

Paper 031-2009

## Building Interactive Web Applications using Stored Processes

Philip Mason, Wood Street Consultants, Wallingford, Oxfordshire, U.K.

### ABSTRACT

Ten years ago, I was building applications using SAS/AF using a client/server architecture. Since the coming of SAS 9 and the spread of the web I have seen the tools arrive that let me build the same functionality that was available in SAS/AF but using stored processes used through a web browser. This paper describes what I have learned during 4 years of building Rich Internet Applications with SAS 9 stored processes.

### INTRODUCTION

In this paper I will outline some of the most useful things that I have learnt while I have been building applications with SAS 9.1. Prior to this I had built SAS applications using SAS/AF, which was very powerful and flexible. Moving to SAS 9.1 and with the advent of Stored Processes and the Stored Process Web Application it has opened up the possibility to easily create applications that run through a web browser.

### BUILDING A WEB APPLICATION

In this paper this is what I mean by an “Interactive Web Application”:

*An application that is used through a web browser which lets the user make choices, runs SAS code to produce results and delivers them back via the web browser.*

For a long time SAS/AF (since version 6 or earlier) was the SAS product that most people would use to produce applications. It is still used by a lot of people and it had many advantages including:

- Interactive development environment, making development quick and easy
- Very powerful and flexible functionality

SAS then delivered another product with version 7/8 which enabled applications to be developed for the web – SAS/Intnet. This was a CGI based tool, which enabled us to interact with SAS from a web page and return results to a web browser.

With SAS 9 came the new metadata based architecture, including the Stored Process. SAS delivered a Stored Process Web Application with this release which meant that we could run stored processes from a web page and return results to it. In many ways this was very similar to SAS/Intnet, but it was designed to be much more flexible and powerful. By using this web application and linking together a bunch of stored processes we can produce an Interactive Web Application. This is what we will focus on in this paper.

#### **One way to build a web app – server side includes**

One way to produce a web based application is to produce a stored process which takes a kind of pseudo code in. This pseudo code would define what a screen would appear as and what it is composed of.

For example, we could have a selection box that would let us choose one option and was filled by getting values for a variable from a dataset, then this could all be specified in a piece of pseudo code.

Example

```
%select_box(dataset, variable, choice, 1)
```

The stored process would then call the macro which carries out the directive in and construct HTML to produce that component. This means that to produce a page you just need to write a list of pseudo code directives. By building up a library of macros to interpret different pseudo code functions, you can extend the functionality a great deal.

There is a book by Don Henderson called "Building Web Applications with SAS/Intrnet" that describes how to do this in great detail. Highly recommended!

### **Another way to build an application**

This is the method I have been using recently. It makes use of the Stored Process Web Application to run stored processes through the web browser. One key technique is that when a stored process needs to prompt a user for input then it is written so that it can be run in two parts. The first part will build a web page which prompts the user for the required input and then calls the same stored process, passing in the selections that the user made. When the stored process is called the second time it can detect that it has been called with various parameters passed in, and it then runs to produce the output that is required. This basic idea was described in a SAS usage note and then developed some more by myself.<sup>1</sup>

### *Example pseudo code*

```
If expected parameters are passed in then
  Produce report
Else
  Produce web page
```

One key part of this technique is that a stored process needs to determine how to call itself. This can be done by constructing a URL from automatic macro variables that are provided to the stored process. The following code is taken from a stored process and it constructs a URL which will call that stored process, as well as passing some extra parameters to it. It makes use of the automatic macro variables: `_program`, `_srvname`, `_srvport` and `_url`. It also passes two parameters in: `graph` and `x`, whose values are `&type` and `&x`.

```
%let rev=%sysfunc(reverse(%superq(_program))) ;
%let index=%sysfunc(index(%superq(rev),/)) ;
%let rev2=%substr(%superq(rev),&index) ;
%let firstPart=%sysfunc(reverse(&rev2)) ;
%let html=http://&_srvname.&_srvport.&_url.?_program=&_program.
%nrstr(&graph)=&type.%nrstr(&x)=&x ;
```

Once we have this URL built we can take it and put into an HTML form, which can then be used to prompt the user for various parameters and pass them to the web app using the URL. The URL can be put into the action field in the HTML form. When the submit button of the form is pressed then the action field is used to start building the actual URL. Other fields on the form are added to the end of the URL. These are then available to the stored process as macro variables.

