THE POWER TO KNOW®

SAS® 9.3 Stored Process Examples

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2013. *SAS® 9.3 Stored Process Examples.* Cary, NC: SAS Institute Inc.

SAS® 9.3 Stored Process Examples

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, April 2013

Contents

1

Introduction

1.1 The Essentials – What Are SAS Stored Processes?

A *SAS Stored Process* is a SAS program that is stored on a server and defined in metadata. That's it! A SAS Stored Process can be almost any SAS program, provided that you store it in a central location and include the necessary metadata. This stored process can then be called and executed by various client applications, without rewriting the program for each client.

1.2 What Can You Do with Stored Processes?

Stored processes are very versatile. They can be used for creating web reports, performing analytics, building web applications, delivering packages to clients or to the middle tier, and publishing results to channels or repositories. Stored processes can also access any SAS data source or external file and create new data sets, files, or other data targets that are supported by SAS.

Stored processes can be used with a variety of clients, including, but not limited to, the following:

- SAS Enterprise Guide
- SAS Stored Process Web Application
- SAS Add-In for Microsoft Office
- SAS Data Integration Studio
- JMP
- SAS BI Dashboard
- SAS Information Map Studio

- SAS Web Report Studio
- SAS Information Delivery Portal

1.3 What's in this Book?

This book contains a series of SAS 9.3 programming examples that illustrate some of the common tasks that stored processes are used to perform. Many of these examples also contain videos illustrating what these stored processes can do when you run them. You can use the SAS windowing environment (or a text editor) as your programming environment with these examples, or you can use SAS Enterprise Guide. If you use the SAS windowing environment, then you can use the New Stored Process wizard in SAS Management Console to register the metadata for your program. If you decide to use SAS Enterprise Guide, then the metadata registration is included in the Create New SAS Stored Process wizard. The steps in both of these wizards are almost the same, and each wizard is illustrated in the Registering Stored Process Metadata section.

The Programming Examples chapter focuses mainly on examples that are run in the SAS Stored Process Web Application. In the Client Examples chapter, there are also examples of how stored processes fit in with many of the clients. The clients illustrated in this book are the 4.31 and 5.1 clients (see the Client Examples chapter for the release information for each client) and are being used with SAS 9.3 Stored Processes. The client examples are not a replacement for the official documentation for each client, and in most cases these examples assume that you have some prior experience with that client.

This book is NOT by any means a replacement for the official SAS Stored Process reference documentation, the *SAS Stored Processes: Developer's Guide*, which is located at http://support.sas.com/documentation/onlinedoc/inttech/index.html. This book can be used as a supplement to the developer's guide. Hopefully, you will enjoy the extension of the material that is already available and find this guide useful.

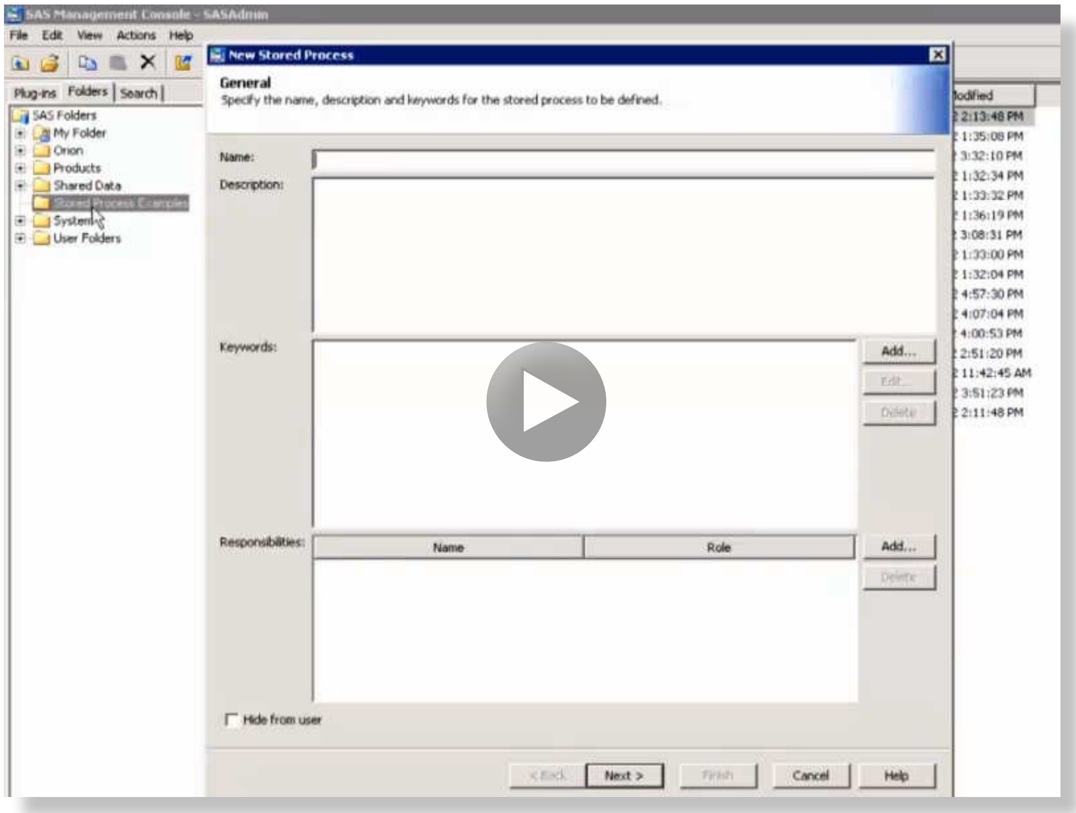
1.4 Registering Stored Process Metadata

A stored process is registered in metadata so that it can be easily located by the client that will run it. The only things that you must specify when registering a stored process are the name of the stored process and the execution details (what server will run the stored process and where that stored process is located, including the file name). Beyond those details, you also have the option to specify parameters for the stored process and for data sources or targets, among other things. Now, let's concentrate on the basic stored process registration. Some of the examples in this book use other parts of the metadata.

