to call 4 stored processes.

```
<html>
<body style='background:#777'>
<h3>top.html</h3>
<table><tr>
<td><a href='#' onclick='window.parent.document.getElementById("nw").src="http://www.sas.com";'>Change
URL in NW</a>   </td>
<td><a href='#' onclick='window.parent.document.getElementById("ne").src="http://www.apple.com";'>Change
URL in NE</a>   </td>
</tr><tr>
<td><a href='#' onclick='window.parent.document.getElementById("sw").src="http://support.sas.com";'>Change
URL in SW</a></td>
<td><a href='#' onclick='window.parent.document.getElementById("se").src="http://www.bbc.co.uk";'>Change
URL in SE</a></td>
</tr></table>
</body>
</html>
```

Dashboard



Clicking on these links fills each iFrame with the link sent to it. The top iFrame should update other iFrames. You need to decide how to handle passing info from one frame to another. There are a range of possibilities for doing this:

- top could make change in SAS dataset, then tell others to update and they would use the change in SAS data to update.
- top could update a cookie, then tell others to update which would read the cookie
- Top could make a selection which would then update itself and call other ones with updated values.

- with JavaScript there are other ways if using Ext JS for instance

Next you could design a collection of Stored Processes which could be used to fill the 2x2 cells. Each one will eventually populate our application iFrames.You could actually just use one stored process since you could pass different parameters to the stored process in each iFrame to produce different output. You could produce graphs, tables, maps, images, etc. with your stored process(es).

Once you have your stored process(es) you need to fill each iFrame with a stored process call. You can have one dropdown call all 4 stored processes. You can handle communication between selections in the top and other objects like this:

- make change in the top perhaps in a drop down box, which uses an HTML select tag
- hit a button to reload that iframe, or if you only have one drop down then you can detect a change and automatically reload it. You could use some HTML a bit like this
- <select name="kpi_number" onchange="this.form.submit();">
- when the stored process is reloaded you will know that a parameter has been passed in since you can test for it using %symexist. You can then update a SAS dataset with that parameter value. This will provide a way that other stored processes can pick up the value that was selected.
- next you can have some JavaScript that says when the page has finished loading you can call a function. That function will update the other objects on the screen, each of which will be able to load the value of the parameter from the SAS dataset and make use of it. You could use some HTML like this. We setup a JavaScript function which causes each of the iFrames to reload. When they reload they will check the SAS dataset we use for passing parameters and then make use of any that have been selected. The other important things here is that on the HTML body tag we have used the unload attribute. This means that when the HTML page has finished loading it will call the JavaScript function, which will reload all the other iFrames. You can also see in this code that when a selection is made of a KPI then the onchange attribute will submit the form, which actually reloads the page. So this little piece of HTML & JavaScript demonstrates the key to having a change in one object update all other objects on the screen with that change.

```
put '<script type="text/JavaScript">' ;
put '  function doLoad() {' ;
put '    window.parent.document.getElementById("ne").contentWindow.location.reload() ;' ;
put '    window.parent.document.getElementById("nw").contentWindow.location.reload() ;' ;
put '    window.parent.document.getElementById("se").contentWindow.location.reload() ;' ;
put '    window.parent.document.getElementById("sw").contentWindow.location.reload() ;' ;
put '  }' ;
put '</script>' ;
put '</head>' ;
put '<body onload="doLoad()">' ;
put 'KPI <select name="kpi_number" onchange="this.form.submit();">' ;
put "&kpilist" ;
put '</select>' ;
```

Ways to link to Stored Processes

There are many ways to do this such as:

- Clicking on text that was created with an anchor tag in HTML to link to a stored process.
- Clicking on part of a graph which has used an HTML option to specify a variable with values that are used to create and HTML map to allow you to click on parts of a graph and link to a stored process.
- Pressing a button made using HTML with some JavaScript.
- Select an item from a drop down, with some JavaScript to carry out the link

When you link from a web page to a stored process or stored process to another, you often need to send some information. There are many ways to do this, some being:

- keep data on client in hidden form fields or URL parameters using input tags with type='hidden'
- keep on client in cookies
- keep on server using stored process sessions
- keep on server in a table accessible by stored processes

Passing Parameters to Stored Processes

There are many ways to pass parameters to a stored process.

- passing parameters to a stored process via URL
  - to pass region=West
    http://yourserver/SASStoredProcess/do?_program=/WebApps/Sales/Weekly+Report&region=West
  - + stands in for a space

    - &region might cause a problem depending on where you use it in SAS, since it can be interpreted as a macro variable, so you can use HTML encoding ... &region=West
    - you can use the SAS function htmlencode to do this encoding for you
- Passing parameters to a stored process using HTML forms
    - Action attribute points to a stored process web application
    - Get method makes url with all parameters on it
        - good for bookmarking and understanding whats going on
        - not good if very long
        - possible security risk
    - Post method means you cant see parameters in URL
        - handles any number of parameters
        - Useful if you don't want user to see all the detail on the URL
        - Able to handle much longer URLs, whereas GET would cut the URL off when it runs out of space which can lead to unpredictable results
    - Specifying _action=properties will show the prompt page if there are prompts specified for the stored process
        - This allows the user to enter parameter values that will be used by the stored process.
    - Specifying _action=form,properties,execute will first display a custom input form if there is one, it not will show a prompt page if there are any parameters, otherwise will just execute the stored process.
        - this is the default when calling a stored process from the portal