### **HTML, RTF or PDF?**

Another thing to consider when building your application is what you want your output to be produced as. You will most likely use a web browser for interacting with the user, but then reports that are produced may be required in HTML, RTF, PDF, EXCEL or other formats. Fortunately the web app makes this easy. You merely specify a value for `_odsdest` prior to calling the `stpbegin` macro, and it will then set the ODS options appropriately for the type of output you require. Since parameters passed in via the URL to the web app appear as macro variables, you can therefore pass a parameter on the URL called `_odsdest`, which will then set your output appropriately. That means that you can prompt the user for the type of output in your HTML and then pass that value through.

### **Want to do something tricky?**

If you are building a nice interactive web page in HTML then it is likely you will be wanting to take control over exactly what HTML is produced. If you don't, then you can get a lovely HTML page produced for you by using the wizard in Enterprise Guide 4.1. You can define a stored process there along with various parameters and values they can have. EG then produces a web page to prompt you for those parameters and then call the stored process. Very easy. However should you want to do your own web page, there are other techniques.

You can use a web development IDE (Interactive Development Environment) to produce your web pages. This is reasonably easy, but reduces flexibility a little. You will need to code in to the page the URL to call your stored process for instance (and I should say I am not an expert web developer and no doubt there are 50 ways of automating this that I am not aware of). But once your URL is encoded you can use the full power of the IDE to build all the other bits and pieces of your page (or pages). I would recommend Aptana Studio for this, which is free and powerful.

---

<sup>1</sup> Usage Note 19475: Example: Creating dynamic parameter values for a stored process

The method I mostly use is to have SAS produce my web page for me. 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 can't 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 `%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.

#### ***A collection of useful macros***

Any SAS application should have a nice collection of useful macros. If you are producing a web application then you will find yourself producing some macros which not only deal with the SAS environment, but also interact with HTML and JavaScript code. For example, the following macro will write a piece of JavaScript code to the HTML file being generated. That JavaScript will place a message into the status area at the bottom of the web page. Using this macro you can update the user on things that are happening in the background – for example “now generating graph 27 of 50”.

```
%macro message_js(text) ;
  %if %symexist(_odsdest) %then
    %if %upcase(&_odsdest)=RTF or
        %upcase(&_odsdest)=PDF %then
      %return ;

  data _null_ ;
    file _webout ;
    put '<script type="text/JavaScript">' ;
    put "window.status = ""&text"" ;" ;
    put '</script>' ;
  run ;
%mend message_js ;
```

I have many other macros for use with web development which do things such as:

- Produce selection lists of various kinds based on variables in a dataset
- Produce messages of various types in HTML code
- Write JavaScript functions to HTML
- Read and write values of HTML cookies
- Produce pop-up dialogs
- And many more

I also have developed a range of macros which produce nice HTML reports. These could be used in other places, but each of them have been enhanced to be highly interactive by supporting tool tips, drill down and special links to other functionality. For example, some of these include:

- Heatmap which shows a graduated shading of colors combined with traffic lighting for exceeded limits
- Listing which has bars that indicate magnitude of values, along with colors that show traffic lighting

- Gantt charts, which use an innovative technique using Proc Gchart

I like to share code, so if you are interested in any of these then please contact me and I can send you code.

