Paper 1272-2014

Developing Web Applications with SAS® Stored Processes

Phil Mason, Wood Street Consultants

ABSTRACT

I have been using SAS for nearly 30 years and developing applications with SAS for over 20 years now. In the last 9 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:

1. The SAS code, which is run when the stored process is executed
2. The metadata for the stored process which holds information about the following:
    i. Which server it will run on, which can be either a stored process server or workspace server.
    ii. Which users are allowed to run it, as well as which users can change the metadata for the stored process.
    iii. What parameters can be used, including any ranges, required parameters and default values.

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 9.3 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://khv-sas-iis.cfsi.local/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.

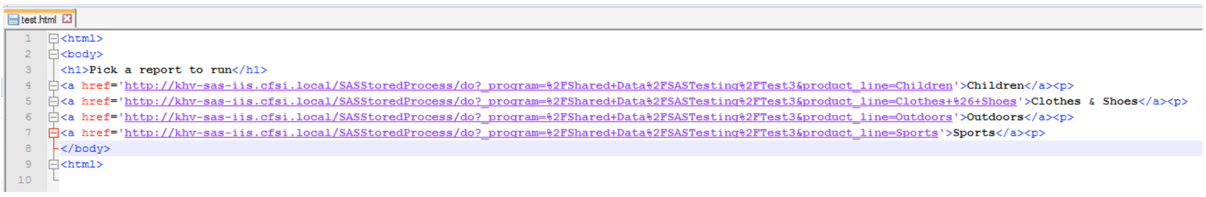
26) Now we can add a prompt for this macro variable. The wizard will search our code for macro variables and allow us to define them as prompts. We can just use the defaults.

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://khv-sas-iis.cfsi.local/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.


test.html

```
<html>
<body>
<h1>Pick a report to run</h1>
<a href='http://khv-sas-iis.cfsi.local/SASStoredProcess/do?_program=%2FShared+Data%2FSASTesting%2FTest3&product_line=Children'>Children</a><p>
<a href='http://khv-sas-iis.cfsi.local/SASStoredProcess/do?_program=%2FShared+Data%2FSASTesting%2FTest3&product_line=Clothes+%26+Shoes'>Clothes & Shoes</a><p>
<a href='http://khv-sas-iis.cfsi.local/SASStoredProcess/do?_program=%2FShared+Data%2FSASTesting%2FTest3&product_line=Outdoors'>Outdoors</a><p>
<a href='http://khv-sas-iis.cfsi.local/SASStoredProcess/do?_program=%2FShared+Data%2FSASTesting%2FTest3&product_line=Sports'>Sports</a><p>
</body>
<html>
```

31) This displays a menu.

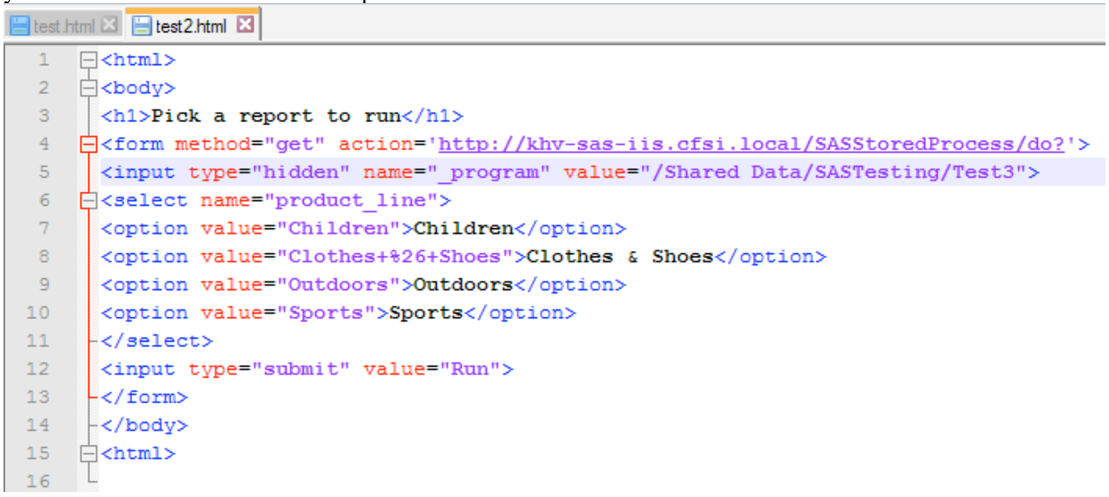
32) Selecting a value (e.g. Sports) runs the stored process with the appropriate parameter to display the required report.

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.



test.html | test2.html

```
<html>
<body>
<h1>Pick a report to run</h1>
<form method="get" action='http://khv-sas-iis.cfsi.local/SASStoredProcess/do?'>
<input type="hidden" name="_program" value="/Shared Data/SASTesting/Test3">
<select name="product_line">
<option value="Children">Children</option>
<option value="Clothes+%26+Shoes">Clothes & Shoes</option>
<option value="Outdoors">Outdoors</option>
<option value="Sports">Sports</option>
</select>
<input type="submit" value="Run">
</form>
</body>
<html>
```