Using Popup windows

Sometimes you will want a secondary window to pop up so that you can make some selections before going back to a main window and applying those selections. This can be done from JavaScript by using window.open. The following line of code shows how we write some JavaScript which opens a stored process in a new window which is small like a popup window.

```
Data _null_ ;
  File _webout ;
  put "<a href='#'
onClick='window.open(""http://&_srvname:&_srvport/SASStoredProcess/do?_program=SBIP%3A%2F%2FFound
ation%2F&env.%2Fbis%2Fmedmon%2Flb_boxplotgroup%28StoredProcess%29"","" ;
  put """Menu"",""menubar=no,width=430,height=360,toolbar=no"") ;'>" ;
 run ;
```

Providing status updates

At the bottom left of a web page there is a status area. You can write to this area using JavaScript. There was a macro that used this on a prior page.

```
window.status="This is a message".
```

Having multiple buttons on a screen to do different things

I have a filter builder which builds up a complex form for all the user selections. One of the nice features of this is the ability to click on a plus or minus icon to add a new filter line or remove one. These buttons are able to do different things by using the onClick attribute to specify some JavaScript to run when they are clicked on. The JavaScript goes and updates a hidden text fields to indicate if the plus or minus was clicked on, and which line it was on. The icon is also a submit button and so the form is submitted with the modified fields so that the stored process can then act on that information.

Example of simple JavaScript interactivity

The following sample code demonstrates a range of things that can be done using JavaScript to add some interactivity. I shall describe each of the features used and what it achieves.

1. In the body section we use onload, which will run a JavaScript function when the HTML page has completed loading. Very useful if you want something to be done once your page loads. We are just using an alert function here which pops up a box with the text in it.
2. Also in the body section we use onunload, which will run a function when the HTML page has been unloaded - which means when it has been closed.
3. We specify onkeypress, which will run the code every time a key is pressed. This will just write in the message area at the bottom of the screen what key was just pressed. This is usually used in case you want to intercept the press of a key and do something based on it. For instance if you displayed a menu then just by pressing '3' you could link to the item at number 3.
4. The onkeyup will run the code specified every time a key that has been pressed is released - and so it comes up. So onkeypress can run code when you press the key down, and onkeyup when you release it.

This gives you a lot of control. In this example we run a JavaScript function called checkkey. It will check the key that was released and display a message only for the 'X' key.

5. The onmouseover function is used when the mouse moves over the HTML item in which this was specified. Here we use it on a link, and so if we move the mouse over that link then the specified code will run. In this case it runs a function called open_popup, and displays a message at the bottom of the screen.
6. The onmouseout function is used to detect when the mouse moves away from the HTML item in which it was used. So in our example here we call close_popup, which closes the popup window which our open_popup function openned. This means that moving the mouse over the link opens a window, and moving it off that link closes the window.
7. Next we use the onmousedown function to open a window when the mouse button is pressed down.
8. Then we use the onmouseup function to close the window when the mouse button is released.
9. You will also see in the open_popup function that the function we use to open the window has a lot of parameters that will let us open windows with all kinds of things present or not.

```
<html>
<head>
<script type='text/JavaScript'>
var popwin = null ;
function open_popup()
{ if (popwin == null)
popwin = window.open('http://www.google.com', '', 'top=150, left=350, width=250, height=50, status=0, toolbar=0,
location=0, menubar=0, directories=0, resizable=0, scrollbars=0') ;}
function close_popup()
{ if (popwin != null)
   {popwin.close() ; popwin = null;}}
function checkKey()
{ var key=String.fromCharCode(window.event.keyCode) ;
  if (key == 'X')
   { alert("You pressed the X key") ; }}
</script>
</head>
<body onload='alert("finished loading");' onunload='alert("finished unloading");' onkeypress='window.status="key
pressed is: " + String.fromCharCode(window.event.keyCode) ;' onkeyup='window.status="key up"; checkKey() ;'>
Pop-up a window with information by moving over <a href='#' onmouseover='open_popup();
window.status="Hovering over the link" ; return true ;' onmouseout='close_popup(); window.status=" " ; return
true ;'>here</a>.
<p>Pop-up a window with information by holding mouse button down <a href='#' onmousedown='open_popup();'
onmouseup='close_popup();'>here</a>.
<p><a href='#' ondblclick='open_popup();'>Double click to open</a>, <a href='#' onclick='close_popup();'>single
click to close here</a>.
<p><a href='#' style='font-size="x-large"' onmousemove='open_popup();'>Move mouse over this to open</a>, <a
style='font-size="x-large"' href='#' onmousemove='close_popup();'>move over this to close</a>.
<p>Press <b>X</b> to make an alert window pop up.
<p>Hold down a key to see what it is in the status bar.
</body>
</Html>
```

Uploading Files

This used to be quite difficult in SAS 9.1 and before, but it is now very easy to upload files from the client to the server. The key to this is using the input type of file, along with the Stored Process Web Application. Applications of this are extensive. You could upload an EXCEL file of data to be analysed by a stored process for example.