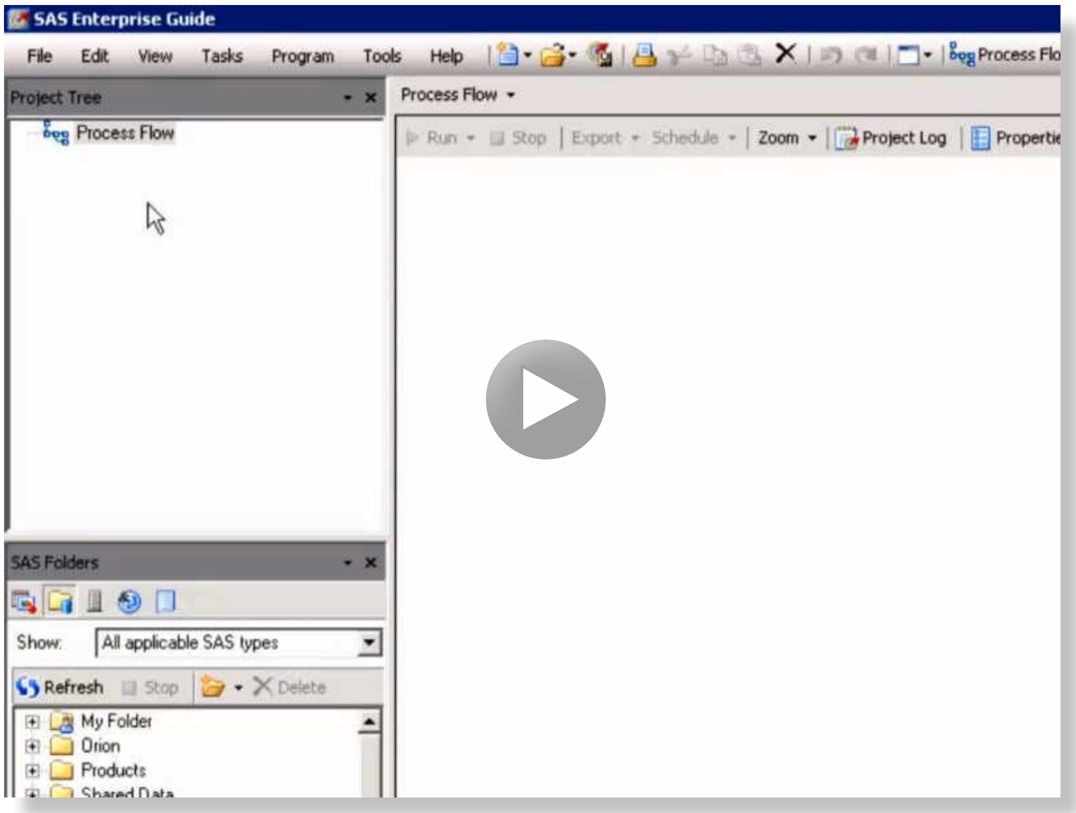
Note that there are differences in SAS Management Console between the way you specify execution details in SAS 9.3 and how they were specified in SAS 9.2. The examples in this book use SAS 9.3. For more information about using SAS 9.2 or for using SAS 9.3 to register stored processes that are compatible with SAS 9.2, see the product Help. (This also applies to SAS Enterprise Guide. This book uses SAS Enterprise Guide 5.1, which has some significant differences from SAS Enterprise Guide 4.3. For more information, see the product Help.)

The following examples show the New Stored Process wizard in SAS Management Console and the Create New SAS Stored Process wizard in SAS Enterprise Guide. You should be aware that these wizards, used for creating and registering new stored processes, are nearly identical to their counterparts that are used for modifying existing stored processes (the Stored Process Properties dialog box in SAS Management Console and the Stored Process Manager in SAS Enterprise Guide).

The following video shows you how to register a stored process in SAS Management Console:



The following video shows you how to create a stored process in SAS Enterprise Guide:


SAS Enterprise Guide
File Edit View Tasks Program Tools Help
Process Flow
Project Tree
Process Flow
Run Stop Export Schedule Zoom Project Log Properties
SAS Folders
Show: All applicable SAS types
Refresh Stop Delete
My Folder
Orion
Products
Shared Data

Please refer to the SAS Enterprise Guide and SAS Management Console product Help for more information about registering and modifying metadata for stored processes.

1.5 Running the Examples in this Book

The examples in this book show the SAS code that needs to be registered as a stored process. If you want to run these examples, you need to follow the steps shown in the previous section's videos (or outlined in the product Help) to register the stored process metadata. You need to know what server will run the stored process and where you want to store the code. Anywhere in the examples in this book that you see *yourserver.xxx*.com, please replace this with your server information. These examples should be easy to run, but you will need to do a bit more than copying and pasting to run them successfully.

2

Programming Examples

2.1 Using ODS Options with the %STPBEGIN and %STPEND Macros

The %STPBEGIN and %STPEND macros generate ODS statements for you based on the values of reserved macro variables (this is only part of what the macros do – they also make packages). However, you can also use ODS with your stored processes to do things like specify different output destinations and modify the appearance of output. The following code uses the _ODSDEST macro variable to create a PDF file that displays the CLASS data set from the Sashelp library:

```
%global _ODSDEST;

*ProcessBody;

data _null_;
   rc = stpsrv_header('Content-type','application/pdf');
   rc = stpsrv_header('Content-disposition','attachment;
      filename=temp.pdf');
run;
```

```
%let _odsdest=pdf;

%STPBEGIN;

proc print data=sashelp.class;
run;

%STPEND;
```

Note that the STPSRV_HEADER functions and the %LET statement are placed before the stored process code. (The HEADER functions need to be placed before any code that writes results to _WEBOUT.) Similarly, the following code uses the same _ODSDEST macro variable, this time to produce a CSV file:

```
%global _ODSDEST;

*ProcessBody;

data _null_;
   rc = stpsrv_header('Content-type','application/vnd.ms-excel ');
   rc = stpsrv_header('Content-disposition','attachment;
      filename=temp.csv');
run;

%let _odsdest=csv;

%STPBEGIN;

proc print data=sashelp.class;
run;

%STPEND;
```

You can also specify other ODS options when using the %STPBEGIN macro. The following example illustrates how to use the ODS STYLE option with the %STPBEGIN macro:

```
%global _ODSSTYLE;

%let _ODSSTYLE=Seaside;

%STPBEGIN;
   proc print data=sashelp.class;
   run;
%STPEND;
```

Note that using the _ODSDEST and _ODSSTYLE macro variables limits which clients can execute the stored process. For example, SAS Web Report Studio could not use the _ODSDEST programs. Also, SAS Web Report Studio ignores _ODSSTYLE and uses its own native style.

If you want to see ODS statements that the %STPBEGIN and %STPEND macros generate, you can use the MPRINT system option right before the %STPBEGIN macro. The log shows all the statements that are generated by the %STPBEGIN and %STPEND macros.

