

## Paper 016-2011

## Unleashing the power behind Stored Processes

Ian Amaranayake, Amadeus Software Ltd., Witney, U.K.

### ABSTRACT

Managing SAS code within an organization can be challenging, especially when trying to control access to most recent versions of SAS® programs. Stored processes provide us with an enterprise wide solution to this problem, enabling us to distribute the latest versions of programs for use in a variety of SAS 9 applications. Stored processes are easy enough to use out of the box but knowing how to customize their appearance and behavior requires a broader understanding of how they operate.

This paper will explore how to customize stored processes with SAS 9.2 and take control of their inputs and outputs. The ability to parse information directly via a URL, customize the prompts user interface and control their operation within desktop and web applications will be discussed. This paper will be useful for beginner through to advanced SAS users who have some prior experience of using the SAS 9 applications.

### INTRODUCTION

Stored processes provide us with the capability to define a SAS program which is managed and distributed via a server but which can be executed from a variety of client applications. It consists of two components:

- A code component which can be authored in the same way as a regular SAS program;
- A metadata component which is registered in a metadata repository using applications like the SAS Management Console or Enterprise Guide.

The code component is typically bounded by two macro calls; **%STPBEGIN** and **%STPEND** which are used not only to top and tail a definition, but more importantly to open and close the ODS destination which the output will be sent to and deliver the results to the requesting client. A number of reserved macro variables can be used in conjunction with these calls when output is streamed or sent to a temporary or permanent package file. A selection of these options will be explored in this paper.

A **\*ProcessBody** comment is also used where the workspace server is responsible for execution. This comment is not required if the stored process server is used, as it enables the use of prompts. This functionality is enabled by default for the Stored Process server.

**Prompts**, from a SAS programmer's perspective, are simply macro variables which are assigned values and resolved on execution, in a similar fashion to regular macro variable substitution. **%GLOBAL** statements are used to initialize both prompts and reserved macro variables and thus help eliminate warning messages that can get generated if prompts are not used when the stored process executes. The capabilities associated with prompts in SAS 9.2 will be explored in depth within this paper.

The following code sample depicts the typical opening and closing statements within a stored process.

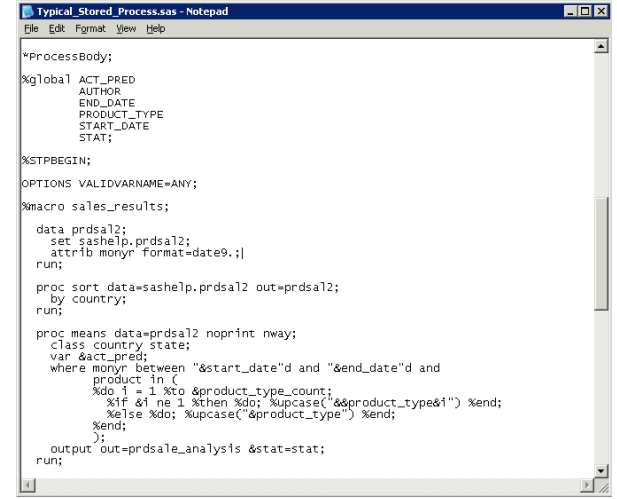
```
*ProcessBody;  
%GLOBAL VAR1 VAR2 VAR3;  
%STPBEGIN;  
...  
...  
%STPEND;
```

The introduction of stored processes in the SAS application toolkit provides a number of important benefits when compared with regular SAS programs:

- The capability to deploy and manage code at an enterprise level which in turns helps to:
  - Prevent the proliferation and continued use of old and bespoke versions of code;
  - Control read, write and execute access across metadata groups and users.
- The capability to execute code within a variety of SAS applications.

- The collection of input values through a graphical user interface.
- Control over how output is returned.

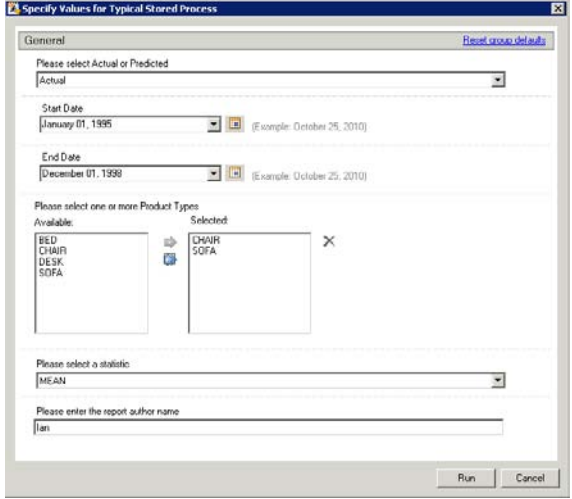
For the purposes of this paper we are going to consider two programs which produce tabular and graphical reports and explore how their inputs and outputs can be customized. The first stored process 'Sales Report' calculates statistics on either actual or predicted sales and provides sub-group analysis based on country, state and product type within a specific date range. The dataset used is the SASHELP.PRDSAL2 table.



```

*ProcessBody;
%global ACT_PRED
AUTHOR
END_DATE
PRODUCT_TYPE
START_DATE
STAT;
%STPBEGIN;
OPTIONS VALIDVARNAME=ANY;
%macro sales_results;
  data prdsal2;
    set sasHELP.prdsal2;
    attrb moryr format=date9.;
  run;
  proc sort data=sasHELP.prdsal2 out=prdsal2;
    by country;
  run;
  proc means data=prdsal2 noprint nway;
    class country state;
    var &act_pred;
    where moryr between "&start_date"d and "&end_date"d and
    product in (
      %do i = 1 %to &product_type_count;
        %if &i ne 1 %then %do; %upcase("&product_type&i") %end;
        %else %do; %upcase("&product_type") %end;
      %end;
    );
    output out=prdsale_analysis &stat=stat;
  run;

```



Specify Values for Typical Stored Process

General

Please select Actual or Predicted: **Actual**

Start Date: **January 01, 1995** (Example: October 25, 2010)

End Date: **December 01, 1999** (Example: October 25, 2010)