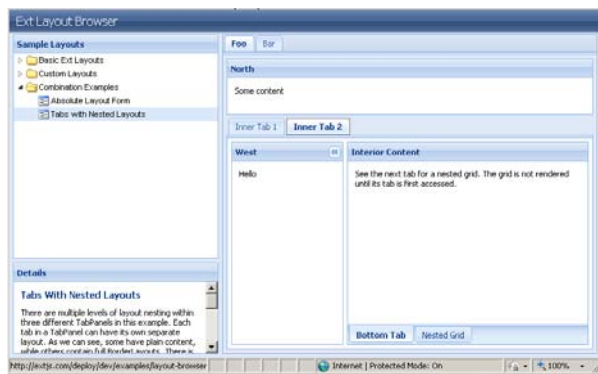
## STORED PROCESS WEB APPLICATION

In SAS 9 we have the SAS Stored Process Web Application, which enables Stored Processes to be run from a web browser and then will stream the results back to the browser. This is the single most useful new facility that SAS have provided in the last 5 years – 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.

## 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://extjs.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.

### Building interactivity using JavaScript

#### 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%2Ffoundation%2F&env.%2Fbis%2Fmedmon%2Fb_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 – `window.status="This is a message"`. There was a macro that used this on a prior page.

### 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.

## USING HTML

### 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 favorites 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 to use as a default */
  sep=%str( ), /* optional one character separator */
  quote=0, /* 1=quote values, 0=dont quote values */
  partstmt=0 /* 1=make part of where statement, 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.