2.2 Customizing Input Forms

You can create your own custom input forms if you are working with the SAS Stored Process Web Application. For example, the following HTML can be saved as a JSP file somewhere that is accessible by your web server:

```
<html><body>
<form method="POST" action="http://yourserver.com:8080/SASStoredProcess/do">
<input type="hidden" name="_program"
   value="/Stored Process Examples/Custom Input Form">
Select the age of the students to be listed:
<select name="age" >
<option value="11"> 11
<option value="12"> 12
<option value="13"> 13
<option value="14"> 14
<option value="15"> 15
<option value="16"> 16
</select>

<br><br>
<input type=checkbox name="_debug" value="log"> Show SAS log?
<br><br>
<input type=submit value="Execute Stored Process">
</body></html>
```

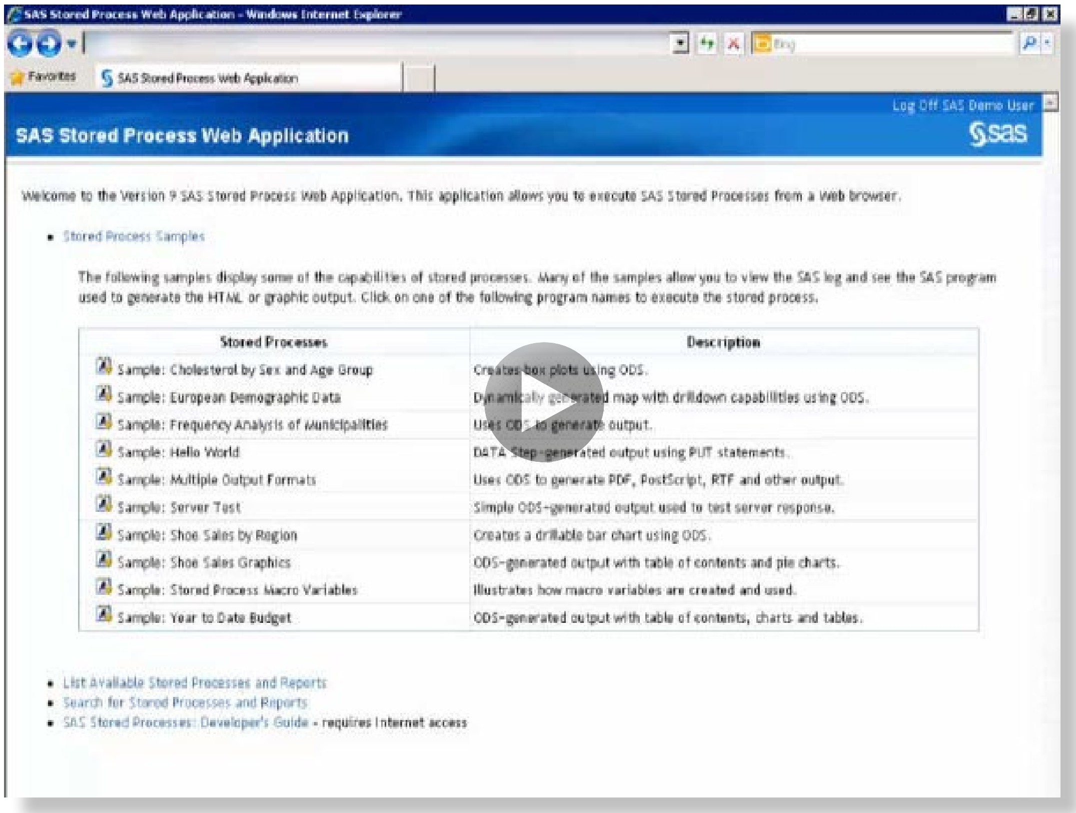
Note that you can save this JSP file in the **`/input`** directory for your web application server, as in previous releases, but then you run the risk of losing your input form if you apply any updates to your software. As a best practice, save it in another directory on your web application server. For this example, save the JSP file in the JBOSS root directory, in this case: **`C:\Program Files\EnterprisePlatform-4.3.0.GA_CP08\jboss-as\server\SASServer1\deploy\jboss-web.deployer\ROOT.war\Custom Input Form.jsp`**.

In SAS Management Console, in the **`Stored Process Examples`** folder, specify **`Custom Input Form`** as the name of your new stored process. For this example, the source code is stored on the metadata server. Specify the following source code:

```
*ProcessBody;

%STPBEGIN;

proc print data=sashelp.class;
   where age = &age;
   title1 "&age-Year-Old Students";
   run;

%STPEND;
```

Define a new, hidden, read-only text prompt named _FORM. Give it a default value of **`http://`*`yourserver`*`.com:8080/Stored Process Examples/Custom Input Form.jsp`**.

Go to the SAS Stored Process Web Application (**`http://`*`yourserver`*`.com:8080/SASStoredProcess/do`**), and navigate to **`/Stored Process Examples/Custom Input Form`**.


SAS Stored Process Web Application - Windows Internet Explorer
Favorites
SAS Stored Process Web Application
Log Off SAS Demo User
SAS Stored Process Web Application

Welcome to the Version 9 SAS Stored Process Web Application. This application allows you to execute SAS Stored Processes from a Web browser.

- Stored Process Samples

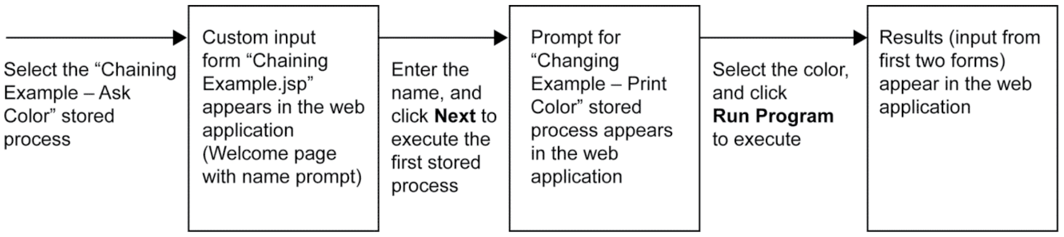
The following samples display some of the capabilities of stored processes. Many of the samples allow you to view the SAS log and see the SAS program used to generate the HTML or graphic output. Click on one of the following program names to execute the stored process.

Stored Processes	Description
Sample: Cholesterol by Sex and Age Group	Creates box plots using ODS.
Sample: European Demographic Data	Dynamically generated map with drilldown capabilities using ODS.
Sample: Frequency Analysis of Municipalities	Uses ODS to generate output.
Sample: Hello World	DATA Step-generated output using PUT statements.
Sample: Multiple Output Formats	Uses ODS to generate PDF, PostScript, RTF and other output.
Sample: Server Test	Simple ODS-generated output used to test server response.
Sample: Shoe Sales by Region	Creates a drillable bar chart using ODS.
Sample: Shoe Sales Graphics	ODS-generated output with table of contents and pie charts.
Sample: Stored Process Macro Variables	Illustrates how macro variables are created and used.
Sample: Year to Date Budget	ODS-generated output with table of contents, charts and tables.