Please select one or more Product Types

Available: BED, CHAIR, DESK, SOFA

Selected: CHAIR, SOFA

Please select a statistic: **MEAN**

Please enter the report author name: **Ian**

Run Cancel

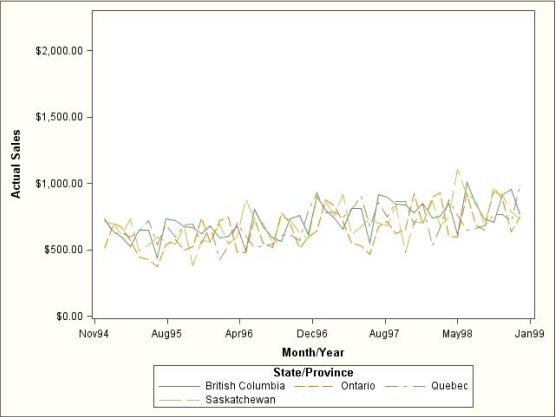
**Summary of Sales figures for the period 01Jan1995 - 01Dec1998**  
Product Types covered: CHAIR SOFA

Country	State	Sales
Canada	British Columbia	\$724.28
	Ontario	\$654.12
	Quebec	\$695.36
	Saskatchewan	\$706.18
Canada		\$2,779.95
Mexico	Baja California Norte	\$443.34
	Campeche	\$452.82
	Michoacan	\$473.93
	Nuevo Leon	\$456.75
Mexico		\$1,826.84
U.S.A.	California	\$1,552.65
	Colorado	\$932.63
	Florida	\$894.32
	Illinois	\$72.00
	New York	\$1,464.82
	North Carolina	\$959.08
	Texas	\$1,458.15
	Washington	\$930.57
U.S.A.		\$8,264.21

Report produced by: Ian on 25OCT2010

**Summary of Sales figures for the period 01Jan1995 - 01Dec1998**  
Product Types covered: CHAIR SOFA

**Country=Canada**



Report produced by: Ian on 25OCT2010

**Table 1. Stored Process Code, Prompts and Output Table and Graph**

The second stored process 'Orders Report' produces a summary of products ordered by product category and sub-category over a specific time period. As with the 'Sales Report' stored process, a number of inputs have been defined to influence the range of data summarized.

## ENHANCED OUTPUT PRESENTATION

The first topic for consideration is the presentation of output. Stored process output, in much the same way as regular SAS output, can be customized to make use of ODS styles. Styles can be defined when working within SAS clients like Enterprise Guide and the Add-In to Microsoft Office, but it is sometimes desirable to control and fix the

presentation of results to promote consistency. The problem here is that unlike regular SAS output, there is no visible ODS open and close statement to attach a style definition to.

There are however a series of reserved macro variables specific to stored processes, which can be used to achieve this and many other purposes. The `_ODSSTYLE` macro variable influences the style of output created, regardless of the destination and default assumed by the requesting client. The `_ODSDEST` macro variable can be used to define the default destination of output. All of these macro variables can be used provided they occur before the `%STPBEGIN` macro call. This is because the `%STPBEGIN` macro relies on these reserved macro variables to define its operation and so must be set up prior to the call in order to take effect. If not defined, applications like Enterprise Guide and the SAS Stored Process web application will initialize variables with defaults. Some of these macro variables are defined such that they apply to all stored processes executing on the server.

The example code below has been taken from the 'Sales Report' stored process where definitions for the ODS destination, style and stylesheet have been included. The reason for initializing the stylesheet with a null value is to prevent clients like Enterprise Guide from using the default stylesheet, which in the example below, would override the Magnify style from being applied to tabular report contents.

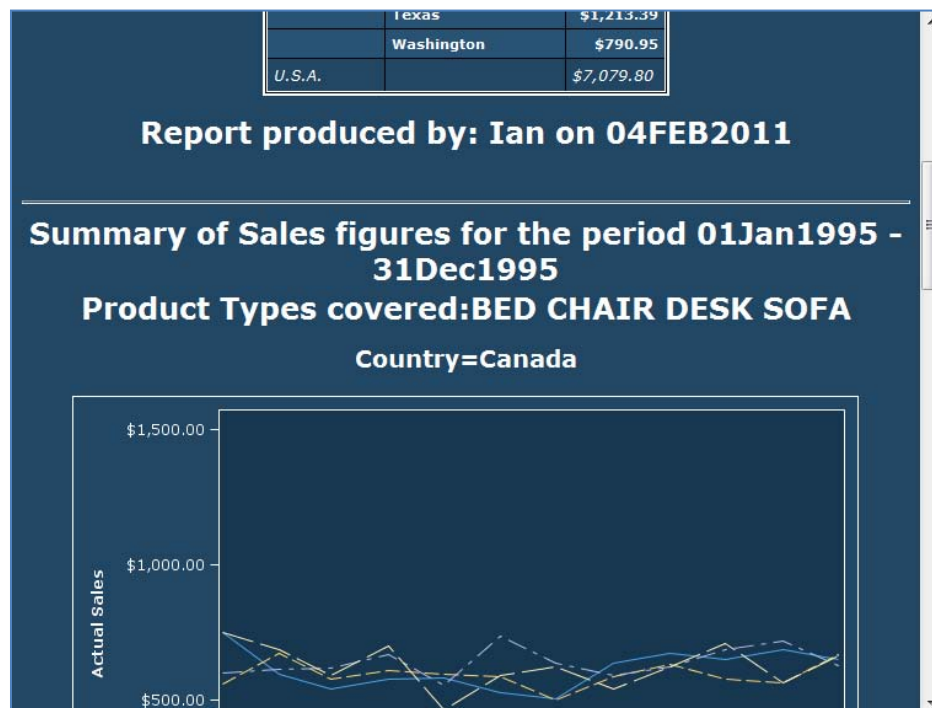
```
*ProcessBody;

%global _ODSDEST _ODSSTYLE _ODSSTYLESHEET
      ACT_PRED AUTHOR END_DATE PRODUCT_TYPE START_DATE STAT;

%let _ODSDEST=HTML;
%let _ODSSTYLE=Magnify;
%let _ODSSTYLESHEET=;

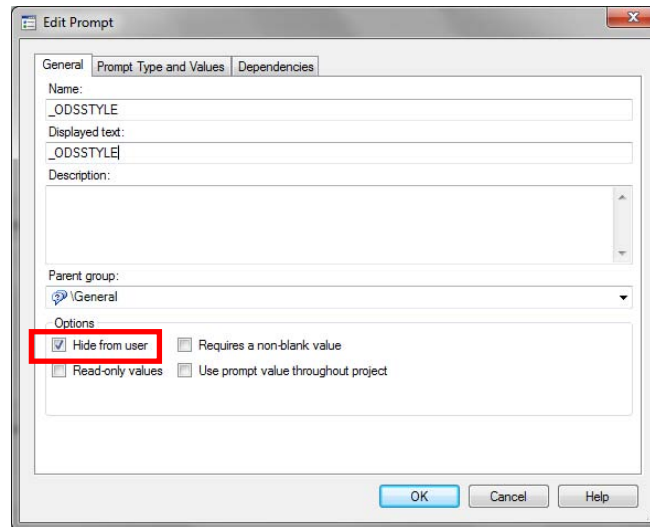
%STPBEGIN;
...
...
%STPEND;
```