The following is a simple example of some HTML which will upload a file for use by a Stored Process.

```
<form action="StoredProcessWebApplicationURL" method="post" enctype="multipart/form-data">
<input type="hidden" name="_program" value="/Path/StoredProcessName">
<table border="0" cellpadding="5">
  <tr>
    <th>Choose a file to upload:</th>
    <td><input type="file" name="myfile"></td>
  </tr>
  <tr>
    <td colspan="2" align="center"><input type="submit" value="OK"></td>
  </tr>
</table>
</Form>
```

Read more about uploading files here …

http://support.sas.com/documentation/cdl/en/stpug/68399/HTML/default/viewer.htm#p0jqbnv7miosidn1xw3tqe3f0hv0.htm

Stored Process Reports

A Stored Process Report is basically a Stored Process output which is cached. This means that if you have a stored process that takes a lot of processing but doesn't need to be run in real-time, then it can be run say once a day and the output made ready for others to use. You setup the Stored Process reports in the management console, and point it to a stored process. Store Process reports produce results packages as output which can then be viewed with the SAS package viewer. This means using them in a web application requires some different techniques, which are not covered here. Read more here …
http://support.sas.com/documentation/cdl/en/stpug/68399/HTML/default/viewer.htm#p0ygb18rvi67ben1dvlbujevraw9.htm

PROC STP

This procedure allows Stored Processes to be executed from a SAS program. This opens up a lot more flexibility and power for the use of stored processes. You can execute them in batch, interactively or on servers. It runs locally but with its own execution environment, so it has its own work library and so on. Here's an example:

```
ods _all_ close;
options metaserver = 'your-metadata-server'
        metaport   =  your-metadata-server-port
        metauser   = 'your-userid'
        metapass   = 'your-password';
proc stp program='/Products/SAS Intelligence Platform/Samples/
   Sample: Cholesterol by Sex and Age Group'
odsout=store;
run;
goptions device=png;
ods html path='your-output-directory' file='test.htm' style=HTMLBlue;
  proc document name=&_ODSDOC (read);
    replay / levels=all;
  run;
ods html close;
```

Read more about the STP procedure here …
http://support.sas.com/documentation/cdl/en/stpug/68399/HTML/default/viewer.htm#p1oh9e7gnrxg4yn1lqfeefi2355l.htm

HTTP Headers

Stored Processes streaming output will always include HTTP headers which describes the output that is being produced. The client using the Stored Process can choose if and how it uses this. If you want to control how these headers are created then there are some things to be aware of.Headers must be defined before _webout is opened. So you need to do it before %stpbegin is called. You can do this using some code like this:

```
data _null_ ;
  file _webout ;
  old = stpsrv_header("Content-type", "text/html; encoding=utf-8");
  old = stpsrv_header("Expires", "Wed, 03 Nov 2004 00:00:00 GMT");
  old = stpsrv_header("Pragma", "nocache");
Run ;
%stpbegin ;
... Rest of stored process …
```

One of the most useful headers that can be set is content-type which can specify things like text/xml, text/plain, text/html, application/vnd.ms-excel. e.g.

```
%let old = %sysfunc(stpsrv_header(Content-type, text/html%str(;) encoding=utf-8);
```

We can use headers to set cookies in the browser like this:

```
old = stpsrv_header("Set-Cookie", "CUSTOMER=WILE_E_COYOTE; path=/SASStoredProcess/do; "
|| "expires=Wed, 06 Nov 2002 23:12:40 GMT");
```

You can then pick up the value of the cookies from the _HTCOOK environment variable.

You can read more about this here …
http://support.sas.com/documentation/cdl/en/stpug/68399/HTML/default/viewer.htm#httphead.htm

PROC JSON

JSON stands for JavaScript Object Notation. It is a way of encoding data for use by computer programs. It is similar to XML but is simpler. One of its main advantages is that it is quite easily

readable by humans, as well as computers. It can represent tabular and hierarchical data structures which makes it very flexible. JSON is widely used on the web as a data source for JavaScript objects.

You can find out all about JSON here … http://www.w3schools.com/json/default.asp

In SAS 9.4 there is a new procedure called PROC JSON which will create data in JSON format from any data that SAS can read. This enables us to create JSON output to be used in JavaScript objects from virtually any other data source. Some options are provided to customize the JSON produced, which enables very flexible JSON output to be created.

Here is a table…

CLASS

Filter and Sort | Query Builder | Where | Data | Describe | Graph

	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84
10	John	M	12	59	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90
13	Louise	F	12	56.3	77
14	Mary	F	15	66.5	112
15	Philip	M	16	72	150
16	Robert	M	12	64.8	128
17	Ronald	M	15	67	133
18	Thomas	M	11	57.5	85
19	William	M	15	66.5	112

Here is the Stored Process code to create some JSON.

```
proc json out=_webout ;
  export &table ;
  run ;
```