- List Available Stored Processes and Reports
- Search for Stored Processes and Reports
- SAS Stored Processes: Developer's Guide - requires Internet access

2.3 Chaining Stored Processes

Stored processes can be chained together, like a daisy chain, where the first stored process calls a second, which can call a third stored process, and so on. You can use this method to create web applications of your own, such as forms and surveys. For example, suppose you want to create a form that has three pages. This example creates a mini-web application that has a welcome page that asks for the user's name, a second page that asks for the user's favorite color, and a third page that enables the user to review the input from the first two pages. The following diagram illustrates how our mini-web application works:


Select the "Chaining Example – Ask Color" stored process
Custom input form "Chaining Example.jsp" appears in the web application (Welcome page with name prompt)
Enter the name, and click **Next** to execute the first stored process
Prompt for "Changing Example – Print Color" stored process appears in the web application
Select the color, and click **Run Program** to execute
Results (input from first two forms) appear in the web application

The first page, the welcome page, is a JSP page that you can create the same way you created the custom input form example. The following HTML is stored somewhere on the web server (this file is named “Chaining Example.jsp” and uses the JBoss root directory: **`C:\Program Files\EnterprisePlatform-4.3.0.GA_CP08\jboss-as\server\SASServer1\deploy\jboss-web.deployer\ROOT.war\Chaining Example.jsp`**):

```
<html>
<head><title>Welcome to MyWebApp
</title></head>
<body><h1>Welcome to MyWebApp</h1>
<form action="/SASStoredProcess/do">Please enter your name:
<input type="text" name="FNAME"><BR>
<input type="hidden" name="_program" value="/Stored Process
   Examples/Chaining Example - Ask Color">
<br><br>
<input type="submit" value="Next">
</form>
</body></html>
```

In this JSP file, the value of _PROGRAM is the location of the first stored process, which actually creates the second page in the chain. Define a new stored process in SAS Management Console named **`Chaining Example - Ask Color`**. Specify that the stored process produces streaming output. This stored process can be stored on the metadata server and contains the following code:

```
data _null_;
   file _webout;
   put '<html>';
   put '<body>';
   put '<h1>Welcome to MyWebApp</h1>';

   /* Create reference back to the SAS Stored Process Web */
   /* Application from special automatic macro variable _URL. */

   put "<form action='&_URL'>";

   /* Specify the stored process to be executed using the */
   /* _PROGRAM variable. */

    put '<input type="hidden" name="_program" '
   'value="/Stored Process Examples/Chaining Example - Print Color">';

   /* Pass first name value on to next program. */
   /* The value is user entered text, so you must */
   /*encode it for use in HTML. */

   fname = htmlencode("&FNAME", 'amp lt gt quot');
   put '<input type="hidden" name="fname" value="'
   fname +(-1) '">';

   put 'What is your favorite color?';
   put '<select size=1 name="fcolor">';
   put '<option value="red">red</OPTION>';
   put '<option value="green">green</OPTION>';
   put '<option value="blue">blue</OPTION>';
   put '<option value="other">other</OPTION>';
   put '</select><br><br><br>';
   put '<input type="submit" value="Run Program">';
   put '</form>';
   put '</body>';
   put '</html>';
run;
```

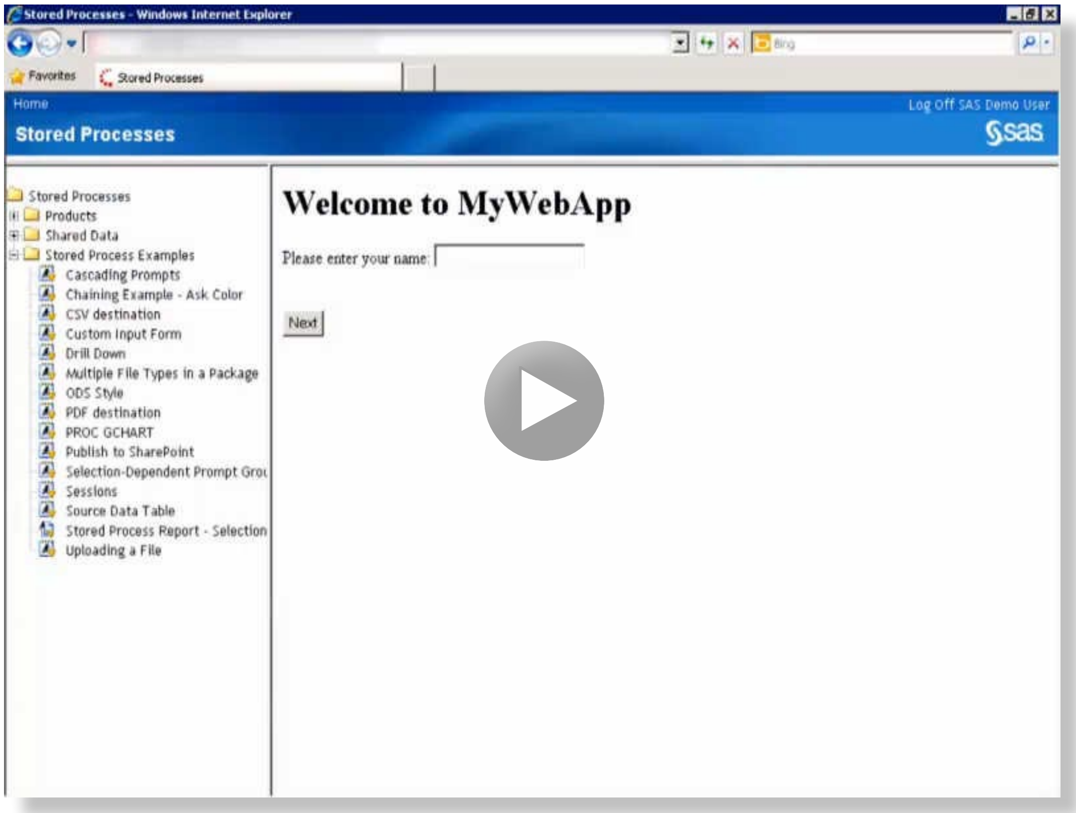
Note that because this example does not use ODS, no %STPBEGIN and %STPEND macros are included. They are not needed here to set up ODS statements.

Define a new, hidden, read-only text prompt named _FORM. Give it a default value of **`http://`** ***`yourserver`*** **`.com:8080/Stored Process Examples/Chaining Example.jsp`**.