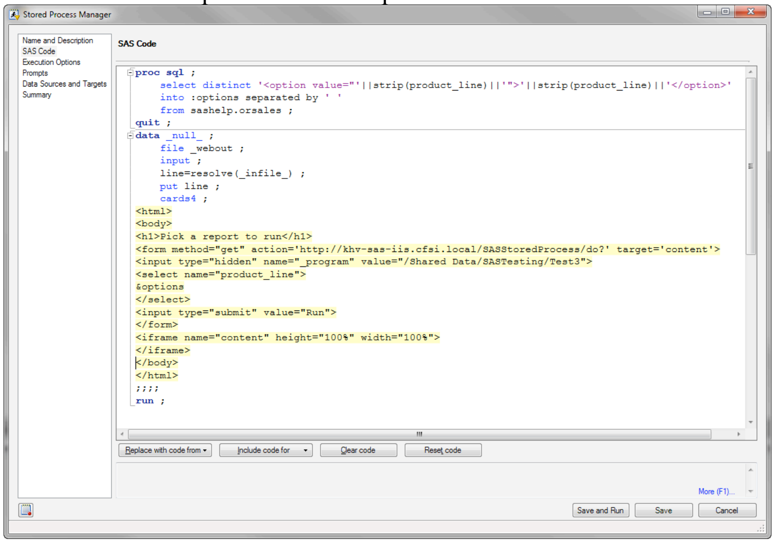
35) To automate this application a little more, we can automatically generate the drop down list of options. Create a new stored process which will create our HTML menu for us. This can be done by using a stored process that will write the HTML directly into the web browser. To do this you need to write 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.



```
proc sql ;
    select distinct '<option value="'||strip(product_line)||'">'||strip(product_line)||'</option>'
    into :options separated by ' '
    from sashelp.orsales ;
quit ;
data _null_ ;
    file _webout ;
    input ;
    line=resolve(_infile_) ;
    put line ;
    cards4 ;
<html>
<body>
<h1>Pick a report to run</h1>
<form method="get" action='http://khv-sas-iis.cfsi.local/SASStoredProcess/do?' target='content'>
<input type="hidden" name="_program" value="/Shared Data/SASTesting/Test3">
<select name="product_line">
&options
</select>
<input type="submit" value="Run">
</form>
<iframe name="content" height="100%" width="100%">
</iframe>
</body>
</html>
;;;;
run ;
```

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

Stored Processes are very flexible and can be used in a large number of ways.

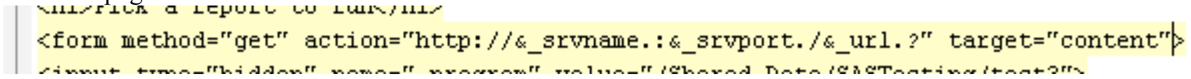
- ♦ Enterprise Guide
  - You can author, edit and test stored processes from EG.
- ♦ SAS Add-in for Microsoft Office
  - Can run stored processes returning their results to EXCEL, Word or PowerPoint
- ♦ JMP
  - Use the JMP Stored Process Packager in Enterprise Guide to prepare it for use with JMP.
- ♦ Information Delivery Portal
  - Can use the SAS Stored Process Navigator to run stored processes
  - Can add stored processes to a Collection portlet
- ♦ Web Report Studio
  - Can insert stored process into sections of a web report
- ♦ DI Studio
  - can publish jobs as stored processes
- ♦ Information Map Studio
  - A stored process can be used to implement information maps, meaning that SAS procedures and data steps can be used in the information map.
- ♦ Stored Process Java & Windows APIs
  - Allow windows and Java programs to call stored processes
- ♦ SAS BI Web Services
  - Provides a web service interface for Stored Processes
- ♦ BI Dashboard
  - Can use stored processes to return data to a dashboard indicator
- ♦ SAS Stored Process Web Application from a web browser
  - Start the SAS Stored Process Web Application, then you can browse to the Stored Process you made and run it
    - http:/myServer/SASStoredProcess/
  - You can run your stored process directly, this wont work if you have a parameter you are prompting for
    - http://myServer/SASStoredProcess/do?&_program=%2FShared+Data%2FSASTesting%2Ftest001
  - You can run your stored process with _action=properties to prompt for any parameters required
    - http://khv-sas-iis.cfsi.local/SASStoredProcess/do?&_program=%2FShared+Data%2FSASTesting%2Ftest001&_action=properties
  - You can start your stored process directly, specifying its location and all other parameters you need
    - http://myServer/SASStoredProcess/do?&_action=form%2Cproperties%2Cexecute%2Cnobanner%2Cnewwindow&_sasapp=Stored+Process+Web+App+9.3&_program=%2FShared+Data%2FSASTesting%2Ftest001

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://khv-sas-
iis.cfsi.local/SASStoredProcess/do?_program=%2FShared+Data%2FSASTesti
ng%2FTest3&product_line=Children&_odsdest=html&_odsstyle=meadow&_debu
g=log&_debug=time
```
5. You can break this URL up into sections to understand what the HTML has generated:
```
http://khv-sas-iis.cfsi.local/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://khv-sas-iis.cfsi.local/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. Some of the most useful ones follow:

Variable	Description
_ACTION	Specifies an action for the Web application to take. Possible values for this variable are as follows: BACKGROUND executes the stored process in the background. DATA … displays a summary of general stored process data. EXECUTE … executes the stored process. FORM … displays a custom input form if one exists. If FORM is the only value for _ACTION, and no form is found, then an error is generated. INDEX … displays a tree of all stored processes. For _ACTION=INDEX, three HTML frames are created with the top frame being the banner frame. LOGOFF … causes the Web application to terminate the active session and to display a logoff screen. NEWWINDOW … displays results in a new window. NOALERT … suppresses generation of alerts. NOBANNER … displays results without adding a banner. PROPERTIES … displays the prompt page, which enables you to set input parameters and execution options and to execute the stored process. SEARCH … displays a search page that enables you to search for a stored process or stored process report. STRIP … removes null parameters. This value can be used only in combination with the