This technique can be used in any application capable of running stored processes and will automatically override any styles or user preferences defined. This can be useful when the developer wishes to enforce a corporate style within a specific type of report.



**Figure 2. Stored Process Output With Fixed ODS Characteristics**

A slightly unintentional by-product of this approach is that all reserved macro variables declared in code will subsequently be displayed as prompts within the user interface. It is therefore considered good practice to also declare reserved macro variables as hidden prompts.



**Figure 2. Defining a Hidden Prompt**

Special care should also be taken when using Enterprise Guide to manage stored process definitions, as it can have a tendency to strip out reserved macro variable definitions whenever changes are saved. Using the SAS Display Manager System or an appropriate text editor provides an appropriate solution to this problem.

There are a variety of reserved macro variables that influence the opening ODS statement generated by %STPBEGIN and consequently the ODS output created. As well as applying SAS supplied styles it is also possible to use custom styles that have been saved within a SAS Item Store or saved in a CSS file. This can be achieved by using the \_ODSSTYLESHEET macro variable.

In the following example, a corporate stylesheet is used to render stored process output in a consistent theme. The TITLE attribute defines the report title in the output files metadata and the GTITLE and GFOOTNOTE options force graphical titles and footnotes to be included as part of the graph. The remaining variables are concerned specifically with SAS/GRAPH output, setting the format to PNG and the graphics output area size for each graph.

```
%let _ODSSTYLESHEET=(URL="file:///D:/Development/Corporate.css");
%let _ODSOPTIONS=ATTRIBUTES=("TITLE"="Stored Process Paper"
                             "CODEBASE"="http://www2.sas.com/codebase/graph/v92/sasgraph.exe#version=9,2")
                             GTITLE GFOOTNOTE;
%let _GOPT_DEVICE=PNG;
%let _GOPT_HSIZE=6 in;
%let _GOPT_VSIZE=4 in;
```

## DYNAMIC AND CASCADING PROMPTS

The remainder of this paper will focus on how we can influence the collection of stored process inputs.

SAS 9.2 saw the introduction of a new prompt framework which provides significant advantages over previous versions of SAS. New prompt types were introduced including date and time ranges which enable boundaries to be defined and libraries, data sources and variables which enable the user to browse for content. It also provided the capability to share prompts between stored processes (and in fact any component capable of working with prompts) thus promoting the use of shared resources.

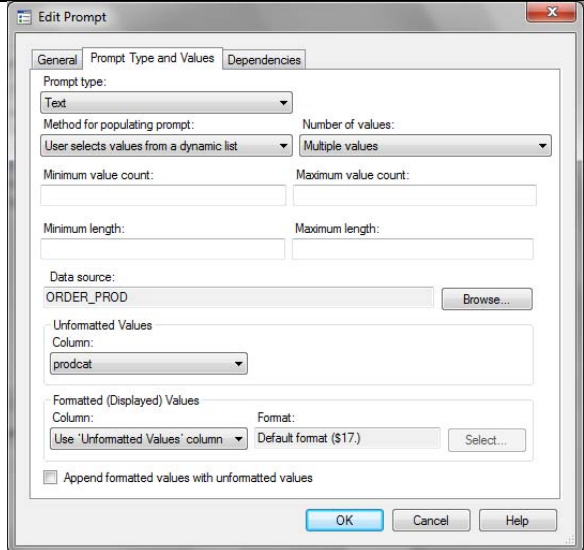
The inclusion of Dynamic and Cascading prompts enable us to address the legacy issue with having to manually load/update valid values within a prompt, every time the source data changes. These prompts do still exist and are now referred to as 'Static' prompts.

Dynamic prompts in contrast enable us to establish a link between the prompt and any valid SAS data source registered in metadata. This ensures prompt values always reflect the most recent version of the data which they are linked to without any need for manual intervention.

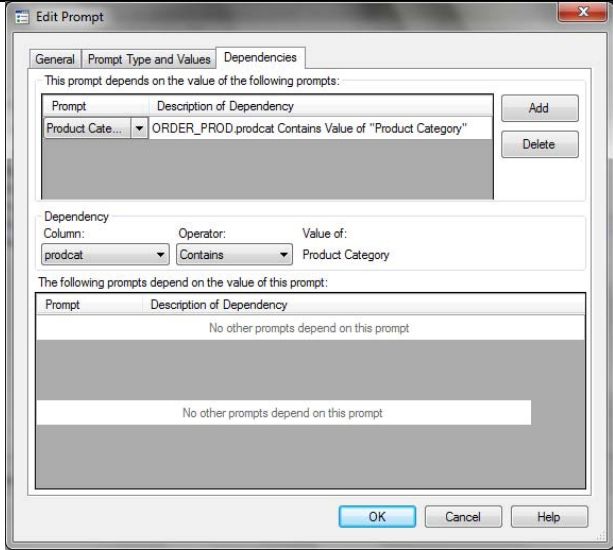
Cascading prompts go one step further by permitting dependencies to be defined between dynamic prompts. This allows us to selectively display information in a prompt based on responses that have already been provided in their dependent counterparts.

If we consider the 'Sales Report' stored process, it is possible to improve the prompts within here by defining the 'Product Category' and 'Product Sub-category' as dynamic prompts so that any updates to our inventory are automatically displayed without the need for manual processing. In order for this to work, the data source which contains these values must be registered in metadata against any server as long as it is not the LOCAL server.