Here is some of the JSON that is produced by it, when I feed in a parameter of &table=sashelp.class.


```
{"SASJSONExport":"1.0","SASTableData+CLASS":[{"Name":"Alfred","Sex":"
{"Name":"Barbara","Sex":"F","Age":13,"Height":65.3,"Weight":98},{"Name
{"Name":"Henry","Sex":"M","Age":14,"Height":63.5,"Weight":102.5},{"Nam
{"Name":"Jane","Sex":"F","Age":12,"Height":59.8,"Weight":84.5},{"Name"
{"Name":"Jeffrey","Sex":"M","Age":13,"Height":62.5,"Weight":84},{"Name
{"Name":"Joyce","Sex":"F","Age":11,"Height":51.3,"Weight":50.5},{"Name
{"Name":"Louise","Sex":"F","Age":12,"Height":56.3,"Weight":77},{"Name"
{"Name":"Philip","Sex":"M","Age":16,"Height":72,"Weight":150},{"Name":
{"Name":"Ronald","Sex":"M","Age":15,"Height":67,"Weight":133},{"Name":
{"Name":"William","Sex":"M","Age":15,"Height":66.5,"Weight":112}]}
```


You can trim some extraneous information from the JSON using the **nosastags** option, and layout the JSON in an easier to read form using the **pretty** option on PROC JSON.

If you need to customize the JSON in order to fit some specific requirements for a JavaScript object that you want to use then you are able to use these the write statement to write out extra structure to your JSON. I used the code here to get my JSON in the right format for using with the jqGrid object.

```
 * create a JSON version of the SAS table ;
proc json out=_webout pretty nosastags ;
     write open object ;
     write values "rows" ;
     write open array ;
     export sashelp.orsales ;
     write close ;
     write close ;
 run ;
```

In PROC DS2 there is a method provided for reading JSON data into SAS. There is a great article by Chris Hemedinger that describes how this is used http://blogs.sas.com/content/sasdummy/2015/09/28/parse-json-from-sas/ .

PROC STREAM

There are various ways that we can get code into a web browser. We could just write a simple file and then load that into the web browser, such as by creating a file report.html and opening it. Usually a better way to do this is to use SAS/Intrnet or a Stored Process to stream code to the browser. From a stored process you can do this by writing lines to the _webout fileref. This could be done by writing to it from a data step, but you can also use PROC STREAM to do this.

A simple use to send some HTML to your web browser is to the right.

```
proc stream outfile=_webout ;
BEGIN
<html><h1>Hello World!!</h1></html>
;;;;
```

And a slightly more complex one shows how macro variables and other macro language are all resolved when used within PROC STREAM.

```
proc stream outfile=_webout ;
BEGIN
<html><h1>Hello &name</h1>
The time is %sysfunc(time(),time.)
</html>
;;;;
```

PROC STREAM reads from one fileref and writes the lines to another filrefref. As the lines are written any macro references are resolved. This is a hugely powerful facility. In the simplest example we could have an HTML file where we have a macro variable for the title, which would be replaced as the HTML is streamed to the browser.

PROC STREAM also works well with other kinds of files, such as RTF files. You could make a letter and save it as RTF, replace certain parts with macro variables and then by using PROC STREAM you could effectively carry out a mail merge to produce a customized letter for a set of macro variables.

JAVASCRIPT FRAMEWORKS

These are libraries of code which make it easy to produce web objects and applications. They are usually a combination of JavaScript, CSS and/or HTML code. You can download these libraries to your own servers and use it from there, or access it directly from the authors web site or else you can even use Code Delivery Networks (CDNs). A very popular CDN is https://cdnjs.com/ which has most JavaScript libraries available through it. Another good option is https://developers.google.com/speed/libraries . Here are some of the best ones available in my opinion.

JQUERY

jQuery is a JavaScript library which greatly simplifies JavaScript programming. You can usually achieve quite a lot with far less statements than it would take with standard JavaScript. A great place to learn about jQuery, HTML, CSS and more is at W3C schools web site http://www.w3schools.com/jquery/ Here are some examples of simple jquery. The one on the left will load a text file into a section of a web page when a button is clicked on. The one top-right will hide all paragraphs on a web page when a button is pressed. And the last one does some animation of some sections of HTML when a button is pressed.

```
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").load("demo_test.txt");
    });
});
</script>

<div id="div1">
<h2>Let jQuery AJAX Change This Text</h2>
</div>

<button>Get External Content</button>
```

```
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
```

```
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").fadeIn();
        $("#div2").fadeIn("slow");
        $("#div3").fadeIn(3000);
    });
});
```

Here is a piece of HTML which will run a stored process and load its output into part of a web page when a button is pressed.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
<script>
$(document).ready(function(){
   $("button").click(function(){

$("#div1").load("http://d351tq92/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+Folder%2FExercise+2");
   });
});
</script>
</head>
<body>
<h1>This is how we can load something to this web page</h2>
```