	EXECUTE and BACKGROUND values. TREE … displays a tree of all stored processes. XML …can be combined with other _ACTION values to return XML data.
_DEBUG	Specifies the debugging flags. You can specify one of these or several, e.g. _Debug=time or _debug=time,log,trace FIELDS - Displays the stored process input parameters. TIME - Returns the processing time for the stored process. DUMP - Displays output in hexadecimal format. LOG - Returns the SAS log file. This log is useful for diagnosing problems in the SAS code. TRACE - Traces execution of the stored process. This option is helpful for diagnosing the SAS Stored Process Web Application communication process. You can also use this option to see the HTTP headers that the server returns. LIST - Lists known stored processes. ENV - Displays the SAS Stored Process Web Application environment parameters. You can even use an older bit string method for choosing _debug parameters where each bit selects a different flag. My favourite is to use _debug=2179 which gives you most of the useful information available.
_GOPT_DEVICE _GOPT_HSIZE _GOPT_VSIZE _GOPT_XPIXELS _GOPT_YPIXELS	Sets the corresponding SAS/GRAPH option. For more information, see the DEVICE, HSIZE, VSIZE, XPIXELS, and YPIXELS options in SAS/Graph documentation.
_GOPTIONS	Sets any SAS/GRAPH option.
_HTCOOK	Specifies all of the cookie strings that the client sent with this request. This variable is not set by default but can be enabled.
_METAPERSON	Specifies the person metadata name that is associated with the _METAUSER login variable. The value of this variable can be UNKNOWN. This variable cannot be modified by the client.
_METAUSER	Specifies the login user name that is used to connect to the metadata server. This variable cannot be modified by the client.
_MSOFFICECLIENT	Specifies the Microsoft application that is currently executing the stored process. Valid values for this macro variable are Excel, Word, PowerPoint, and Outlook.
_ODSDEST	Specifies the ODS destination. The default ODS destination is HTML if _ODSDEST is not specified.
_ODSOPTIONS	Specifies options that are to be appended to the ODS statement.
_ODSSTYLE	Sets the ODS STYLE= option. You can specify any ODS style that is valid on your system.
_PROGRAM	Specifies the name of the stored process. The value of _PROGRAM is a path, such as/Sales/Southwest/Quarterly Summary
_RESULT	Specifies the type of client result that is to be created by the stored process. Some of the possible values for this variable are as follows: STATUS - no output to the client. STREAM - output is streamed to the client through _WEBOUT fileref. STREAMFRAGMENT - this is not documented, but works like stream but without all the extra CSS that usually goes along with it. It also means that you don't get tags like <html> and </html> produced for you. You just get the HTML for the thing you are trying to make.
_SRVNAME	Specifies the host name of the server that received the request
_SRVPORT	Specifies the port number on which this request was received
_TARGET	Specifies the fixed HTML form target value to use in displays that are generated by the SAS Stored Process Web Application. Use _blank to always force a new window.
_URL	Specifies the URL of the Web server middle tier that is used to access the stored process
_username	Specifies the value for the user name that is obtained from Web client authentication

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. The following macro has proved to be incredibly useful since it will get a lock on a dataset so that an update to a shared dataset can be made, and then the lock can be released for others to use it. Additionally it will keep trying to get the lock every .01 seconds for up to a minute. Calling it with _type=unlock, will release the lock.

```
%macro locksave(_type=lock,
                _member=,
                _timeout=60,
                _retry=0.01);

   %if &_type=lock %then %do;
      %* set start time;
      %local _starttime;
      %let _starttime=%sysfunc(datetime());
      %* try locking until lock is obtained or until timeout is
exceeded;
      %do %until(&syslckrc=0 or
                 %sysevalf(%sysfunc(datetime())>(&_starttime +
&_timeout)));
         options noerrorabend;
         lock &_member;
         options errorabend;
         %* pause before retrying;
         %let sleep=sleep(&_retry.,1);
      %end;
   %end;
   %else %do;
      %* release lock;
      lock &_member clear;
   %end;
%mend;
```