The second stored process, which creates the third page in this web application, is named **`Chaining Example - Print Color`**. Note that the value of _PROGRAM in the first stored process points to the location of this second stored process. When you register the metadata for this stored process, you can select **Hide from user** so that users will not see it listed as an independent stored process in applications like the SAS Stored Process Web Application. This helps ensure that it will be accessed only as part of the stored process chain. The following code makes up this final stored process in the chain:

```
data _null_;
   file _webout;
   put '<html>';
   put '<body>';
   fname = htmlencode("&FNAME");
   put 'Your name is <b>'
   fname +(-1) '</b>';
   put '<br>';
   put "Your favorite color is <b>&FCOLOR</b>";
   put '<br>';
   put '</body>';
   put '</html>';
run;
```

Go to the SAS Stored Process Web Application (**`http://yourserver.com:8080/SASStoredProcess/do`**), and navigate to /**`Stored Process Examples/Chaining Example - Ask Color`**.

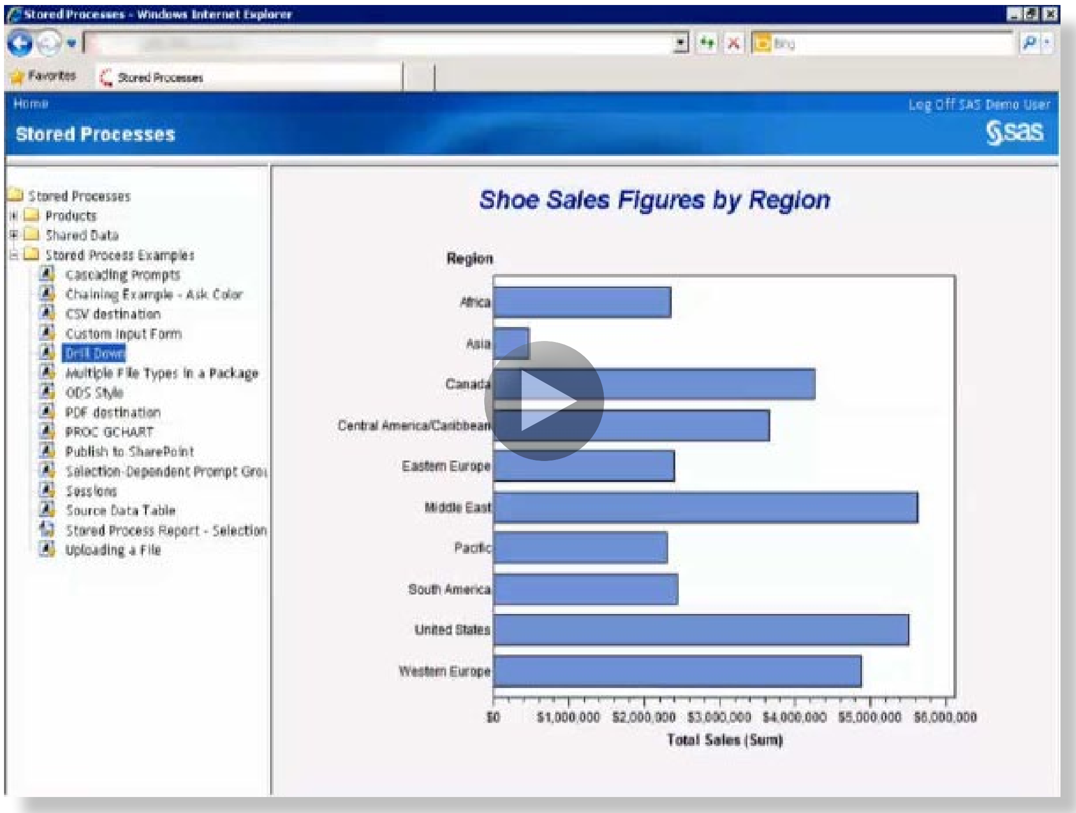


2.4 Enabling Drill Down

You can use macro processing to create drill down results in your HTML file. This particular example displays a bar chart of shoe sales by region. You can click any of the bars to drill down to a detailed report of the sales in that region, broken down by subsidiary and shoe style.

In the following example, the Work.Shoes data set is created to contain HREF hyperlinks. This data set generates the client side image map. The **`HTML=HREF`** parameter in the GCHART procedure specifies that the data set variable named HREF contains the hyperlink text. The GCHART procedure creates the initial bar chart of shoe sales by region. The REPORT procedure creates the detailed reports for each region that are linked to from the bar chart.

```
%global _GOPT_DEVICE _GOPTIONS _ODSOPTIONS;

legend1 label=none across=2 frame;

title "Shoe Sales Figures by Region";
footnote;

%STPBEGIN;

data work.shoes;
   set sashelp.shoes;
   by region;
   length rnumber 8 file $80 href $32767;
   retain rnumber 0;
   if first.region then rnumber + 1;
   file = "region"     || trim(left(put(rnumber, 4.0))) || ".html";
   href  = 'href="'    || &_replay || trim(file) || '"';
run;

proc gchart data=work.shoes;
   hbar region / sumvar=sales
                 html=href;
run;
quit;

%STPEND;

%let _ODSOPTIONS=%str(body=region1.html newfile=table);

%STPBEGIN;

proc report data=work.shoes nowd;
   column region subsidiary product sales inventory returns;
   define region       / order noprint  "Region";
   define subsidiary   / order          "Subsidiary";
   define product      / order          "Shoe Style";
   define sales        / sum            "Sales";
   define inventory    / sum            "Inventory";
   define returns      / sum            "Returns";
   break after subsidiary / summarize suppress style=Header;
   break before region / page ;
   compute before _page_ / center style=BeforeCaption;
      line "Shoe Sales Figures for " region $25.;
   endcomp;
run;

%STPEND;
```



2.5 Working with Sessions

You can also use macro processing with sessions. A *session* is often thought of as an environment where software is executed – that is not the case here. A *stored process session* is a temporary storage location that can be accessed by stored processes by using a session ID. Sessions are useful in larger applications where it is inconvenient to pass parameter values in a URL or through HTML. Sessions are also used to store summarized data or for process-intensive tasks that need to be run only once. Each stored process request to the stored process server has a session. That session is saved after the execution of the stored process if the STPSRV_SESSION stored process server function was used to explicitly create the session or if data has been written to the session. Sessions expire after a defined timeout period – the default is 15 minutes, but it can be changed using the STPSRVSET function (see the *SAS Stored Process: Developer's Guide*). You can use this stored process server function in the stored process code, or you can define it as a parameter in SAS Management Console.

The following is a short example of using sessions. On the first page, the user provides values for three parameters. When the first request is executed by the first macro, a session is created, a data set is created in the Save library, and the second page displays the values that were entered on the first page, along with a session ID. When the second request is executed by the second macro, a session ID is passed, and the third page displays the values again (along with the data set that was saved) to show that the values have been saved in the session and passed all the way through the application.