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;
```

## 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 PHIL 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 3 other lines of code you can get a lot more power out of a stored process.

### *\*ProcessBody;*

This comment should be placed at the start of a stored process since it will initiate input parameter processing, if there are any input parameters – otherwise it does nothing. Whatever input parameters are passed to the stored process, including any defaults, are inserted in the code at this point when it runs. So if you pass a value in for a parameter called MONTH as FEB, then it is just like having the statement “%let MONTH=FEB” in your code at that point. So you will get a global macro variable defined for every stored process parameter. If you don't include “\*ProcessBody;” then values for parameters will be available.

### *%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 3 things, %stpbegin is the most complex to understand since it can make use of about 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.

### **Macro variables used with %stpbegin**

Some of the following macro variables will be populated by the web application and you can look at the value to use it in your stored process (e.g. \_metauser). Other values can be set by you prior to %stpbegin being called, and then the stpbegin macro will make use of the values you set.

**\_ACTION** – an action for Web application to take (form, execute, properties, background, strip, index, data)

**FORM** - displays custom input form if one exists.

**EXECUTE** - executes the stored process.

**PROPERTIES** - displays the property page, which enables you to set input parameters and execution options and to execute the stored process. This is really useful and flexible when you want to run an unfamiliar stored process

**BACKGROUND** - executes the stored process in the background. Useful if your stored process runs for a long time, especially since browsers will usually timeout after about 3-5 minutes and if your stored process runs longer then you can lose track of it.

**STRIP** - removes null parameters, used in combination with EXECUTE and BACKGROUND.

**INDEX** - displays a tree of all stored processes. This is very useful if you just want to browse all the stored processes that are defined and then select which one you want to run.

**DATA** - displays a summary of general stored process data.

You can combine parameters as follows, e.g.

**\_ACTION=FORM,PROPERTIES** ... displays a custom input form if one exists, otherwise displays the property page.

**\_ACTION=FORM,EXECUTE** ... displays a custom input form if one exists, otherwise executes the stored process.

**\_DEBUG** – debugging flags. These have a range of possible values:

**Log** – shows the SAS log after the stored process runs



*Time* – shows the real time taken by the stored process at the end

You can combine several `_debug` flags with commas like this: “`_debug=log,time`”. You can also use SAS/Intrnet style numbers to specify these flags. My favorite is using “`_debug=2179`”. This is a decimal converted from a binary, in which I set bits for various `_debug` options I want.

**`_GOPT_DEVICE`** – set the `goption device` parameter. I usually use `sasemf` for this, although other popular choices are `java`, `activex` and `png`. One nice thing about using `sasemf` on UNIX is that true type fonts are more easily used from it.

**`_GOPT_HSIZE`** – set the `goption hsize` parameter. Useful if you want to specify the horizontal graph size precisely.

**`_GOPT_VSIZE`** – set the `goption vsize` parameter. Useful for specifying the vertical graph size.

**`_GOPT_XPIXELS`** – set the `goption xpixels` parameter. I usually query my browser to work out the width, allow for any other things taking up space on the screen, and then set the width appropriately. I need to adjust this when I change destinations though, since producing a graph for an RTF document is best done by customizing its size for the page.

**`_GOPT_YPIXELS`** – set the `goption ypixels` parameter. In addition to the comments for `_GOPT_XPIXELS`, I use this parameter when I have lots of items I want to put on my y-axis. I can make the graph very long and then display them all clearly as the user scrolls the HTML page down.

**`_GOPTIONS`** – set some SAS/Graph options.

**`_METAPERSON`** – shows the real name of the person associated with the `userid` in the metadata. Will be *unknown* if there is no association.

**`_METAUSER`** – shows `userid` that was used to connect to metadata server.

**`_ODSDEST`** – Specifies the ODS destination (default is HTML). Can also be one of (CSV, CSVALL, TAGSETS.CSVBYLINE, HTML, LATEX, NONE (which produces no ODS output), PDF, PS, RTF, SASREPORT, WML, XML or any other tagset destination.

**`_ODSOPTIONS`** – specifies options that are added to the end of the ODS statement. One key use of this is if you want titles and/or footnotes to be included in graphs, since `NOGTITLE` and `NOGFOOTNOTE` are default options. You can override them by specifying `GTITLE` and/or `GFOOTNOTE` in `_ODSOPTIONS`.

**`_ODSSTYLE`** – sets ODS `STYLE=` option.

**`_ODSSTYLESHEET`** - Sets the ODS `STYLEHEET=` option.

**`_PROGRAM`** - Name of the stored process. This is really useful if you want to build up a link from the current stored process to itself, since this gives you the name of the stored process.

**`_RESULT`** – Specifies what kind of final result is produced by the stored process. It can be one of the following:

*STATUS* – produces no output to client.

*STREAM* - output is streamed to client through the `_WEBOUT` fileref.

*STREAMFRAGMENT* – just like *stream* but kind of a cut down version. This is not documented but I find it really useful for producing HTML when I want to have more control over my HTML.

*PACKAGE\_TO\_ARCHIVE* - package is published to an archive file.

*PACKAGE\_TO\_REQUESTER* - package is returned to the client. The package can also be published to an archive file in this case.

*PACKAGE\_TO\_WEBDAV* - package is published to a WebDAV server.

*PACKAGE\_TO\_EMAIL* - package published to one or more e-mail addresses.

*PACKAGE\_TO\_QUEUE* - package published to a message queue.

*PACKAGE\_TO\_SUBSCRIBERS* - package published to a subscriber channel.

**`_SRVNAME`** – the host name of the server. This is very useful when you want to write stored processes that can build URLs for links

**`_SRVPORT`** – the port number on which this request was received. Also useful in building up a URL for links.

**`_STPERROR`** - Global error variable, 0 if everything worked properly, otherwise non-zero.

**\_URL** - Specifies the URL of the Web server middle tier used to access the stored process. Also useful in building up a URL to use with links.

**\_USERNAME** - the user name obtained from Web client authentication.

## EXTENDING THE USEFULNESS OF METADATA/SECURITY

In SAS 9 we have metadata available – metadata is data about data. SAS Metadata tells us about the SAS 9 Business Intelligence environment such as Stored Processes, Servers, Users, etc. For example when a user runs a stored process through the SAS Stored Process Web Application they specify a userid and password. The userid is looked up in the metadata and a check done that they are allowed to run that stored process. If they can, then metadata is used to retrieve the SAS code to run and to determine what server to run it on. If the server hasn't started running, then metadata is used to start the server. When results are produced metadata determines what is done with them. This is only part of what goes on automatically behind the scenes when running a stored process in a web browser.

There is much more that you can do with your metadata though. SAS provides some tools for interacting with metadata. There are several functions that can be used from the data step. There is a procedure called Proc Metadata in which you specify XML to tell it what you want to do with the metadata. There is also Management Console that allows you to interactively use the Metadata.

A colleague of mine<sup>2</sup> developed an incredibly useful macro which we use to determine what groups a user is a member of in the SAS metadata. We can feed it a userid and get back a list of the groups. That means that in management console we can define groups for particular functionality (e.g. the 123 group get to see data from Study 123). We then assign users as members of groups. Then in a stored process we can look up the user to see what functionality we should make available to them. So when a stored process runs that produces a list of studies from which they can choose data, we will only show those studies which they are supposed to see the data.

You can use the following macro as follows: "%meta\_person(person=Philip Mason)". You then get a collection of other macros that will list things defined in the metadata for that person. So you can work out what things they should be able to see and then only show those things.