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>
```

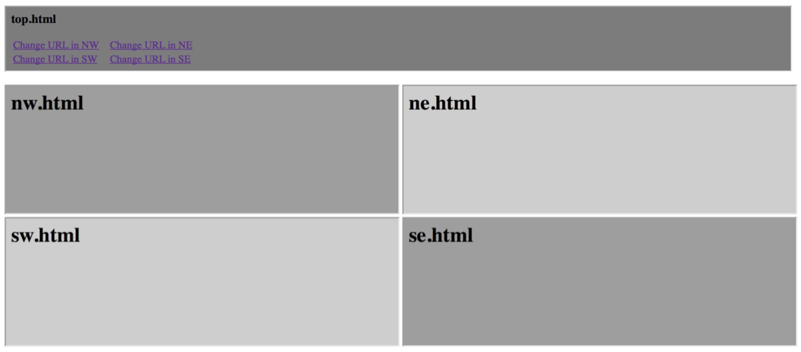
•

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>&nbsp;&nbsp;&nbsp;</td>
<td><a href='#'
onclick='window.parent.document.getElementById("ne").src="http://www.app
le.com";'>Change URL in NE</a>&nbsp;&nbsp;&nbsp;</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.reloa
d() ;' ;
    put '
window.parent.document.getElementById("nw").contentWindow.location.reloa
d() ;' ;
    put '
window.parent.document.getElementById("se").contentWindow.location.reloa
d() ;' ;
```

```
     put '
window.parent.document.getElementById("sw").contentWindow.location.reloa
d() ;' ;
     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 ... &amp;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?_p
rogram=SBIP%3A%2F%2FFoundation%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/62758/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 about Stored Process Reports here
http://support.sas.com/documentation/cdl/en/stpug/62758/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.

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/62758/HTML/default/viewer.htm#p0yy4kd3k4dc03n1mcd76hog6y2u.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/62758/HTML/default/viewer.htm#httphead.htm

How to use Ext JS objects

By far the quickest and easiest way in to using a javascript framework is to pick one that has a design tool available. This then means that you are provided with a GUI environment that helps you to create you web application. The main key to using SAS with javaScript web applications is to build something based on JSON (or XML, although that is more complex for most people) and then to simply replace the JSON with Stored Processes which provide the JSON. That enables you to build something with the design GUI that works, substitute the stored processes in for the JSON and you then have your basic SAS based web app. You get all

the advantages that building a web app with the framework provides, such as nice looking objects with lots of built in functionality, with minimal effort and minimal javaScript expertise required. One reason that I think that ExtJS is the best framework because it has a design tool available called Sencha Architect. You can get this on a free 30 day trial, and they are 30 disconnected days so it may actually last you several months or longer. And it isn't very expensive. You can find it here: https://www.sencha.com/products/architect/
I will cover 2 approaches here. Firstly how we can replace a stored process with a javaScript object, so that we could populate one of the iFrames in our example with it. Secondly, how to just build an application completely in ExtJS.

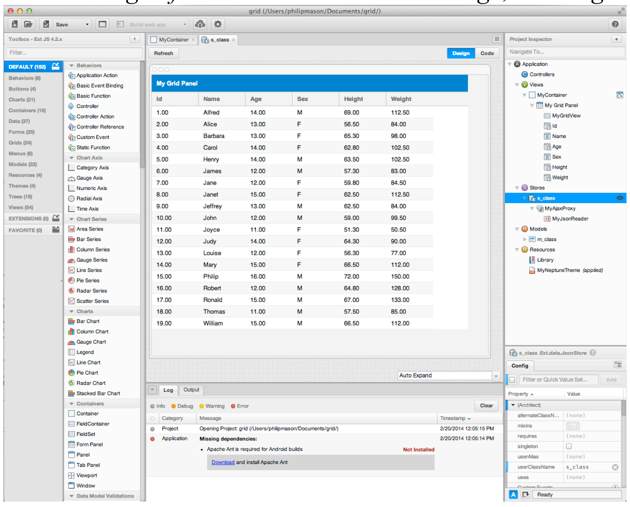
Making an ExtJS grid to replace a SAS table

1) Prerequisites
    - Need a web server available. If running on a single PC you can setup WAMP for windows or MAMP for Mac. This is an easy way to install an Apache web server along with MySQL and PHP.
    - b. Web server should be running before you use Sencha Architect. It is needed to server the JSON sample data.
2) Create some static JSON to use to build the User Interface. It is useful to have some correctly formatted JSON to feed your table display when you are building it.
    - Open an editor and create a JSON using the following guidelines
        - JSON should start and end with a [and].
        - Each row of table should start and end with a { and }.
        - Rows should be separated by a comma.
        - Each variable on a row has the variable name followed by a :.
        - The value of each variable follows the : and depends on the type
    - Character variables should be in single or double quotes
    - Numerics should just appear as numbers
    - Booleans should be either true or false, e.g.

    -
      ```
      [
      {name: "John", age: 12, atSchool: true},
      {name: "Mary", age: 9, atSchool: true},
      {name: "Peter", age: 37, atSchool: false}
      ]
      ```
1) Create an ExtJS application using static JSON
    - Start Sencha Architect
        - Create a new Ext JS 4.2x Project
    - From the Toolbox area on the left do the following
        - Drag a "Container" from the "Containers" section into the Design area
        - Drag a "Grid Panel" from the "Grids" section into the container you just dragged in
        - Drag a "Json Store" from the "Data" section into the design area (not onto your Grid Panel)
        - Drag a "Model" from the "Data" section into the design area (not onto your Grid Panel)
    - Now we need to make some changes to properties in the Project Inspector on the right. First click on the "Code" selector. This will let you see the javaScript code that is created as you set properties in the project.
        - In Resources/Library
            - For better performance you can refer to a local copy of the ext JS library. To do this change "library base path" to point to it. For example if you put it on your web server in the root/htdocs directory you can use a value like "/extjs/" to point to it. Or something like http://localhost/extjs/.
            - Check the "Debug" checkbox, so that the debug version of ext JS will be used which provides a well commented version of the ext-JS code.
        - In Models
            - Go into the properties for the model you added.
            - Find "userClassName" and give it a value, e.g. m_class. This is used by the store to refer to the model
            - Find "fields" and click on the plus icon.
            - Enter the names of your fields separated by commas: e.g. id, name, age, sex, height, weight

        - Select the id field and set the "type" property to "int"
        - Select the name field and set the "type" property to "string"
        - Select the age field and set the "type" property to "int"
        - Select the sex field and set the "type" property to "string"
        - Select the height field and set the "type" property to "float"
        - Select the weight field and set the "type" property to "float"
    - In Stores
        - Go into the properties for the store you added.
        - Find "userClassName" and give them a value, e.g. s_class. This is used by the grid to refer to the store
        - Find "model" and select one from the drop down list, e.g. m_class
        - Find "autoLoad" and set it to true. This makes sure that the store gets loaded automatically when you start the application.
        - Go into the proxy for that store and set the "url" property to point to the JSON created at the start of this, e.g. /sample.json
        - Right click on the store and select "load data".
        - Click on the eye icon to see the data that was loaded.
    - In Views
        - Go into properties for the grid panel you added.
        - Find the "store" property and select the store you defined from the drop down list.
        - Right click on the grid panel item and select "auto columns". This defines all the appropriate column properties for the store you had selected.
    - Save project.
        - Open Settings
        - "URL prefix" should be what must be entered before your application on the URL, e.g. http://localhost/grid_tutorial1
        - "Deploy path" is the location in the file system that your code should be copied to so that it can be used by the web server, e.g. c:\wamp\www\grid_tutorial1
        - Click on save and give it a suitable name.

This shows an image of Sencha Architect at this stage, showing a nice preview of the grid using the JSON data.



grid (/Users/philipmason/Documents/grid/)

My Grid Panel

Id	Name	Age	Sex	Height	Weight
1.00	Alfred	14.00	M	69.00	112.50
2.00	Alice	13.00	F	56.50	84.00
3.00	Barbara	13.00	F	65.30	98.00
4.00	Carol	14.00	F	62.80	102.50
5.00	Henry	14.00	M	63.50	102.50
6.00	James	12.00	M	57.30	83.00
7.00	Jane	12.00	F	59.80	84.50
8.00	Janet	15.00	F	62.50	112.50
9.00	Jeffrey	13.00	M	62.50	84.00
10.00	John	12.00	M	59.00	99.50
11.00	Joyce	11.00	F	51.30	50.50
12.00	Judy	14.00	F	64.30	90.00
13.00	Louise	12.00	F	56.30	77.00
14.00	Mary	15.00	F	66.50	112.00
15.00	Philip	16.00	M	72.00	150.00
16.00	Robert	12.00	M	64.80	128.00
17.00	Ronald	15.00	M	67.00	133.00
18.00	Thomas	11.00	M	57.50	85.00
19.00	William	15.00	M	66.50	112.00