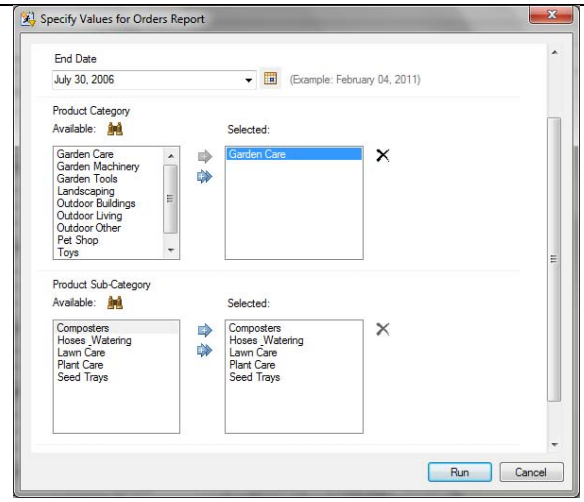
Product Sub-category in this example has also been defined as a Cascading prompt as the selections of Product Category should limit the available selections displayed for the Product Sub-category. We can do this as the two variables have a hierarchical relationship.



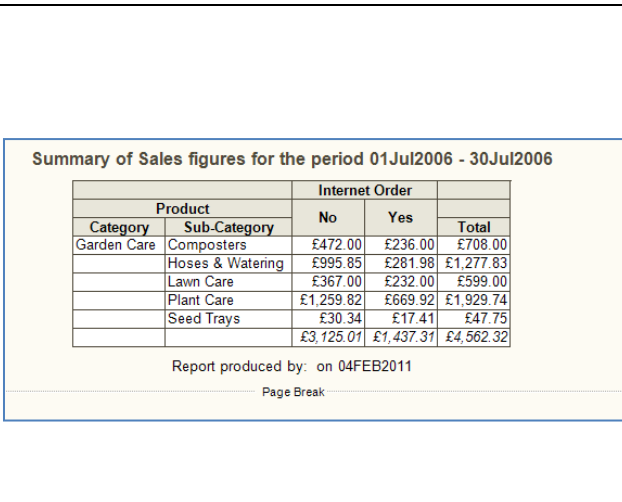
**Product Category Prompt Properties**



**Product Sub-category Prompt Dependency Properties**



**Prompts Screen**

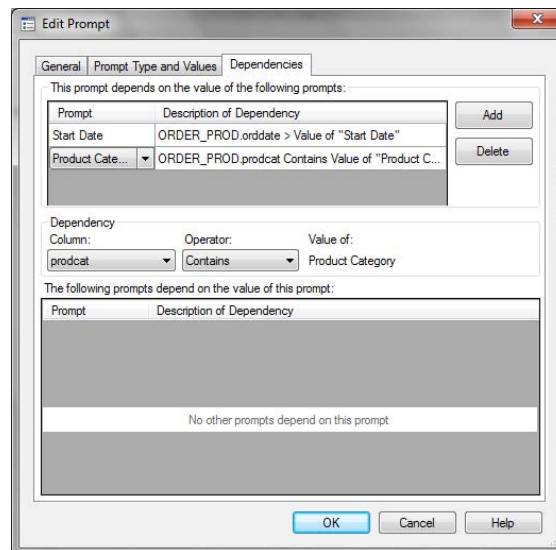


**Output Report**

**Table 2. Defining Dynamic and Cascading Prompts**

In the previous example, the Product Sub-categories displayed are the ones relevant to the Product Categories selected. The way in which these prompts are used in the stored process code is in the definition of a compound WHERE clause which filters the records being analyzed and displayed in the report.

The advantage of this technique is that it is scalable to larger more complex processes. We could cascade prompts throughout the prompts screen such that each question leads on from the last. It is also possible to define multiple dependencies that need to be fulfilled prior to activating a prompt for selection. The screenshot below shows the 'End Date' prompt being changed to depend on values where the 'Order Date' variable is greater than the selected 'Start Date' but also dependent on a valid 'Product Category' being selected.



**Figure 3. Multiple Dependencies for Dynamic Prompts**

There are still some restrictions which are not straightforward to overcome. If a prompt is to be defined as cascading it must be defined with the 'Method for populating prompt' set to 'User selects values from a dynamic list' which results in valid values being selected from a drop down or a list box. This poses a problem for our stored process example where the 'End Date' prompt should be presented as a calendar control, consistent with the 'Start Date'. The current version of the prompt framework does not support this functionality as of yet. This issue could potentially be overcome by defining an entirely bespoke user interface.

## RUNNING STORED PROCESSES IN SAS WEB APPLICATIONS

One of the strengths of stored processes is that they can be executed from the SAS 9 web applications such as the Stored Process Web application or the Information Delivery Portal, using nothing more than a compliant web browser and an active metadata connection. Within the Information Delivery Portal there are four options available to the developer for distributing stored process content.

1. Collection portlets can be used to display a list of stored processes for the user to select from. On selecting a link, the stored process is then run, displaying the prompt screen on a separate web page;
2. URL Display portlets can be used to display both the prompt screen and results;
3. From SAS 9.2 TS2M3, a SAS Stored Process portlet can be used to display both the prompt screen and results;
4. A new application can be created to display the stored process prompt screen and results.

In most cases, the results produced will be displayed in a new browser window by default.

The URL Display portlet method will be discussed in detail, but it should be noted that the techniques discussed are broadly applicable to all implementations. The SAS Stored Process portlet has been designed specifically to display stored process content, prompts and results, however the URL Display portlet has some additional functionality which



can prove useful.

The URL Display portlet can be used to directly reference the address of a stored process. Due care should be taken when entering the address, as copying the URL from, for example, a Collection portlet will result in the prompts page being embedded in the portlet itself (including the page banner). The address can be best retrieved from the SAS Stored Process web application and will typically adopt the following structure.

```
http://<web-server:port-number>/SASStoredProcess/do?_action=form,properties,execute,nobanner,newwindow
&_program<stored-process-address>
```

What does this HTTP request mean? Well the 'do?' keyword is an instruction to execute the stored process via the application identified by 'SASStoredProcess' which is the SAS Stored Process web application. An example of an actual address is shown below. The '%2F' and '+' characters are encoded path and separator characters generated by the web server to represent forward slashes and spaces respectively.

```
http://sasweb.amadeus.net:8080/SASStoredProcess/do?_action=form,properties,execute,nobanner,newwindow
&_program=%2FShared+Data%2FDevelopment%2FStored+Processes%2FTypical+Stored+Process
```

Entering this address in the URL Display portlet results in the stored process being run every time the portal page is refreshed. If prompts are defined then the input screen will be displayed each time. It is important when defining this to make sure the 'Show full HTML content in an inline frame' is selected along with an appropriate 'Content Height' (otherwise multiple scroll bars may be displayed).