```
<hr>
<div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>
<button>Get External Content</button>
</body>
</html>
```

This produces the following page.

This is how we can load something to this web page

Let jQuery AJAX Change This Text

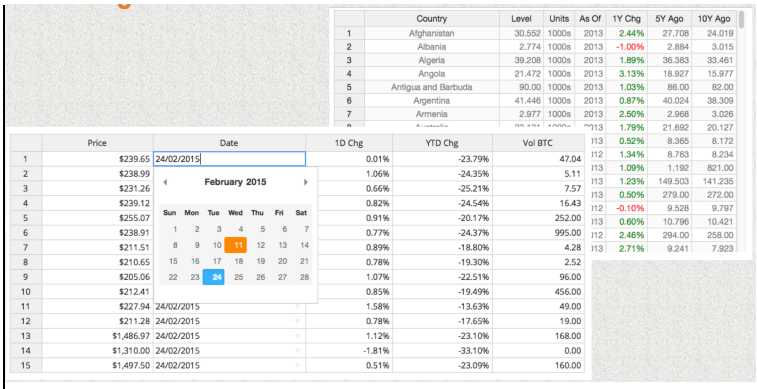
Get External Content

JQGRID

jqGrid is a JavaScript library which enables creating data grids on web pages quite easily.

You can see demos and learn all about jqgrid here … http://www.guriddo.net/demo/guriddojs/ These are the kind of grids you can display on a web page, though there are many more and a huge amount of functionality available.

The following code shows an example of using a grid like this in conjunction with a stored process, which produces JSON formatted data that is then displayed by the JavaScript object as a grid.



	Country	Level	Units	As Of	1Y Chg	5Y Ago	10Y Ago
1	Afghanistan	30.552	1000s	2013	2.44%	27.708	24.019
2	Albania	2.774	1000s	2013	-1.00%	2.884	3.015
3	Algeria	39.208	1000s	2013	1.89%	36.383	33.461
4	Angola	21.472	1000s	2013	3.13%	18.927	15.877
5	Antigua and Barbuda	90.00	1000s	2013	1.03%	86.00	82.00
6	Argentina	41.446	1000s	2013	0.87%	40.024	38.309
7	Armenia	2.977	1000s	2013	2.50%	2.968	3.026

	Price	Date	1D Chg	YTD Chg	Vol BTC
1	$239.65	24/02/2015	0.01%	-23.79%	47.04
2	$238.99		1.06%	-24.35%	5.11
3	$231.26		0.66%	-25.21%	7.57
4	$239.12		0.82%	-24.54%	16.43
5	$255.07		0.91%	-20.17%	252.00
6	$238.91		0.77%	-24.37%	995.00
7	$211.51		0.89%	-18.80%	4.28
8	$210.65		0.78%	-19.30%	2.52
9	$205.06		1.07%	-22.51%	96.00
10	$212.41		0.85%	-19.49%	456.00
11	$227.94	24/02/2015	1.58%	-13.63%	49.00
12	$211.28	24/02/2015	0.78%	-17.65%	19.00
13	$1,486.97	24/02/2015	1.12%	-23.10%	168.00
14	$1,310.00	24/02/2015	-1.81%	-33.10%	0.00
15	$1,497.50	24/02/2015	0.51%	-23.09%	160.00

February 2015

Sun Mon Tue Wed Thu Fri Sat

```
<!DOCTYPE html>

<html lang="en">
<head>
   <!-- The jQuery library is a prerequisite for all jqSuite products -->
   <script type="text/ecmascript" src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.1.1.min.js"></script>
   <!-- This is the JavaScript file of jqGrid -->
   <script type="text/ecmascript" src="http://www.guriddo.net/demo/js/trirand/jquery.jqGrid.min.js"></script>
   <!-- This is the localization file of the grid controlling messages, labels, etc. -->
   <!-- We support more than 40 localizations -->
   <script type="text/ecmascript" src="http://www.guriddo.net/demo/js/trirand/i18n/grid.locale-en.js"></script>
   <!-- A link to a jQuery UI ThemeRoller theme, more than 22 built-in and many more custom -->
   <link rel="stylesheet" type="text/css" media="screen" href="http://www.guriddo.net/demo/css/jquery-ui.css" />
   <!-- The link to the CSS that the grid needs -->
   <link rel="stylesheet" type="text/css" media="screen"
href="http://www.guriddo.net/demo/css/trirand/ui.jqgrid.css" />
   <meta charset="utf-8" />
   <title>jqGrid Loading Data - JSON</title>
</head>
<body>
   <table id="jqGrid"></table>
   <div id="jqGridPager"></div>
<script type="text/JavaScript">

$(document).ready(function () {

                        $("#jqGrid").jqGrid({
                        url:
'http://d351tq92/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Fjson',
                        datatype: "json",
                         colModel: [
                                { label: 'Year', name: 'Year', width: 75 },
                                { label: 'Quarter', name: 'Quarter', width: 90 },
                                { label: 'Product_Line', name: 'Product_Line', width: 100 },
                                { label: 'Product_Category', name: 'Product_Category', width: 100 },
                                { label: 'Product_Group', name: 'Product_Group', width: 100 },
                                { label: 'Profit', name: 'Profit', width: 80, sorttype: 'integer' },
                                { label: 'Total_Retail_Price', name: 'Total_Retail_Price', width: 80, sorttype: 'integer' },
                                // sorttype is used only if the data is loaded locally or loadonce is set to true
                                { label: 'Quantity', name: 'Quantity', width: 80, sorttype: 'number' }
```

```
                    ],
                    viewrecords: true, // show the current page, data range and total records on the toolbar
                    width: 780,
                    height: 400,
                    rowNum: 30,
                    loadonce: true, // this is just for the demo
                    pager: "#jqGridPager"
            });
});

 </script>
</body>
</html>
```

The following SAS code is for the stored process (json) that delivers the data to this grid. Here I have looked at the format of the JSON data that the grid needs, and used some nice options available in SAS 9.4 that allows me to customize my JSON data to match what is required exactly.