```
Category     Message                                                      Timestamp
Project      Opening Project: grid (/Users/philipmason/Documents/grid/)   2/20/2014 12:05:15 PM
Application  Missing dependencies:                                        2/20/2014 12:05:14 PM
               • Apache Ant is required for Android builds   Not installed
               Download and install Apache Ant
```

1) Test application
    - Use a good browser that provides proper JavaScript debugging tools. So anything except Internet Explorer is usually good. Firefox with the Firebug extension is highly recommended.
    - Click on Publish in the Sencha Architect toolbar and the application is copied to a location on the web server ready for testing.
    - Click on Preview App in the Sencha Architect toolbar and the application is started using the default system browser. You might need to get the URL and copy it to a better browser for debugging.
2) Change JSON store to a dynamic store.
    - Create a stored process that will be used to produce the JSON data.
        - The data can come from a SAS dataset or another table of some kind.

        - The stored process should be on the stored process server (not workspace server) so that it can run more quickly. The workspace server would spawn a new SAS session before running the program.
        - Stored process macros should be turned off, so that %stpbegin and %stpend are not included in the stored process code.
        - In the stored process it is possible to write text directly to the browser by writing to the fileref _webout. So the basic purpose of the stored process is to read data into a data step and then write it to _webout in JSON format.
    - Sample stored process. The following stored process can be used to produce JSON data based on a SAS dataset. It supports some optional parameters as follows:
        - Dset … dataset to use
        - Whr … where clause to use with dataset
        - Start … observation number to begin from
        - dropVars … list of variables to drop from dataset
        - keepVars … list of variables to keep from dataset
        - renameVars … list of pairs of variables for renaming, e.g. oldName=newName
        - sort … specification for variable to sort by, e.g. [{"property":"age","direction":"desc"}]
        - limit … number of records to show
        - callback … if specified then it produces JSON inside a callback function with this name. This is used when making JSONP calls which are used for cross-domain applications where the application runs in one domain but the data comes from another.

```
%let paging=1 ; %* we always want paging active ;

%macro gridgetdata;
  %* these defaults may be passed in, or else they default ;
  %global whr;
  %if %symexist(dset)=0 %then %global dset;
  %if %symexist(whr) eq 0 %then %let whr=1;
       %else %if %bquote(&whr) eq %then %let whr=1;
            %else %let whr=%sysfunc(strip(&whr));
  %if ^%symexist(start) %then %let start=0;
       %else %if &start lt 0 %then %let start=0;
  /* Apply a relevant 'where' clause on the input file, if necessary */
  data subset;
       set &dset (
            where=(%sysfunc(strip(&whr)))
            %if %symexist(dropVars) %then drop=&dropVars;
            %if %symexist(keepVars) %then keep=&keepVars;
            %if %symexist(renameVars) %then rename=(&renameVars);
       );
  run;
  /* If remote data sort specified, it will come in the following form:
       SORT=[{"property":"XXX","direction":"YYY"}]
       - where XXX is the name of a sort variable and YYY is sort
direction*/
  %if %symexist(sort) %then %do;
       %let sortVar=%scan(%bquote(&sort),2,%str(%,%"%:%[%{%}%]));
       %let sortDir=%scan(%bquote(&sort),4,%str(%,%"%:%[%{%}%]));
       %put sortVar=&sortVar;
       %put sortDir=&sortDir;
       /* Proc SORT adjustment: */
       %if %upcase(&sortDir) eq DESC %then %let sortDir=descending;
            %else %let sortDir=;
       /* Now ready to do the actual sorting: */
       proc sort data=subset out=subset;
            by &sortDir &sortVar;
       run;
  %end;
       %else %put No sorting order specified.;
  /* Only a relevant subset of information will be extracted from the
server at a given time. */
  data _null_ ;
       dsid=open("subset");
       nobs=attrn(dsid,'nobs');
       call symput('nobs',strip(put(nobs,8.)));
       *if &last gt nobs then call symput('last',nobs);
       dsid=close(dsid);
  run;
  /* Set the default macro variables. */
  %let first=%eval(&start+1);
  %if ^%symexist(limit) %then %let last=&nobs;
       %else %if &limit le 0 %then %let last=&nobs;
       %else %let last=%eval(&start+&limit);
  %put ;
  %put dset=&dset;
  %put whr=&whr;
  %put first=&first last=&last nobs=&nobs;
  %put ;
  data subset;
       dataLineId=_n_+&first-1 ;
    set subset (firstobs=&first obs=&last);
  run;
  /* Stream all requested information back to web browser. */
  data _null_ ;
       length char $ 256;
       file _webout;
       dsid=open("subset");
       nvars=attrn(dsid,'nvars');
       put
            %if %symexist(callback) %then "&callback.(";
            /* Define metadata to be used to create dynamic grids: */