There are two additional problems that occur at this point; the first of which being that in order to use the SAS Stored Process web application, the user must enter their logon credentials the first time they access the page. The second problem is that the results generated automatically open in a separate window.

Both of these properties can be changed by exploring the syntax of the URL. Both the `_PROGRAM` and `_ACTION` keywords are reserved macro variables which are used by the SAS web clients. They can be changed by editing the URL directly. Removing the 'newwindow' option solves our problem with a new browser window being opened, but to solve the login issue we need to use a new directive introduced in SAS 9.2 called **STPRun**. This directive has the advantage of passing the users credentials which were used to login to the Information Delivery Portal to the Stored Process Web application. It also supports the same reserved macro variables discussed earlier. Notice how the SAS Information Delivery Portal is now the responsible application in the following URL. The STPRun directive is case sensitive so due care should be taken when entering the Stored Processes location in metadata.

```
http://<web-server:port-number>/SASPortal/Director?_directive=STPRun&_action=form,properties,execute,nobanner
&_program=<stored-process-address>
```

It is also possible to pass values to existing macro variables and prompts in the stored process via the URL. This is beneficial in circumstances where the same stored process is to be re-used with either different default values or where no prompts are entered by the user but the URL itself is used to pass all of the values.

In the next example we set the ODS style to RTF and also initialize the default statistic to SUM. In order to pass values to reserved macro variables like `ODSSTYLE` which are handled by `%STPBEGIN`, it is necessary to add this to the stored process definition again as a hidden prompt to prevent it from being displayed.

The key part of the address is highlighted in the URL field below.

Figure 4. Editing the Stored Process URL

These changes result in the default statistic being changed and ultimately the new output being generated using the RTF style.

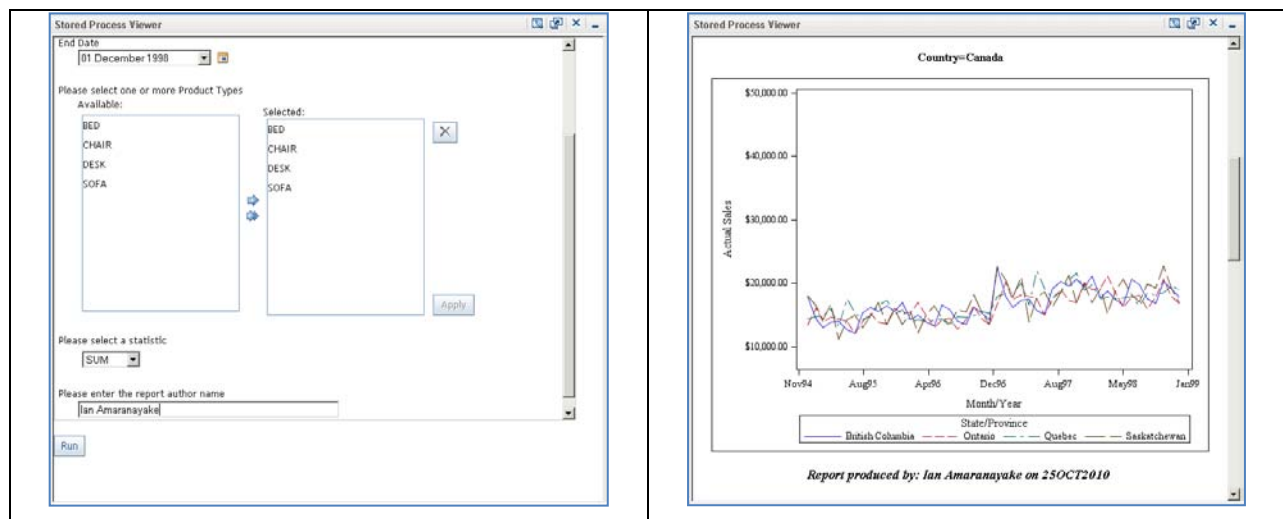


Table 3. Defining Reserved Macro Variables on the Stored Process URL

Finally, editing the `_ACTION` macro variable to remove the instruction to display the prompt dialog (represented by the `PROPERTIES` keyword) and any available user defined input form (the `FORM` keyword), we can now run the same stored process by passing all required values into our prompts and execute it immediately. Notice how all the prompts values are appended one after the other on the URL.

```
http://<web-server:port-number>/SASPortal/Director?_directive=STPRun&_action=execute,nobanner &_program=<stored-process-address>&_odsstyle=Medaow&stat=SUM&act_pred=ACTUAL&author=Ian&Product_type=BED
```

This results in the following output being produced directly on refreshing or accessing the portal page.



Stored Process Viewer

Summary of Sales figures for the period 01Jan1995 - 01Dec1998  
Product Types covered:BED

Country	State	Sales
Canada	British Columbia	\$197,706.60
	Ontario	\$194,493.60
	Quebec	\$204,737.40
	Saskatchewan	\$193,568.40
Canada		\$790,506.00
Mexico	Baja California Norte	\$130,696.20
	Campeche	\$132,886.80
	Michoacan	\$130,464.00
	Nuevo Leon	\$141,361.20
Mexico		\$535,408.20
U.S.A.	California	\$440,389.08
	Colorado	\$265,046.40
	Florida	\$248,428.80
	Illinois	\$101,707.06
	New York	\$445,593.60
	North Carolina	\$260,938.80
	Texas	\$383,364.90
	Washington	\$238,869.00
U.S.A.		\$2384337.64

Report produced by: lan on 25OCT2010

Figure 5. Defining All Reserved Macro Variables via the URL

## ENHANCED PROMPTING

As sophisticated as the prompting facility is, there are times when the default input controls provided are not the most suitable choices for collecting information from the user. Unfortunately, the developer has very little say in which controls are actually used, the decision being made internally by SAS based on the type of prompt defined and the method for populating values. For example, defining a prompt of type 'Date' with a method of populating values set to 'User enters values' automatically displays a calendar control. As we saw earlier, changing this prompt to read a dynamic list precluded the use of this control for our second date parameter.