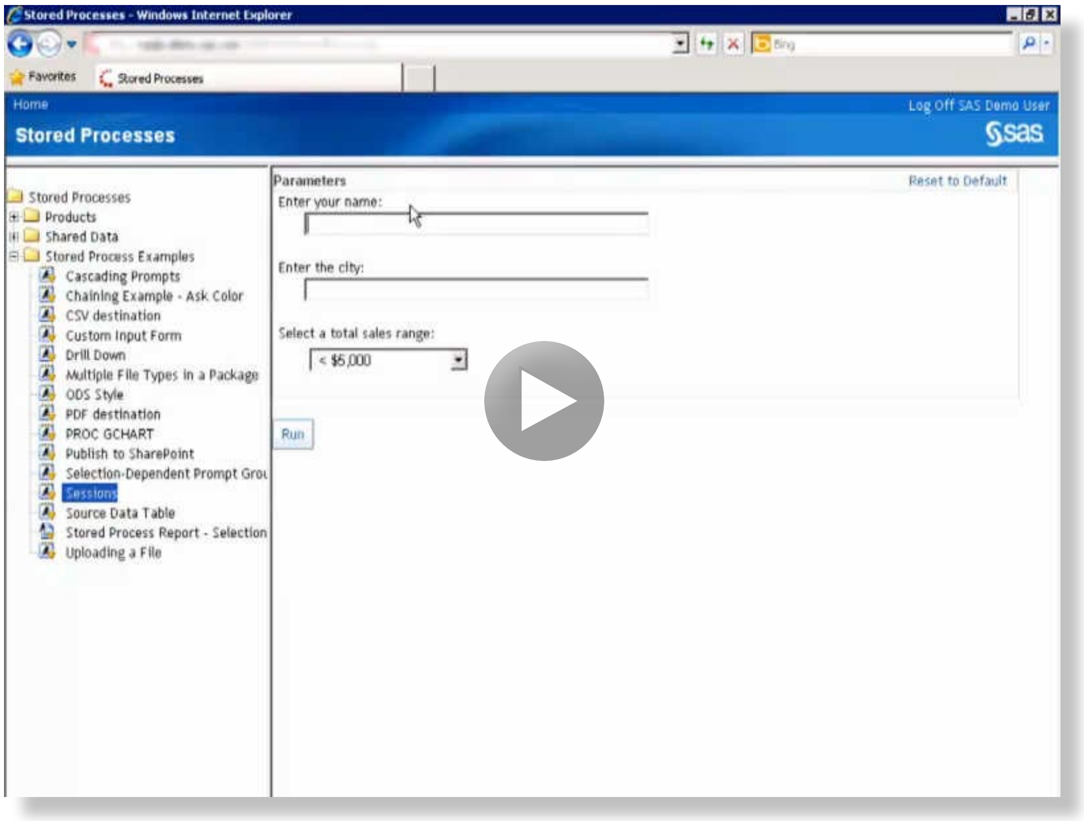
Note that the Save library is automatically created when the %sysfunc(stpserv_session(create)) function creates a session. The Save library and SAVE_ variables exist only for the life of the session.

Store the code for this example on the application server rather than in the metadata. Register three parameters for this stored process: name, city, and range. Name and city are each user-entered text parameters. For the text parameter range, the user selects a single value from a static list. The values for the static list were added as follows (formatted values are displayed in the run-time prompt, and unformatted values are sent to the server):

Unformatted Value	Formatted Value
`<5000`	`< $5,000`
`in (5000:25000)`	`$5,000 - $25,000`
`in (25000:50000)`	`$25,000 - $50,000`
`in (50000:100000)`	`$50,000 - $100,000`
`in (100000:250000)`	`$100,000 - $250,000`
`in (250000:500000)`	`$250,000 - $500,000`
`in (500000:1000000)`	`$500,000 - $1,000,000`
`>1000000`	`> $1,000,000`

The code for this example follows:

```
*ProcessBody;

%global _sessionid save_name save_city;

%macro firstreq;

   /* This is the first user request - Create a New Session */

   %let rc=%sysfunc(stpsrv_session(create));

   /* Create SAVE_ macro variables for session */

   %let save_name=&name;
   %let save_city=&city;

   data _null_;
      file _webout;
      put '<html>';
      put '<head><title>Session Test</title></head>';
      put '<body>';
      put '<h1>Session Test - First Request</h1>';
      put "Your sessionid is &_sessionid <br>";
        x3=symget('_THISSESSION');
      put "Your _thissession is " x3 "<br><br>";
      put "<b>The following macro variables were saved in the session:
        </b><br><br>";
      put "Value of save_name is: &save_name";
      put "<br>";
      put "Value of save_city is: &save_city";
      put "<br><br><b>Data set save.salesrange was saved in the session,
        where total sales &range.</b>";
      put "<br>";
      put '<form action="http://' "&_srvname:&_srvport" "&_url" '">';
      put '<input type="hidden" name="_program" value="' "&_program" '">';
      put '<input type="hidden" name="_sessionid" value="'
        "&_sessionid" '">';
      put '<input type="submit" value="Get variables saved in session by first
         request">';
      put '</form>';
      put '</body>';
      put '</html>';
   run;

   %put save_name is: &save_name;
   %put save_city is: &save_city;

   /* Create data set in SAVE library */

data save.salesrange;
   set sashelp.shoes;
   where sales &range;
run;

%mend firstreq;

%macro nextreq;

   /* This is NOT the first request                        */
   /* Use session that was created in the first request.  */

   %put This is the second request;
   %put Values saved in previous request;
   %put save_name is: &save_name;
   %put save_city is: &save_city;

   ods listing close;

   ods html body=_webout (title='Session Test');
   ods text='<h1>Session Test - Second Request</h1>';
   ods text='<h3>Variables that were created in the previous request and
     stored in the session:</h3>';
   ods text="Value of save_name (Assigned in previous request) is:
     &save_name";
   ods text="Value of save_city (Assigned in previous request) is:
     &save_city";
   ods text='<h3>The following data set was retrieved from the session:</h3>';

   proc print data=save.salesrange;
   run;

   ods html close;

%mend nextreq;

%macro main;

   /* If a session ID is passed, it is not the first request */

   %if "&_sessionid" = "" %then %do;

      %firstreq;
   %end;
   %else %do;
      %nextreq;
   %end;

%mend main;

%main;

run;
```



2.6 Uploading Files

You can use a custom input form in the SAS Stored Process Web Application to upload files to the stored process server or the workspace server. This is useful for tasks like uploading a CSV file to a SAS table; uploading an Excel workbook to one or more SAS tables; and uploading a SAS table, view, or catalog.

You can use the following example to upload a SAS data set and then view the first 10 observations in the web application. This particular custom input form enables you to upload only one file at a time, but you can make custom input forms that upload more than one file at a time. Also, you can use the same HTML for the custom input form, no matter what type of file you are uploading. You would need to change the SAS code in the stored process if you were uploading a file that was not a SAS table or view.