                 "{""total"":&nobs," /
                 '"success":true,' /
                 '"rows":'
            "[" @
       ;
       do row=&first to &last;
            put
                 '{'
            ;
            rc=fetch(dsid) ;
            do col=1 to nvars;
                 vname=varname(dsid,col);
                 vlabel=varlabel(dsid,col);
                 vtype=vartype(dsid,col);
                 vfmt=varfmt(dsid,col);
                 put '"' vname +(-1) '":' @;
                 if vtype eq 'C' then do;
                      char=getvarc(dsid,col);
                      put '"' char +(-1) '"' @;
                      if col ne nvars then put ',' @;
                 end;
                      else do;
                           if vfmt eq 'DATE9.' then
num=put(getvarn(dsid,col),ddmmyy10.);
                                else num=getvarn(dsid,col);
                           put '"' num +(-1) '"' @;
                           if col ne nvars then put ',' @;
                      end;
            end;
            if row ne &last then put '},';
                 else put
```

```
                          '}]'
                       %if %symexist(callback) %then ');';
                       @
                    ;
           end;
           %if &nobs eq 0 %then
              put
                 ']'
                 '}' @
                 %if %symexist(callback) %then ');' @ ;
              ;
           ;
           dsid=close(dsid);
      run;
    %mend;
    %gridgetdata
```

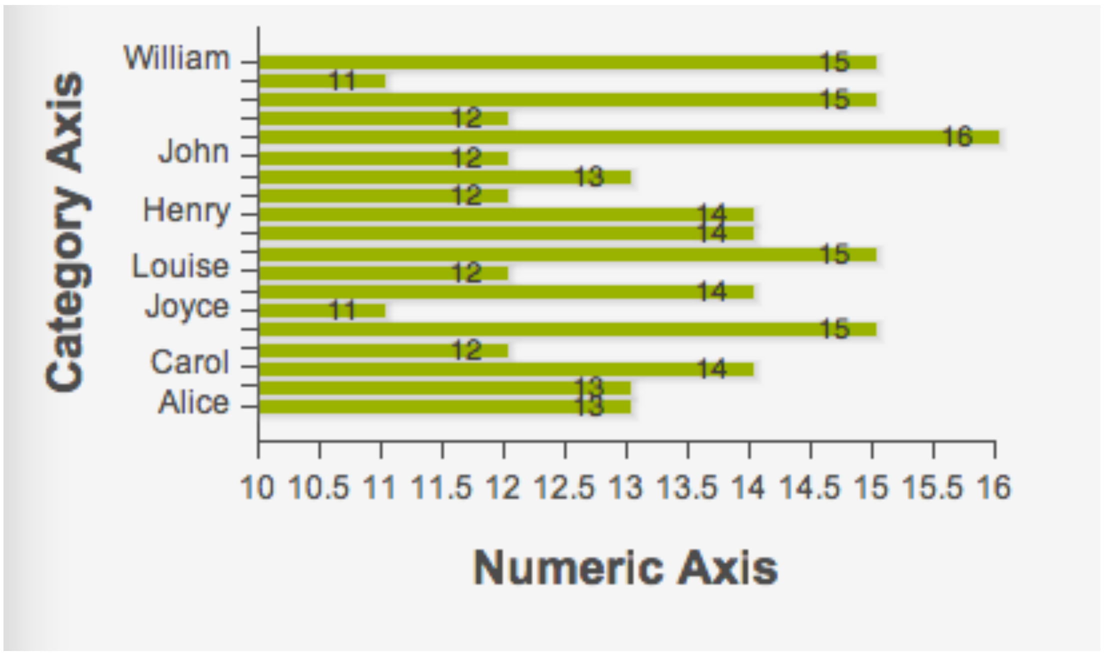
1) Point the store to the stored process. This makes the store use the stored process for its JSON data.
   - Select the URL property in the proxy for the store. It would have been set to something like /sample.json. It will need to be changed to the stored process call required to deliver the JSON, such as http://sasportal.intranet.group:10120/SASStoredProcess/do?&_program=/Shared+Data/Team/Group+Monitoring/cste/framework/stp/jsonGridProvider&dset=sashelp.class
   - The stored process can be run on its own and the output checked to make sure that it is valid JSON. You can copy the output and paste into www.jsonLint.com to check the validity of the JSON.
2) Enhance the grid with some more features.
   - You can group the rows in the grid by specifying the groupField property in the store. This will then enable the rows to be displayed in groups based on the variable selected in groupField.
   - You also need to go to the grid panel and find Grid Features. Add the grouping feature. This will use the group field defined above and enable grouping in the grid.
3) Handle very long tables. If you do nothing about this, then all the data will be loaded into memory when the store is loaded. This may adversely affect performance.
   - You could add a paging toolbar to the grid. This will then only load a page of data at a time and allow you to page through the data.
     - This is done by dragging a paging toolbar from the toolbox (on left) onto the grid.
4) Set the store for the toolbar to the one you are using for the Grid (should be done automatically)
   - In the store change the pageSize property to how many rows you want displayed in each page.
   - If you use paging then the JSON that is returned by the stored process needs to also return the following items:
     - "success": true (or false)
     - "total": 1234 (total number of records in the store)
     - "rows": [{}] (the row data goes here – the root property must be used to indicate where this item is)
   - If you use paging then 3 additional parameters are sent to the read stored process. The stored process that returns records should use these parameters so that it returns the right number of rows from the store that have been requested.
     - page – this is the page number that we want to display.
     - start – this is the record number we want the store to return first.
     - limit – this will be the pageSize. It is how many records we want returned by the store.
   - You could use the infinite scrolling feature which essentially treats the table as one big table, but only loads the rows that you need to see. This is easily done by simply setting the buffered property to true in the store associated with the table.
5) Once you have done this then you can do the same for other objects on your screen. You can make some general purpose stored processes which produce JSON for Ext JS objects. Then once you build each object you can just replace the JSON with the stored process to produce the needed JSON. It should merely be a matter of changing the parameters passed to it in order to produce the JSON needed.

ExtJS Graphs

If you wanted to add a graph, it is a very similar process to that above.

1. Rather than dragging a grid into your container, drag a Bar Chart.
2. Change the store to the one for your JSON data.
3. In Category Index, change fields to name.
4. In Numeric Axis, change fields to age.
5. In MyBarSeries, change
   - Xfield to name
   - Yfield to age
6. Edit label, and change the field to age which will make it show the age on the bar

Now we have a chart which is based on JSON data.



Generating Ext-JS code

In order to do something in extjs you will usually need some data and some directives telling extjs what you want to do with the data.

One simple way to provide data to EXT-js is by using arrays in javascript. For example you could use some lines like these ….