Considering our current 'Sales Report' prompt screen, we may wish to change some of the single-select drop-downs to radio buttons and the list boxes to check boxes to provide a more intuitive User Interface (UI). We may also wish to influence the prompt screens appearance by adding cosmetic changes like styles and company branding.

All of this can be achieved by developing a bespoke UI for which there are a number of approaches to consider:

- Custom Java Server Pages (JSP) which make use of HTML forms to submit inputs to a stored process;
- Custom HTML form which submits inputs to a stored process.

In either case, the SAS Stored Process web application will be responsible for displaying the custom input form and submitting the results to the 'Sales Report' stored process. The **STPRunParameters** directive is used which provides similar control to using **STPRun**, but with the added benefit of displaying our new input form via an implicit **\_ACTION** value of 'FORM, PROPERTIES, EXECUTE'.

We start by considering the JSP method which defines a static HTML form. If the reader is not familiar with Java or JavaScript, a useful start point is to review the JSP's which are used for the Sample Stored Processes. These samples are included as part of the SAS Web Infrastructure Platform and will typically be listed under the web application servers root directory within the 'input' folder.

In our 'Sales Report' implementation we are using a JBoss application server. We can find the sample JSP's under 'SASServer1\tmp\deploy\tmp47569sas.storedprocess9.2.ear-contents\sas.storedprocess-exp.war\input\Products\SAS\_Intelligence\_Platform\Samples'. Note the entire contents under 'input' get generated on (re-)starting the web application server or redeploying the web applications.

The JSP we have created for the 'Sales Report' aims to simplify the user interface for brevity, but demonstrates the use of a number of HTML form controls:

- The analysis variable to be used is now defined using a series of radio buttons;

- A country filter has been added using a single selection drop down box;
- The products to include are now collected using check boxes;
- Additional report titles are collected using text input fields;
- A button with type 'Submit'.

The JSP which collects this information must define three things:

- The METHOD attribute on the FORM tag has a default value of GET (not shown below). This ensures on selecting submit, all form field values are transferred to the SAS Stored Process web application, in a similar fashion to macro variable parsing described earlier in this paper. All form field values are written to the \_URL reserved macro variable.
- The ACTION attribute on the FORM tag has been set to 'URI'. This defines what will happen when the user clicks on the submit button. This references the URL which the JSP will ultimately run under.
- The \_PROGRAM reserved macro variable, represented in the form as a hidden field. This references our revised 'Sales Report' stored process which will perform the analysis and build the reports.

A sample of the JSP is displayed below:

```
<%
out.flush();
String URI = ((String)request.getAttribute("SASStoredProcessURI"))!
=null?(String)request.getAttribute
("SASStoredProcessURI"):request.getContextPath()+"/do";
session.setAttribute("Banner_Title", "Sales Report Custom Format");
javax.servlet.RequestDispatcher dispatcher =
request.getRequestDispatcher("/jsp/banner.jsp?" +
request.getQueryString());
dispatcher.include(request,response);
%>

<div style="margin-top: 15px; margin-left: 15px; margin-right:
15px;"><!-- Main body content -->

<p>
This customised input form demonstrates how we can override the
default user interface presented by the stored process facility.
</p>

<hr />

<form action="<%=URI%>">
<input type="hidden" name="_program" value="/Shared Data/Stored
Processes/Sales_Report_Custom" />

<table cellpadding="3" cellspacing="5">
<tr>
<td valign="top">Select the variable to analyse</td>
<td valign="top">
<input type="RADIO" name="act_pred" value="ACTUAL" checked>
ACTUAL</input>
<input type="RADIO" name="act_pred" value="PREDICT">
PREDICT</input>
</td>
</tr>
</table>
```

**Figure 6. Custom Stored Process JSP**

In order to test the new input form we need to create a folder structure within JBoss's 'input' folder which is identical to the 'Sales Reports' metadata folder location. In order to make this an applicable JSP file path we also need to bear in mind that special characters like spaces will be converted to underscores (whether they occur in the file or folder names or both).