```
* create a JSON version of the SAS table ;
proc json out=_webout pretty nosastags ;
   write open object ;
   write values "rows" ;
   write open array ;
   export sashelp.orsales ;
   write close ;
   write close ;
run ;
```

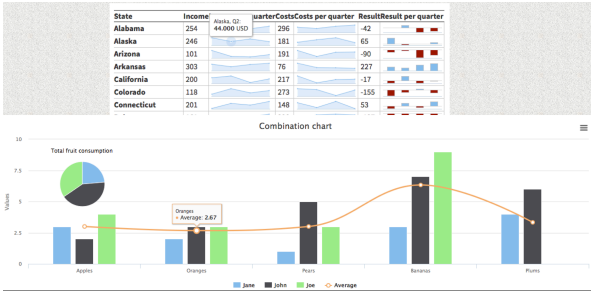
The grid that this produces will look like this, and you can sort the columns.

Year	Quarter	Product_Line	Product_Category	Product_Group	Profit	Total_Retail_P	Quantity
1999	1999Q1	Children	Children Sports	A-Team, Kids	4980.15	8990.9	286
1999	1999Q1	Children	Children Sports	Bathing Suits, Kids	1479.95	2560.4	98
1999	1999Q1	Children	Children Sports	Eclipse, Kid's Cloth	9348.95	18768.8	588
1999	1999Q1	Children	Children Sports	Eclipse, Kid's Shoe	7136.8	14337.2	334
1999	1999Q1	Children	Children Sports	Lucky Guy, Kids	7163	12996.2	303
1999	1999Q1	Children	Children Sports	N.D. Gear, Kids	19153.05	34250.5	755
1999	1999Q1	Children	Children Sports	Olssons, Kids	1975.35	3339.3	209
1999	1999Q1	Children	Children Sports	Orion Kid's Clothes	288.8	580.4	14
1999	1999Q1	Children	Children Sports	Osprey, Kids	7334.7	13219.6	454
1999	1999Q1	Children	Children Sports	Tracker Kid's Cloth	21847.85	40049.5	1243
1999	1999Q1	Children	Children Sports	Ypsilon, Kids	3020.85	5354.7	139
1999	1999Q1	Clothes & Shoes	Clothes	Eclipse Clothing	84982.5	170206.1	2938
1999	1999Q1	Clothes & Shoes	Clothes	Green Tomato	4706.85	7846.2	171
1999	1999Q1	Clothes & Shoes	Clothes	Knitwear	79951.69	140077.94	1554
1999	1999Q1	Clothes & Shoes	Clothes	LSF	16878	32535.5	335
1999	1999Q1	Clothes & Shoes	Clothes	Leisure	14394.3	26047.2	312

Page 1 of 31 View 1 - 30 of 912

HIGHCHARTS

Highcharts is a JavaScript library which enables you to create powerful charts quite easily with a relatively small amount of JavaScript. You can feed the graphs with data from a stored process, like all other JavaScript objects we will look at.



Here is some sample code which provides the data for the graph using a stored process.

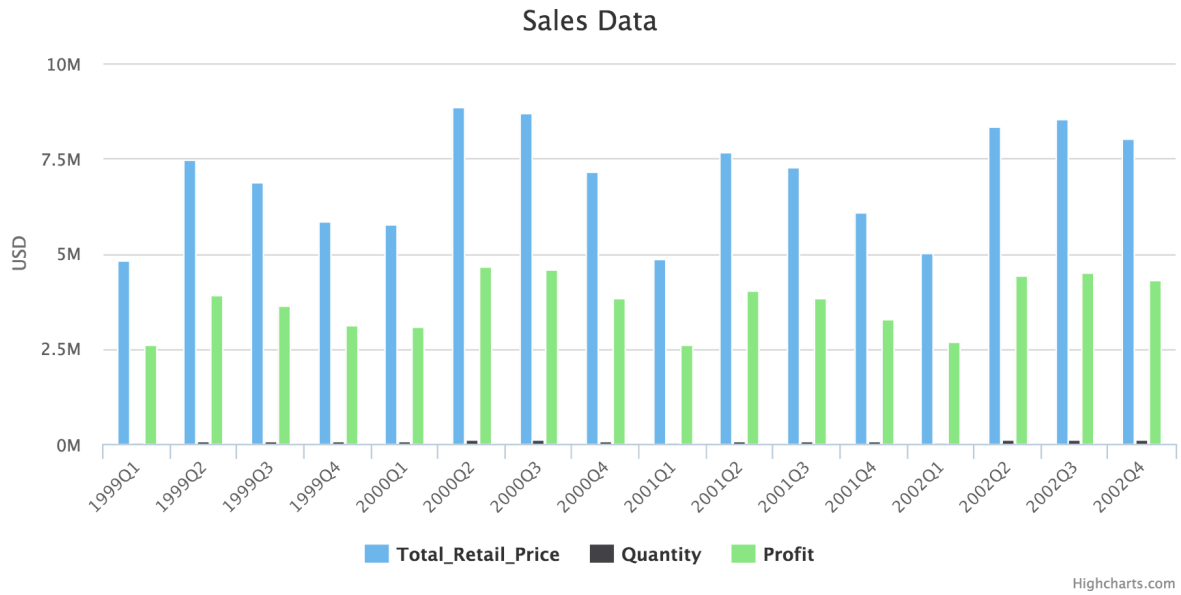
```
<html>
        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

```
                <title>Highcharts Example</title>

                <!-- 1. Add these JavaScript inclusions in the head of your page -->
                <script type="text/JavaScript" src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
                <script type="text/JavaScript" src="http://code.highcharts.com/highcharts.js"></script>
                <script type="text/JavaScript" src="http://code.highcharts.com/modules/data.js"></script>

                <!-- 2. Add the JavaScript to initialize the chart on document ready -->
                <script type="text/JavaScript">
                $(document).ready(function() {

        $.get('http://d351tq92/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+Folder%2F
csv', function(csv) {
                        $('#container').highcharts({
                           chart: { type: 'column' },
                           data: { csv: csv },
                           title: { text: 'Sales Data' },
                                   yAxis: {  title: { text: 'USD' } }
                        });
                     });
                });
                </script>
        </head>
        <body>
                <!-- 3. Add the container -->
                <div id="container" style="width: 800px; height: 400px; margin: 0 auto"></div>
        </body>
</html>
```