```
        // sample static data for the store
        var myData = [
            ['3m Co',                               71.72, 0.02,  0.03,
    '9/1 12:00am'],
            ['Alcoa Inc',                           29.01, 0.42,  1.47,
    '9/1 12:00am'],
            ['United Technologies Corporation',     63.26, 0.55,  0.88,
    '9/1 12:00am'],
            ['Verizon Communications',              35.57, 0.39,  1.11,
    '9/1 12:00am'],
            ['Wal-Mart Stores, Inc.',               45.45, 0.73,  1.63,
    '9/1 12:00am']
        ];

        // create the data store
        var store = new Ext.data.ArrayStore({
            fields: [
               {name: 'company'},
               {name: 'price',      type: 'float'},
               {name: 'change',     type: 'float'},
               {name: 'pctChange',  type: 'float'},
               {name: 'lastChange', type: 'date', dateFormat: 'n/j h:ia'}
            ]
        });
        // manually load local data
        store.loadData(myData);
```

However a better and more flexible way would be to use a data store. The simplest data store is a JSON data store. You may be wondering what a JSON data store is. JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.
JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

A simple JSON data store might look like this…

```
   {‘TotalCount’:’2’,
     'metaData':{'root':'rows', 'fields':["id","month","sales","wages"] },
     'rows':[{"id":"1", "month":"1", "sales": 600, "wages": 900},
          {"id": "2", "month":"2", "sales": 800, "wages": 1300}   ]      }
```

A JSON data store can be created by a SAS program, written to a file on the server and then used by the stored process that makes the grid. Or it could be created by a stored process so that it is streamed back to the browser. This enables ext JS objects to use a URL pointing to a stored process as a source of JSON data.
Once you understand the basic way that you can use an object like this, then it is quite easy to translate this into a stored process, so that SAS can do the same. This can be very simply done by adding the following code in front of the HTML code that you are using…

```
   *ProcessBody;
   /* generate html code */
   data html_code;
      infile datalines4 length=l;
      input #1 htmlline $varying400. l;
   datalines4;
```

And then add the next code after the HTML code you are using…

```
   ;;;;
   run;
   /* stream html code to browser */
   data _null_;
      file _webout;
      set html_code;
      put htmlline;
   run;
   *';*";*/;run;
```

This then gives you the SAS Stored Process code required to run a Stored Process, putting all your HTML code into a dataset, and then writing it to the web browser. You can see how this technique can easily then be extended to create parts of the HTML and Javascript code before it is written out.
An even better way to do this is to have another SAS stored process generate the JSON data store on the fly. A stored process to make a JSON data store could look like this.

```
   /* stream data to browser in JSON format */
   data _null_;
      file _webout;
      set sashelp.class end=last;
      if _n_ =1 then
         put "{ success:true, rows:["; /* return data retrieved was a
   success, rows is an array containing all the data */
      else
         put ","; /* seperator for each row in the data */
      put "{ name: '" name+(-1) "', sex: '" sex+(-1) "', age: '" age+(-1)
   "'}";
      if last then
        put "]}"; /* close array and JSON dataset */
   run;
   *';*";*/;run;
```

The extjs to use this would then look like this. It could replace the extjs for the sasdatasetStore variable that was shown in the previous complete version.

```
               var sasdatasetStore = new Ext.data.Store({ /* data store
   necessary to load data from the server */
                  id : 'sasdatasetStore' /* id is a unique id of the
   component so we can later reference it using
   Ext.getCmp('sasdatasetStore') */
                 ,url: 'do?_program=/CBA/dimitri_test2' /* stored process
   to retrieve data */
                 ,reader: new Ext.data.JsonReader({ /* reader to process
   the json data */
                     root: 'rows', /* The property in the JSON data which
   contains an Array of record objects */
                     id: 'name' /* name of the value which is used as id */
                  }, ['name', 'sex', 'age']) /* names of the values */
               });
```

You could also create a macro to create JSON data stores, so then it could be used in various stored processes as require. All of this means that by building 2 stored processes you can make a single URL that can be called and will deliver a nice looking grid with all the functionality that the extjs grid provides as standard, including:

- column sorting, on host or client
- excluding columns
- paging long tables

There are many other features that can be added to extjs grids by adding a little more complexity to your javascript. This can then achieve things like traffic lighting, putting special objects into table cells and so on. As an example, the following code shows how to implement traffic lighting with an extjs grid.