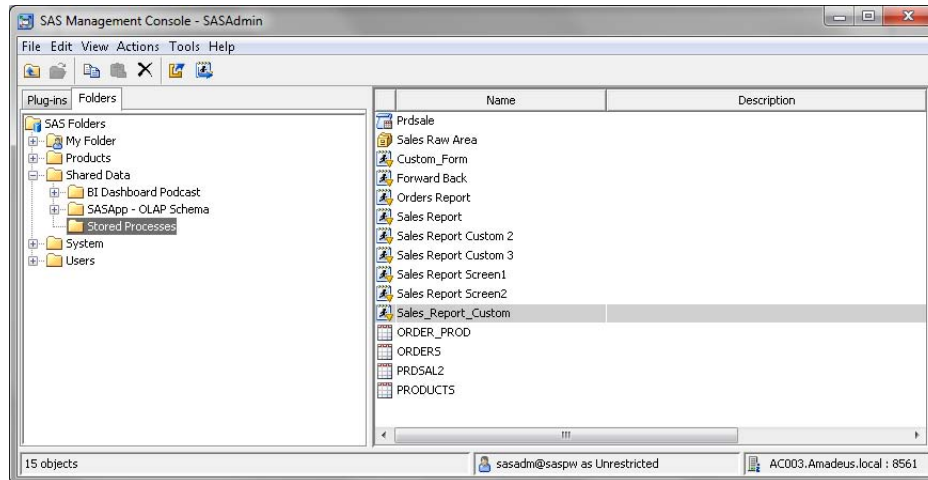


Figure 7. Location of Stored Process in Metadata

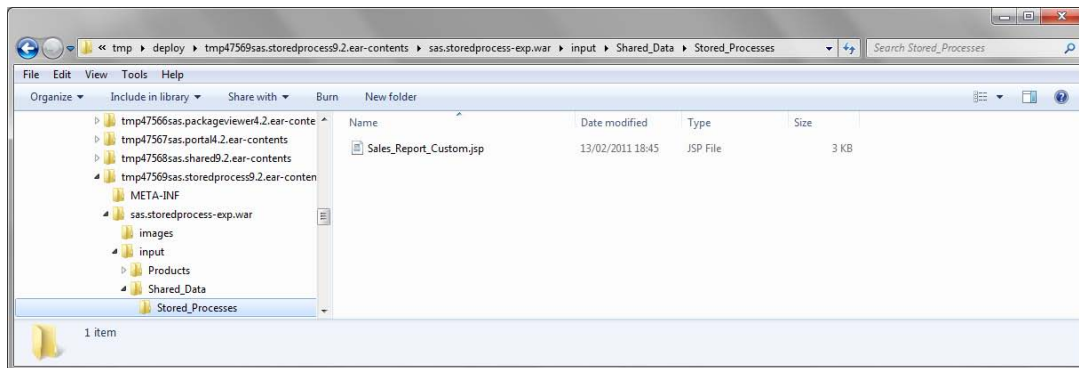


Figure 8. Location of Stored Process under JBoss application server

Saving the JSP within here now allows us to refer to the form using the STPRunParameters directive. This HTTP request identifies the location of the customized 'Sales Report' stored process in metadata as well as the JSP (i.e. under /input/Shared\_Data/Stored\_Processes).

```
http://<web-server:port-number>/SASPortal/Director?_directive=STPRunParameters&_program=
%2FShared+Data%2FStored+Processes%2FSales_Report_Custom
```

Accessing the page via a URL Display portlet shows us the complete user input form and on making selections and selecting to 'Run Report' we can see the resulting report.

Stored Process Viewer

This customised input form demonstrates how we can override the default user interface presented by the stored process facility.

Select the variable to analyse

☒ ACTUAL ☐ PREDICTED

Select the Country to filter on:

-- All --

Select one or more products:

☒ Bed

☒ Sofa

☐ Chair

☐ Desk

Additional Report Titles:

Produced by: Ian A

Amadeus Software Ltd.

Run Report

Summary of ACTUAL Sales figures for the period 01JAN1995 - 31DEC1995

Product Types covered:BED SOFA

Produced by: Ian A

Amadeus Software Ltd.

Country	State	Sales
Canada	British Columbia	\$638.23
	Ontario	\$601.08
	Quebec	\$657.25
	Saskatchewan	\$645.64
Canada		\$2,542.21
Mexico	Baja California Norte	\$402.98
	Campeche	\$391.28
	Michoacan	\$438.68
	Nuevo Leon	\$399.78
Mexico		\$1,632.72
U.S.A.	California	\$1,360.16
	Colorado	\$796.81
	Florida	\$803.85
	Illinois	\$65.68
	New York	\$1,310.18
	North Carolina	\$840.57
	Texas	\$1,275.67
	Washington	\$767.52
U.S.A.		\$7,220.43

Table 4. Customized JSP Input Form and Results

Once tested, the next stage is to deploy the JSP for production use. This will require redeploying the web application so that it is included as part of the SAS Stored Process web applications EAR file (enterprise archive file). For more information on how to do this, refer to the SAS 9.2 Intelligence Platform: Web Application Administration Guide.

It is quite likely that the person developing stored processes may not be in a position to redeploy web applications, in which case an alternative approach is required.

Custom HTML forms provide us with similar functionality to JSP's but without needing to re-deploy web applications. HTML forms can be defined statically or dynamically through a SAS program which could be the stored process itself!

The goal of the SAS program is to generate the required HTML tags, including the FORM definition with the ACTION and METHOD attributes, all of the form fields and the \_URL and \_PROGRAM macro variables in a similar fashion to the JSP. Quite how it achieves this is up to the developer. The use of INFILE and DATALINES statements can provide a quick way of generating a UI. Alternatively PUT statements can be used to test conditions and optionally create HTML tags as required. In both cases, the \_WEBOUT file stream should be used as a means of creating the input form in the current output destination. This can be achieved by using a FILE statement. It is important when doing this to make sure the %STPBEGIN and the %STPEND macros are **not** included within the stored process definition. If they are not removed this will result in the \_WEBOUT destination being locked for write access prior to the FILE statements execution which will cause the process to fail.

The following sample shows some of the key attributes which have been defined within the stored process. Note the METHOD attribute defaults to GET which populates the \_URL macro variable as in our JSP example with the contents of all form fields.

```

%let stpname=/shared Data/Stored Processes/Sales_Report_Custom;

data _null_;
  length infile $256;
  infile = resolve(_infile_);
  input;
  file _webout;
  put infile,
  datalines;
<HTML>
<BODY>
<H1>Sales Report Custom 3</H1>
This sample illustrates how to create stored processes.
<p>
This customised input form demonstrates how we can override the default user interface presented by
the stored process facility.
</p>
<br />
<FORM ACTION="&_URL">
<input type="hidden" name="_program" value="%stpname" />
<table cellpadding="3" cellspacing="5">
  <tr>
    <td valign="top">Select the variable to analyse</td>
    <td valign="top">
      <input type="RADIO" name="act_pred" value="ACTUAL" checked>ACTUAL</input>
      <input type="RADIO" name="act_pred" value="PREDICT">PREDICTED</input>
    </td>
  </tr>
  <tr>
    <td valign="top">Select the country to filter on:</td>
    <td valign="top">
      <select name="group_by">
        <option value="ALL" selected>-- All --</option>
        <option value="U. S. A.">U. S. A.</option>
        <option value="Canada">Canada</option>
        <option value="Mexico">Mexico</option>
      </select>
    </td>
  </tr>
</table>

```

**Figure 9. Definition of a Customized HTML Input Form**

From this point the STPRunParameters directive can be used in a similar fashion to before, this time to run the stored process which contains the definition of the HTML form. On submitting the form, user selections are then passed through to the 'Sales Report' stored process.

```

http://<web-server:port-number>/SASPortal/Director?_directive=STPRunParameters&_program=
%2FShared+Data%2FStored+Processes%2FSales+Report+HTML

```

This process of linking together stored processes is commonly referred to as 'chaining'.

## CHAINING STORED PROCESSES

It is possible to create even more flexible applications by linking together stored processes such that the definition of one is determined from another. This technique enables us to distribute processing and decision making across a number of programs and user interfaces, making it possible to develop complex systems. There are a number of scenarios where this can prove useful:

- The capability to create separate 'pages' means we can guide the user through prompts in a similar fashion to a wizard, rather than displaying all information on the same input form;
- The pages displayed can be made dependent on previous selections enabling us to branch functionality in a more effective way than using cascading prompts or groups. A hierarchy of stored process pages can be defined in this way with navigation links built in;

- A 'menu' of programs could be defined which, dependent on user selection, runs a series of tasks.