The following SAS code is the stored process (csv) which delivers the data to the HighCharts object.

```
proc summary data=sashelp.orsales nway ;
        class Quarter  ;
        var Total_Retail_Price Quantity Profit ;
        output out=sum_orsales(drop=_type_ _freq_) sum= ;
run ;

* create a CSV version of the summary ;
proc export data=sum_orsales outfile=_webout dbms=csv replace ;
run ;
```

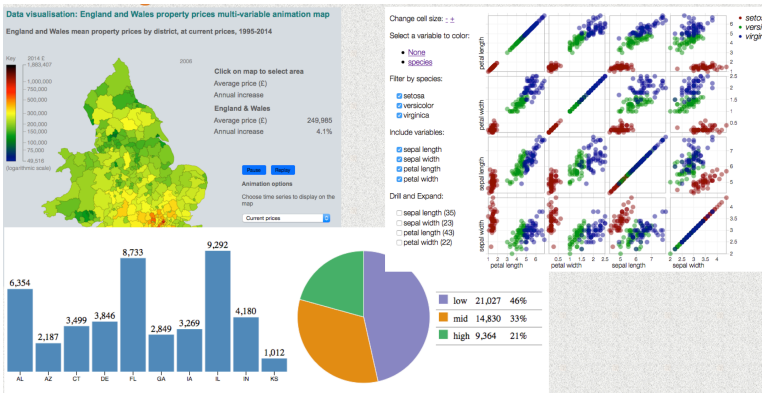
This code produces the following output. One simple thing you can do is to click on the legend and eliminate bars from the graph, or click again to include them.



D3

D3 is a very powerful and hugely popular JavaScript visualization library. It is reasonably

easy to make use of, but is capable of quite complex usage to visualize in almost any way you can think of.



This code uses the D3 library together with the DC & crossFilter libraries which allows you to link objects to create a very powerful visualization. Data is provided using the stored process.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>dc.js - Number Display Example</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="http://dc-js.github.io/dc.js/css/dc.css"/>
</head>
<body>
<h1>Stored Process with simple crossfilter</h1>
<div id="chart-ring-Year"></div>
<div id="chart-hist-spend"></div>
<div id="chart-row-spenders"></div>

<script type="text/JavaScript" src="http://dc-js.github.io/dc.js/js/d3.js"></script>
<script type="text/JavaScript" src="http://dc-js.github.io/dc.js/js/crossfilter.js"></script>
<script type="text/JavaScript" src="http://dc-js.github.io/dc.js/js/dc.js"></script>
<script type="text/JavaScript">

var YearRingChart   = dc.pieChart("#chart-ring-Year"),
    spendHistChart  = dc.barChart("#chart-hist-spend"),
    spenderRowChart = dc.rowChart("#chart-row-spenders");

d3.csv("http://d351tq92/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Fcsv2&table=sashelp.orsales", function(spendData) {

// normalize/parse data
spendData.forEach(function(d) {
    d.Quantity = d.Quantity.match(/\d+/);
});

// set crossfilter
var ndx = crossfilter(spendData),
    YearDim  = ndx.dimension(function(d) {return +d.Year;}),
    spendDim = ndx.dimension(function(d) {return Math.floor(d.Quantity/1000);}),
    Product_CategoryDim  = ndx.dimension(function(d) {return d.Product_Category;}),
    spendPerYear = YearDim.group().reduceSum(function(d) {return +d.Quantity;}),
    spendPerProduct_Category = Product_CategoryDim.group().reduceSum(function(d) {return +d.Quantity;}),
    spendHist    = spendDim.group().reduceCount();