```
       function change(val) {
           if (val > 0) {
               return '<span style="color:green;">' + val + '</span>';
           } else if (val < 0) {
               return '<span style="color:red;">' + val + '</span>';
           }
           return val;}

       /**
        * Custom function used for column renderer
        * @param {Object} val
        */
       function pctChange(val) {
           if (val > 0) {
               return '<span style="color:green;">' + val + '%</span>';
           } else if (val < 0) {
               return '<span style="color:red;">' + val + '%</span>';
           }
           return val;}
```

The best place to find useful examples of Extjs is in the samples section of their web site, which contains many samples along with full source code. http://www.sencha.com/products/extjs/examples/

Making use of ext-js from Stored processes

One of the most useful things that I have discovered in the last few years is how to really use JavaScript objects from stored processes. By learning how to make SAS drive these web technologies is very powerful. Traditionally with SAS 8 and beyond we have been able to quite simply have SAS generate HTML to provide output for the web. Through the graphic device drivers available we have been able to add some interactivity very simply by taking advantage of the ActiveX and Java device drivers. These allow us to make a range of graphs which then provide some very good interactive functionality. This is great but has some problems. ActiveX is only supported on Windows platforms, which can be limiting. There is also a potential problem in that data can be edited using the ActiveX driver, and so output can be changed by the end user. There are a range of features available in each, which can be great for users but which you may not want to make available to users.
If users wanted to go beyond interactivity provided through the HTML ODS tag sets and ActiveX/Java drivers, then they would need to do some work and have a far greater understanding of further technologies in order to achieve this. A range of techniques can be used to do this including the following:

- Interfacing to another tool (such as EXCEL) and creating your output there via DDE, OLE, VBA or other techniques
- Creating a custom tagset, perhaps modified from an existing one, in order to implement further functionality
- Passing raw markup to your output in order to add functionality (e.g. passing raw RTF to MS Word)
- Creating custom Javascript within HTML to implement any required functionality
- Creating HTML to make use of features like the HTML 5 Canvas element.

I have found a more effective method is to make use of Ext JS, which is a Javascript framework that provides a large number of objects that can be used to build an application. There are many javascript frameworks, and so people will have their favourites, but I chose extjs by surveying the most popular ones and then looking at the application building functionality that each provided at that time. extjs provided far more than most others, and at the time of writing I think it still does. Additionally it has an Architect tool available that will write most of the code for you making it very easy to build powerful applications. And those applications will work on mobile devices too.

Using JavaScript frameworks

There are many JavaScript frameworks (or libraries) available on the internet, mostly which are free. They provide a collection of pre-written JavaScript controls which allow us to build web applications much quicker than we could otherwise do so. We also need much less knowledge to make use of these frameworks than we would to write the functionality from scratch. Common frameworks are: Prototype, script.aculo.us, jQuery, Ext and Dojo Toolkit. There are even libraries available from Microsoft and Yahoo!. You can Google these and find out all about them. You will find that each offer a similar but different range of controls to do all manner of things. For those readers who are familiar with SAS/AF you can think of these libraries as adding SAS/AF functionality for web development.
The one that we have most recently chosen to use is called Ext-js, and you can find it at http://sencha.com/. This framework has many features available that make building applications much easier. One of the key things that I have used is a JavaScript layout. It lets me design a screen where I can have different parts of the screen used for different things. In the screen shot to the left you can see how we have two areas on the left, including the top one in which items can be expanded and collapsed. On the right we have tabs, and within tabs we have other tabs with various content.
All of this kind of layout can be used with your SAS application. We have a SAS macro that builds the overall layout with various elements we want defined. Then we populate various parts of the framework by calling stored processes within them. For instance, the area in the screen shot about called “north” could contain the output of a stored process. That might let you make some selections and submit them, which could then run a stored process in the area called “west” that would display the HTML generated. This makes for a very powerful and flexible layout which can produce all kinds of applications.

Risks

When an innovative new solutions is proposed at a company good managers will usually ask what the risks involved are. For instance, if you press ahead with building an application that uses Ext-js, Javascript, Flash, HTML and SAS then what can go wrong? Well, based on my experience using this technology at four companies I would suggest the following:

- Browser lacks functionality or has bugs. IE6 is still the standard for many companies although it is no longer supported by Microsoft and has many bugs and vulnerabilities. Supporting this has proved to be a real headache at several of my clients.
- Flash unavailable to users and they cant install it. This means you can’t use flash objects which were supported in older versions of ExtJS.
- Browser might not support HTML 5 which is generated by the newer version of ExtJS for creating graphics.
- Lacking proper web development tools such as Javascript debuggers and IDE.
- Costs involved with various tools. For some large companies, it doesn't matter how expensive a tool is but merely that there is a cost.
- Introducing new tools/products to a controlled environment. In some companies there is a long and involved process involved in getting a new product accepted. Sometimes there are also support implications such as increased costs.
- Tools such as extjs which are free if you make your code public. Some companies have a problem with making their code available to others, although they like the idea of having their software for free.

Safer Alternatives

If creating web apps using HTML & javScript 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 the Ext JS framework allows us to use stored processes with powerful objects which have been pre-built. Using these techniques I have built some powerful applications for some major organisations. There is much more I could say, but I have been constrained by the amount that I can cover in a single paper/presentation. Be on the lookout for my upcoming Stored Process book which will cover all these things in much more detail. And drop me an email if you want me to send you the code from this paper to help in your own web application building.

Recommended Reading

See lots of useful examples of ExtJS code here … http://www.sencha.com/products/extjs/examples/
Download ExtJS here … http://www.sencha.com/products/extjs/download/
Lots of fabulous examples using SAS Graph to build dashboards and all kinds of things … http://www.robslink.com/SAS/Home.htm
Validate your JSON code here … http://jsonlint.com
Validate your XML code here … http://www.xmlvalidation.com

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