```
%macro meta_person(person=);

  ** input parameters ****;
  ** person      : name of person to get memberships for as stored in      **;
  **          metadata ****;
  ****;

  * create empty global macro variables;
  %global _meta_person_uri _meta_person_name _meta_group_names _meta_group_uris
  _meta_group_num;
  %let _meta_person_uri=;
  %let _meta_person_name=;
  %let _meta_group_num=;
  %let _meta_group_names=;
  %let _meta_group_uris=;

  %if %index(%superq(_srvname),mysassystem)>0 %then
    %do ;
      %let server=%scan(&_srvname,1,.)%scan(&_srvname,2,.) ;
      %let pwd={sas001}eDJ00TJ2MGFz;
    %end ;

  * set metadata server options;
  options metaserver=&server"
    metaport=8561
    metaprotocol=bridge
    metauser="sas"
    metapass="&pwd"
    metarepository="Foundation";
```

<sup>2</sup> Edwin Van Stein of Astellas, Netherlands.

```

%* get URI for person in metadata;
data _null_;
  length id type $256;
  type='Person';
  id='';
  rc=metadata_resolve("omsobj:Person?@Name='&person.'", type, id);
  %* create macro variables if person is found;
  if rc gt 0 then do;
    call symput('_meta_person_uri',strip(id));
    call symput('_meta_person_name',strip("&person"));
  end;
run;

%* if URI for person is found get group memberships;
%if &_meta_person_uri ne %then %do;

  %* set temporary filename;
  filename out temp ;

  %* get group memberships for person;
  proc metadata server="&server"
    port=8561
    userid="sas"
    password="&pwd"
    in="<GetMetadata><Metadata><Person
id='&_meta_person_uri' /></Metadata><NS>SAS</NS><Flags>1</Flags><Options/></GetMetad
ata>"
    out=out ;
  run;

  %* get groups person is member of from output of proc metadata;
  data work._groups (keep=id name);
    infile out lrecl=32767;
    input;
    %* read in proc metadata output;
    line=_infile_;
    %* get number of groups person is member of;
    numgroups=count(line,'<IdentityGroup ','i');
    call symput('_meta_group_num',strip(put(numgroups,8.)));
    do i=1 to numgroups;
      %* go to beginning of identity group;
      line=substr(line,index(line,'<IdentityGroup '));
      %* the URI of the group is the second word;
      id=scan(line,2,'');
      %* the name of the group is the fourth word;
      name=scan(line,4,'');
      output;
      %* go to end of identity group;
      line=substr(line,index(line,'/>')+2);
    end;
  run;

  %* create macro variables with URIs and names of groups;
  proc sql noprint;
    select name , id into :_meta_group_names separated by '#' ,
:_meta_group_uris separated by '#'
    from work._groups;
  quit;

  %* clean up;
  proc datasets lib = work nolist;
    delete _groups;
  run;

```

```

quit;

%end;

%* if no person found write message to log;
%else %do;
    %put INFO: &person was not found in the metadata;
%end;

%* write values for create macro variables to log;
%put INFO: _meta_person_name = &_meta_person_name;
%put INFO: _meta_person_uri = &_meta_person_uri;
%put INFO: _meta_group_num = &_meta_group_num;
%put INFO: _meta_group_names = &_meta_group_names;
%put INFO: _meta_group_uris = &_meta_group_uris;