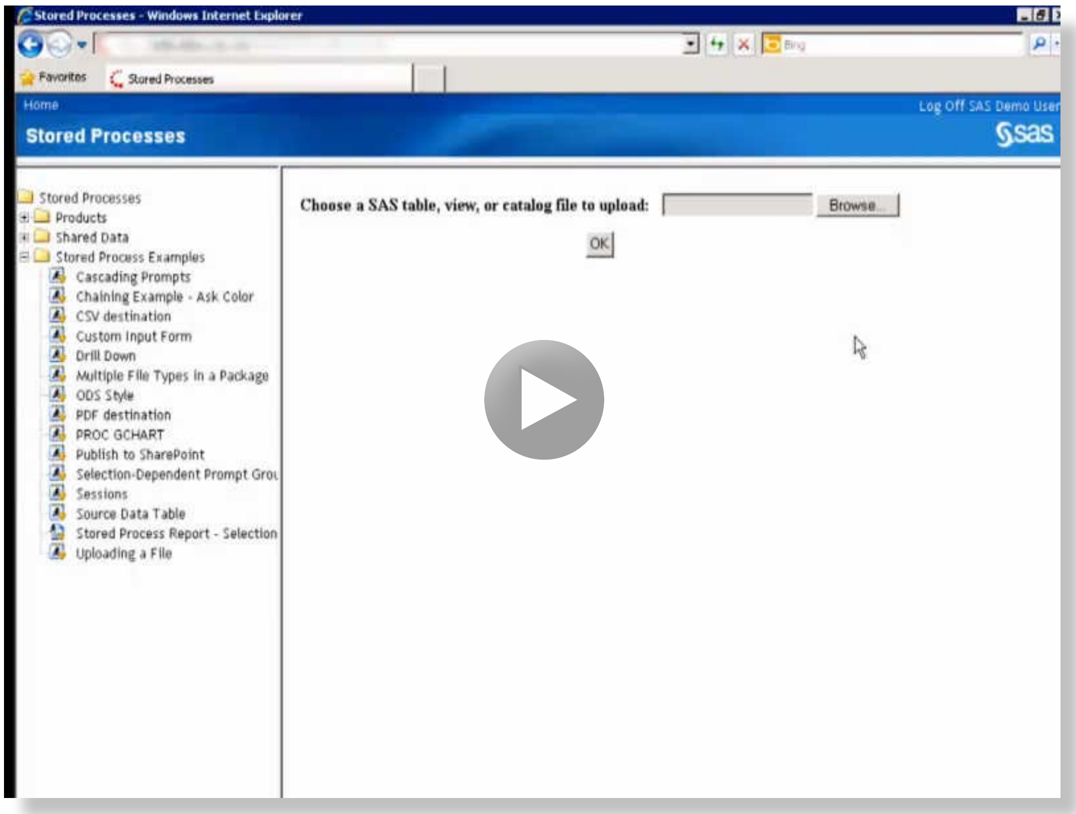
The first page is a JSP page that you create the same way you created the custom input form example. The following HTML is stored somewhere on the web server (name the file “File Upload Form.jsp” and use the JBoss root directory, as in the custom input form example):

```
<html><body>
<form action="http://yourserver.com:8080/SASStoredProcess/do" method="post"
   enctype="multipart/form-data">
<input type="hidden" name="_program" value="/Stored Process Examples/Uploading
a File">
<table border="0" cellpadding="5">
   <tr>
      <th>Choose a SAS table, view, or catalog 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>
</body></html>
```

Define a new stored process named **`Uploading a File`**. When you define the stored process, define a new, hidden, read-only text prompt named _FORM. Give it a default value of **`http://`** ***`yourserver`*** **`.com:8080/Stored Process Examples/File Upload Form.jsp`**. This stored process can be stored on the metadata server and contains the following code:

```
title 'First 10 records of uploaded SAS data file.';

%STPBEGIN;
   proc print data=&_WEBIN_SASNAME(obs=10); run; quit;
%STPEND;
```

Go to the SAS Stored Process Web Application (**`http://yourserver.com:8080/SASStoredProcess/do`**), and navigate to /**`Stored Process Examples/Uploading a File`**.



For more information about using the _WEBIN_* macro variables and how to perform other tasks with uploaded files, see “Uploading Files” in the *SAS Stored Processes: Developer’s Guide*.

2.7 Including Multiple File Types in a Package

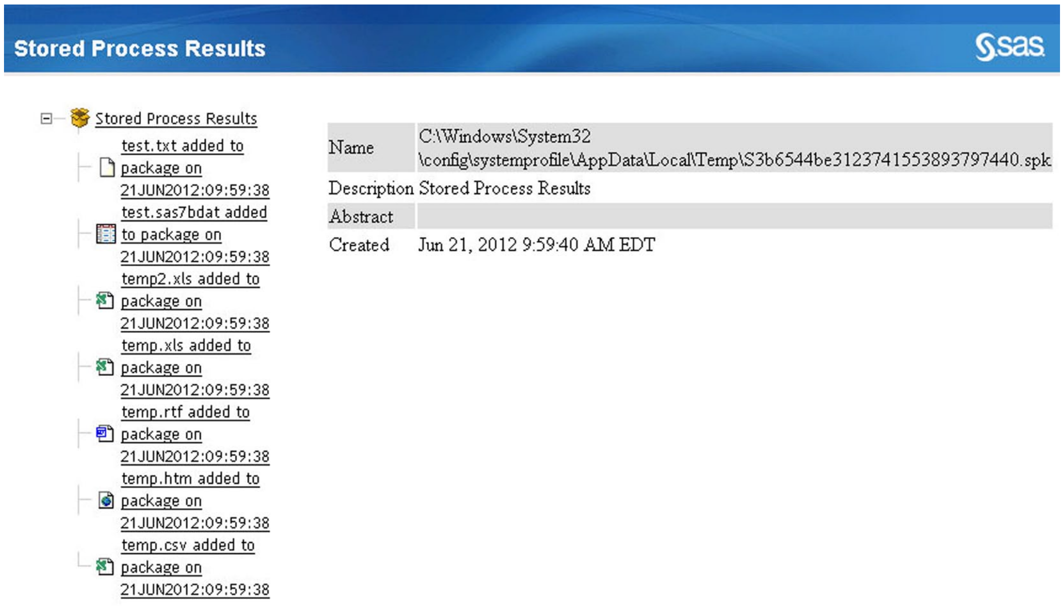
With ODS, you can create multiple output types. With a stored process, you can include files with different output types in one package. For this example, create a stored process named **`Multiple File Types in a Package`** and select the Package result capability. The stored process contains the following code:

```
*ProcessBody;

%let _ODSDEST=tagsets.ExcelXp;
%let _ODSOPTIONS=file="temp.xls";
%let _ODSSTYLESHEET=;

%STPBEGIN;

   /*  Files in the "&_STPWORK" directory will be copied to the result */
   /*  package.                                                        */

libname stpwork "&_STPWORK";

   /* Insert the "temp.xls" file using "tagsets.ExcelXp" */

proc print data=sashelp.class;
run;

ods &_ODSDEST close;

   /* Insert a SAS data set in the result package */

data stpwork.test;
   set sashelp.class;
run;

ods html file='temp.htm' path="&_STPWORK";

   /* Insert the "temp.htm" file  */

proc print data=sashelp.class;
run;

ods html close;

   /* Insert "temp2.xls".  This is an HTML file that    */
   /* is disguised as an XLS file. Excel will be used   */
   /* to open the file.                                 */

ods html file='temp2.xls' path="&_STPWORK";

proc print data=sashelp.class;
run;

ods html close;

   /* Insert the "temp.csv" file  */

ods csv file='temp.csv' path="&_STPWORK";

proc print data=sashelp.class;
run;

ods csv close;

   /* Insert the "temp.rtf" file  */
```

```
ods rtf file='temp.rtf' path="&_STPWORK";

proc print data=sashelp.class;
run;

ods rtf close;

   /* Insert "test.txt" file  */

filename stpout "&_STPWORK\test.txt";

data _null_;
   file stpout;
   put "This is a test file inserted in the result package";
run;

%STPEND;
```