The main challenge we are faced with here is in ensuring relevant information is passed from one stored process to the next. This is an important consideration as each stored process is independent which means details established in one cannot be automatically accessed by another. There are however a number of methods for passing values between them:

- HTML form input fields
- Cookies
- Server-side sessions
- URL's

This section will explore the use of the first three techniques as the fourth has already been outlined in this paper (i.e. passing information directly via a URL). For this example we will consider our updated 'Sales Report' which currently collects four groups of data. This will be split onto two separate pages of input with navigation controls provided between each. Four stored processes will be chained to provide this functionality, the last of which is our updated version of 'Sales Report' program which produces the table and graphs.

As we wish to provide a means of navigating between both HTML input pages, the first two stored processes will be designed to display two separate forms with HTML tags explicitly output via PUT statements to the \_WEBOUT output stream. As with our previous bespoke web form approach, we must take care not to use %STPBEGIN or %STPEND, otherwise the \_WEBOUT destination will be locked for output. The third stored process will be used to navigate from the second page back to the first. This could in practice be extended to cover multiple pages from a single program if desired.

The first stored process is used to initialize all macro variables, which will capture user input in the global symbol table. This is an important first step as it ensures all fields are initialized to null. Default values could be defined at this point.

Both screens have hidden input fields defined. These are used as a means of passing values from one page to the next. The sample below shows two hidden form fields defined on page one which will be used to store values collected on page two should the user navigate back a page. Notice the slightly unorthodox use of single quotes around macro variable references as opposed to double quotes. This is required as the single quotes are understood by the INPUT tag but the double quotes around the PUT statement still allow us to resolve the macro variable.

```
put "<input type='hidden' name='product_type4' value='&product_type4' />";
put "<input type='hidden' name='title3' value='&title3' />";
```

When outputting the various form controls, care is taken to ensure any values already entered are automatically displayed.

```
put " <input type='RADIO' name='act_pred' value='ACTUAL'
%if &act_pred = ACTUAL %then %do; checked %end;
>ACTUAL</input>";
```

The \_PROGRAM parameter is also defined as a hidden field which references the second stored process. Selecting the 'Submit' button executes the second stored process which builds the second page of prompts. To the end user, this has the effect of navigating forward a page.

```
put "<input type='hidden' name='_program' value='/Shared Data/Stored Processes/Sales Report Screen2' />";
...
put "<input type='submit' class='button' value='Next Page' />";
```

In order to pass values forward, a hidden input field is defined on page two which collects the values of global macro variables populated on page one. The HTMLENCODE function is used here to encode any special characters so that they can be transferred appropriately within HTML. For example, the ampersand '&' character is typically encoded to '&amp;' when referenced via a URL. The encoding of special characters can help avoid potential problems with characters which have a reserved meaning in HTML. This is especially important for free text entry fields where



special characters may be entered.

```
act_pred= htmlencode("&act_pred");
put '<INPUT TYPE="hidden" NAME="act_pred" VALUE="' act_pred '">';
```

The second page has two 'Submit' buttons defined which represent two possible outcomes. The first is to submit all of the details entered on both forms and run the 'Sales Report'. The second is to navigate back to the first form making sure all current selections are saved. The use of hidden fields will automatically ensure the latter. In order to provide navigation back, both 'Submit' buttons call our third stored process which has the sole purpose of determining which button was selected in order to run the appropriate code.

```
%if %upcase(&run) = PREVIOUS PAGE %then
  %inc "Sales Report Screen1.sas";
%else %inc "Sales_Report_Custom.sas";
```

The net effect of these customizations is a sophisticated interface which retains user selections between pages.

Figure 10. Chained Stored Process Screen One

Figure 11. Chained Stored Process Screen Two

This technique has many advantages, not least through its ease of use and extensibility which allows the developer to very quickly develop large systems comprised of a network of stored processes. The main disadvantage of this technique lies in the visibility of HTML and its appropriateness as a vehicle for potentially sensitive information. By transferring user selections through HTML tags, the astute user will be able to see content being defined in hidden fields by viewing the HTML source. If security is of primary concern to the developer, alternative techniques may need to be considered although these short comings could still be circumvented through careful design of the input form selections themselves and the stored process code which ultimately processes them.

As far as viable alternatives go, cookies could be used in a similar way to hidden fields to transfer information between stored processes. Cookies offer the advantage of being more robust in the way they are passed but in general require additional configuration on the web application server, as well as a detailed knowledge of how to work with them in the first instance!

It is also possible to pass data through the stored process server session courtesy of a session key, which when passed to the client can then be used to retrieve data from the server. This eliminates the transfer of large amounts of data between the web server and the requesting client and can be created by using the stored process server functions like STPSRV\_SESSION. This topic is not explored further within this paper but the author would encourage the keen web developer to explore this further as a possible alternative to HTML forms.

## CONCLUSION

In conclusion, there are a variety of techniques that can be used to extend the default functionality of Stored Processes and take greater control over their inputs and outputs. Designing and creating customized input forms provide precise control over the collection of information and even allow us to cascade data to further stored processes using the 'chaining' method. The use of reserved macro variables and directives can be used to greatly enhance the delivery of output, enforce presentation standards and promote re-use of stored process code.

## REFERENCES

Cynthia L. Zender, SAS Institute Inc., Cary, NC "ODS Options and SAS Stored Processes", proceedings of the SAS Global Forum 2007 Conference.

Kevin Davidson and Minh Duong, University of Houston, Houston, TX "Using Dynamic and Cascading Prompts in SAS® Enterprise Guide®".

SAS® 9.2 Stored Processes: Developer's Guide.

## CONTACT INFORMATION

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

Ian Amaranayake  
Amadeus Software Ltd.  
Mulberry House, 9 Church Green  
Witney,  
Oxfordshire.  
OX28 4AZ  
U.K.  
Work Phone: +44 (0)1993 - 848010  
E-mail: [ian.amaranayake@amadeus.co.uk](mailto:ian.amaranayake@amadeus.co.uk)  
Web: [www.amadeus.co.uk](http://www.amadeus.co.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.