%mend meta_person;

```

## POTENTIAL PROBLEMS & SOLUTIONS

### TABLES WITH FIXED HEADERS

One major problem I have found when developing web applications with SAS is that the tables produced by ODS HTML don't fix the row and column headers. That means that if you have a very wide table and scroll to the right, then you no longer can see the row headers on the left. If your table is very long then as you scroll down you will lose the column headers and so can't tell what the columns are for. SAS have provided a few solutions to this problem which can be found in the SAS usage notes.<sup>3</sup>

The first solution is to use a tagset provided by SAS which actually fixes the row and column headers. This tagset also does some other nice things, like allowing you to sort based on values in columns. I have found that this works well on small tables but once they get large it slows down a great deal and becomes almost unusable.

The next solution is to effectively insert some JavaScript into the HTML table which fixes the position of the row headers and column headers relative to the web page. This works, but the headers appear to jerk about as you scroll – which is not very pretty.

I haven't found an ideal solution yet. I plan to experiment with some tables provided with JavaScript frameworks, and perhaps to look at embedding an EXCEL table into a web page.

### PROBLEMS YOU MIGHT HIT DURING DEVELOPMENT

#### **Handling different browsers**

Different browsers behave in different ways and you can detect which browser you have and make allowances in your code. During development of my latest web application I tried to support a range of browsers, but discovered odd little differences between them in various areas. Something that worked in one browser would sometimes not work in another one – and to fix it in the other one would require some special work around coding. In the end I decided to only support the company standard web browser, which was Internet Explorer 6, and that made life somewhat easier.

You can see the browser compatibility tables<sup>4</sup> which show which browsers support the official web standards. It is interesting to note that no browser supports all the standards.

#### **Large graphs fail to be produced**

We ran into some problems when trying to produce very large and complex graphs. Even though we had set the **memsize** option in SAS to allow 512meg of memory, we found that some graphs were running out of memory at a much lower level. I had decided to take advantage of the web page delivery by making some graphs very detailed and long, so that you would scroll down the web page to see them all. Anyway, after some investigation I discovered the **maxmemqry** SAS option, which sets the maximum amount of memory that a single SAS procedure can use. This had to be increased so I could produce my extra large and long graphs – then all was well.

<sup>3</sup> Usage Note 24066: In ODS HTML, how can I freeze my table headers while I scroll the page?

<sup>4</sup> <http://www.quirksmode.org/dom/compatibility.html>

**Axis labels become too small**

Early in development we were looking at a single clinical study, which didn't produce particularly large graphs. But eventually we developed graphs that were comparing many treatments between many different studies. This meant that I ideally want to fit a lot more onto my graph. Producing a horizontal bar chart with 60 items being compared resulted in tiny almost unreadable axis labels. But then it occurred to me that I was using a web browser in which I had a limited width, but an unlimited length. As long as people didn't have to scroll side-to-side, they would probably be happy scrolling down to see more details on a graph. So I set the graph ypixels to be very large and the x pixels to match the width of my screen (to allow for different resolutions). This let me make long graphs. To get the size of my screen (which may differ from user to user) I just used a simple bit of JavaScript which I then delivered to my stored process as a macro variable.

I put the following macro in reports where I am creating graphs. A lot of my graphs were not appearing properly on low res screens, so I adjust the resolution for that. Also, when I change the `_odsdest` to send the graph to RTF then I adjust the resolution so that the graph comes out making full use of the page in RTF.