YearRingChart
    .width(300).height(300)
    .dimension(YearDim)
    .group(spendPerYear)
    .innerRadius(50);

spendHistChart
    .width(400).height(300)
    .dimension(spendDim)
    .group(spendHist)
    .x(d3.scale.linear().domain([0,10]))
    .elasticY(true);

spendHistChart.xAxis().tickFormat(function(d) {return d*1000}); // convert back to base unit
spendHistChart.yAxis().ticks(2);
```

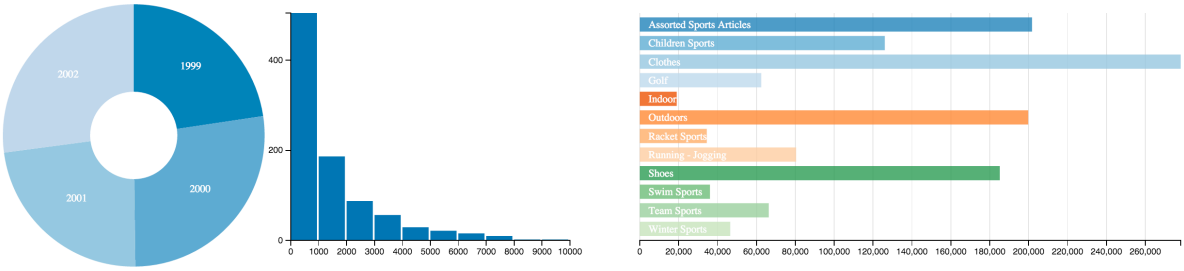
```
spenderRowChart
    .width(700).height(300)
    .dimension(Product_CategoryDim)
    .group(spendPerProduct_Category)
    .elasticX(true);

dc.renderAll();
});

</script>
</body>
</html>
```

The SAS code in the stored process (csv2) which delivers the data is very simple, as follows:

```
proc export data=&table outfile=_webout dbms=csv replace ;
run ;
```

The following output is produced by the code above. The 3 graphs displayed are linked together so that if you click on any bar or pie segment then that is selected and the other graphs change to reflect your selection. This enables some interactive data exploration to be done.

Stored Process with simple crossfilter



GENERAL APPROACH

My general approach to making use of these great frameworks with SAS Stored Processes is to start with one of the huge number of examples provided and to customize it. So I would suggest the following steps to create a web application.

1) Find one or more examples from various frameworks which provide something close to what you want.
    a) If you want a graph, then I suggest HighCharts or AMcharts.
    b) If you want a grid, then I suggest jqGrid.
    c) If you want some unusual or more advanced graphics then I suggest D3, or something based on D3 which makes it simpler to use like C3.
    d) If you want linked graphs on screen which enable you to make selections on one and show the corresponding results in others, I suggest CrossFilter.
    e) If there is something that SAS can easily do just as well as a JavaScript object, then use that. You should check out Rob Alison's web site which shows amazing things that you can make purely using SAS.
2) Most (or all) of these examples will work well together enabling you to combine them on a page using some of the features of HTML, especially using div sections which can easily be loaded with results using jQuery functionality. This enables you to create a layout on a screen if required. If not, just stick with a simple one page layout.
3) Most examples will use JSON, CSV or TSV data as input for the various objects you use. Often this is hard-coded in the examples you may start with, but there are always ways of loading this data from external files. These external files can be a file on the web server, but can also be a file somewhere on the web. The great thing about stored processes is that you can also provide data using a stored process as long as it just streams the content that would otherwise come from a file you would use.

4) You can put the example code you are using into a stored process too and stream it using PROC STREAM. That enables you to use macro variables in the HTML. JavaScript, CSS, etc. and they will be resolved as the code is streamed into the web page. This is a very powerful technique.

So you end up with a stored process that streams code to a web browser and that code gets its data from one or more stored processes.

Safer Alternatives – "Go SAS!"

If creating web apps using HTML & JavaScript seems like overkill, or just seems too hard, then you should consider various SAS solutions that are available which will give you much of what I have described but using a simpler point & click interface. Of course you have to buy the solution from SAS which may not be practical. But you should investigate the following offerings from SAS: OLAP Cube Studio, Visual Analytics, Visual Statistics, JMP, Office Add-in, Enterprise Guide, Web report studio

CONCLUSION

In this paper I have tried to show you how to build a web application step by step. If you follow through the examples then you will build one yourself and have the basis to understand the process and build more. Using JavaScript frameworks allow us to use stored processes with powerful objects which have been pre-built. With just a little bit of knowledge you can modify example code from one of these frameworks and feed it data with SAS Stored Processes using Proc Stream and Proc JSON. You can even parameterize the stored process to make it even more flexible by making a stored process to generate it's HTML, JavaScipt & CSS code. Using these techniques I have built some powerful applications for some major organisations. I hope this will get you started doing the same.

Recommended Links

You will be able to download the code used in my Hands On Workshop to give you lots of examples that relate to what I have covered in this paper. Look for it on the site hosting the conference proceedings.

Code distribution Network for JavaScript libraries - https://cdnjs.com/
Comparison of JavaScript graphing libraries - http://www.jsgraphs.com/
Resources for visualizing data - http://www.visualisingdata.com/resources/
D3 visualization library - http://d3js.org/
JSFiddle to see code & edit in web browser - http://jsfiddle.net/gh/get/jquery/1.9.1/highslide-software/highcharts.com/tree/master/samples/highcharts/demo/scatter/

Validate your JSON code here … http://jsonlint.com

Lots of fabulous examples using SAS Graph to build dashboards and all kinds of things …
http://www.robslink.com/SAS/Home.htm

Contact Information

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

Name:	Philip Mason
Enterprise:	Wood Street Consultants Limited
Address:	21-22 High Street
City, State ZIP:	Wallingford, Oxfordshire, OX10 0BP, England
E-mail:	phil@woodstreet.org.uk