```
%macro handle_low_res ;
  /* handle low resolution screen ;
  %if &_gopt_xpixels<450 %then
    goptions xpixels=650 ypixels=425 ;
  %else
    %if &_gopt_xpixels<800 %then
      goptions xpixels=875 ypixels=850 ;
  ;
  /* A4 sized graphs for RTF destination ;
  /*A4 in Pixels - 300 dpi (print) = 2480 X 3508 pixels */
  /*A4 in Pixels - 200 dpi (default for ODS RTF) = 8.3 x 11.7 inches = 1660 X 2340
  pixels */
  /* PA4 is largest paper format that can fit on both A4 and US Letter without
  resizing
  it is 210mm x 280mm, 8.26771654in x 11.023622, 1653pixels x 2204pixels (RTF),
  ratio of 3:4 - which matches traditional TV screen ;
  %if %upcase(&_odsdest)=RTF %then
    %do ;
      goptions xpixels=1900 ypixels=1425 ;
      options orientation=landscape papersize=("280mm","210mm") ;
    %end ;
  %mend handle_low_res ;
```

**Making graphs look good on UNIX, while still being produced fast**

When developing an application for the web you have to consider various factors. Response time is one important one. People ideally want to see pretty and detailed graphics, but also want their graphics to arrive quickly. The slowest part of producing a graph and delivering it to a web browser is usually the network speed – and so the size of the graph is a major factor. So it becomes a balancing act of weighing up the size of the graphic against the speed it takes to download.

So, select a device driver that produces high quality, yet small file sizes.

Another factor with selection of device driver is the support for fonts. Some device drivers support true type fonts, which usually look by far the best and lift the whole appearance of your graph. Other device drivers don't support true type fonts and so you are reduced to using SAS software fonts, which often don't produce very nice looking graphs.

So, select a device driver that can use true type fonts.

**Getting True-type fonts to work in graphs**

When I initially began to produce graphs on UNIX (Solaris and AIX) I began to run into problems with the use of fonts. If you use a SAS software font such as swiss, then that will work in all graphic devices (PNG, JPEG, BMP, TIFF, etc.). However if you want your graphs to look really good and use true-type fonts then you find that they just don't work automatically in most devices. In fact they are not supported at all in most devices, but you can get them to work in certain ones. For instance you can define particular fonts to work with the PNG device, and I have done this but it is a little annoying. However if you use the SASEMF driver then any true-type fonts available on UNIX can be automatically used. At least this is true after they are imported into SAS. You must use **Proc fontreg** to import whichever fonts you want to use:

```
proc fontreg ;
  fontfile "/sas/general/fonts/arial.ttf" ;
```

```
run;
```

You can also just import all the fonts in a directory and then use any of them – which is a bit less efficient if you have lots of fonts.

```
proc fontreg ;  
  fontpath "/sas/general/fonts" ;  
run;
```

## CONCLUSION

This paper attempts to give some insight into a very large topic – web development using SAS. It is not only a large topic, but a fast evolving one. That means that by the time you read this, it may not be totally up to date. I hope it has given you some insight into what can be done and how. To the uninitiated it may seem that SAS is not the best tool to develop a web application. In fact I have run into experienced java developers who have refused to believe this until the evidence was collected and beat over their heads – but we had java developers come to love SAS.

The key to the power of SAS as a web application enabler is the Stored Process web application. Start with that and you can't go wrong.

## REFERENCES

There are some very useful sections in the documentation on the SAS web site.

Creating Stored Processes - [http://support.sas.com/rnd/itech/doc9/dev\\_guide/stprocess/program.html](http://support.sas.com/rnd/itech/doc9/dev_guide/stprocess/program.html)

Building a Web Application - [http://support.sas.com/rnd/itech/doc9/dev\\_guide/stprocess/webapp.html](http://support.sas.com/rnd/itech/doc9/dev_guide/stprocess/webapp.html)

## RECOMMENDED READING

Visual Quickstart Guide: CSS, DHTML & AJAX, by Cranford & Teague

Visual Quickstart Guide: HTML, XHTML & CSS, by Castro

## CONTACT INFORMATION

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

Name:	Philip Mason
Enterprise:	Wood Street Consultants Limited
Address:	16 Wood Street
City, State ZIP:	Wallingford, Oxfordshire, OX10 0AY, England
E-mail:	phil@woodstreet.org.uk
Web:	www.woodstreet.org.uk

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

Other brand and product names are trademarks of their respective companies.