Run the stored process in the SAS Stored Process Web Application, and a tree view listing of the results appears, showing each file that was created. You can click these links to open each of the created files.

Note that you need to use the second release for SAS 9.3 for this example to work correctly.



Stored Process Results
- test.txt added to package on 21JUN2012:09:59:38
- test.sas7bdat added to package on 21JUN2012:09:59:38
- temp2.xls added to package on 21JUN2012:09:59:38
- temp.xls added to package on 21JUN2012:09:59:38
- temp.rtf added to package on 21JUN2012:09:59:38
- temp.htm added to package on 21JUN2012:09:59:38
- temp.csv added to package on 21JUN2012:09:59:38

Name	C:\Windows\System32 \config\systemprofile\AppData\Local\Temp\S3b6544be3123741553893797440.spk
Description	Stored Process Results
Abstract	
Created	Jun 21, 2012 9:59:40 AM EDT

2.8 Using SAS/GRAPH Software

If you are using SAS/GRAPH software with stored processes, then you can use the _GOPT_DEVICE macro variable to change the device driver to PNG or GIF. This example adds the following statement to the stored process code prior to the %STPBEGIN statement:

```
%let _GOPT_DEVICE=PNG;
```

Note that starting with SAS 9.2, it's preferable to use **`DEV=PNG`** rather than GIF because PNG supports 16 million colors, while GIF supports only 256.

Use the following statement (you might eliminate the style or select a different style) when you are using a SAS/GRAPH program as a stored process:

```
%let _ODSOPTIONS=gtitle gfootnote style=Seaside;
```

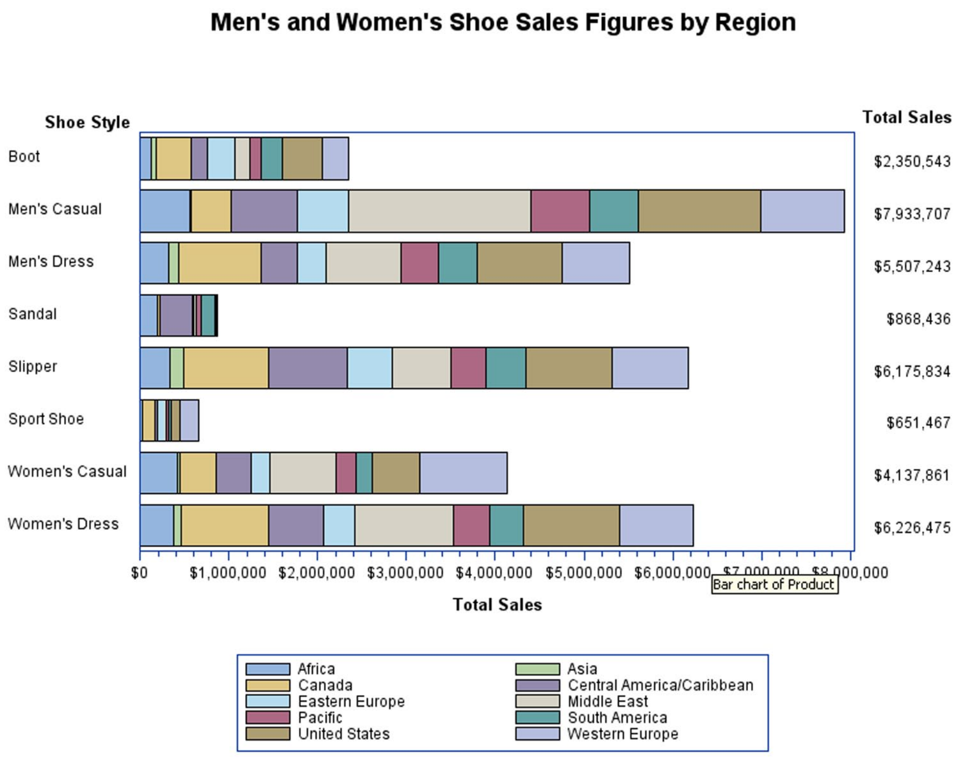
This statement keeps the titles and footnotes inside the graph (otherwise, the titles and footnotes appear outside the graphics image as text in the body of the file). This is important when you are using the GREPLAY procedure to create dashboards.

When you register the stored process, specify streaming as the type of output. You can also add the _ODSOPTIONS and _GOPT_DEVICE parameters to the stored process instead of defining those parameters with %LET statements. If you do not want the user to see or change the value, mark the parameters as non–visible and non–modifiable.

The following SAS/GRAPH code is used in this stored process:

```
%global _GOPT_DEVICE _ODSOPTIONS;
%let _GOPT_DEVICE=png;
%let _ODSOPTIONS=gtitle gfootnote style=Seaside;

legend1 label=none across=2 frame;

title "Men's and Women's Shoe Sales Figures by Region";
footnote;

%STPBEGIN;

proc gchart data=sashelp.shoes;
   hbar product / sumvar=sales
                  subgroup=region
                  sum
                  patternid=subgroup
                  legend=legend1;
   label product='Shoe Style';
run;
quit;

%STPEND;
```

This example produces the following output:



2.9 Using Prompts

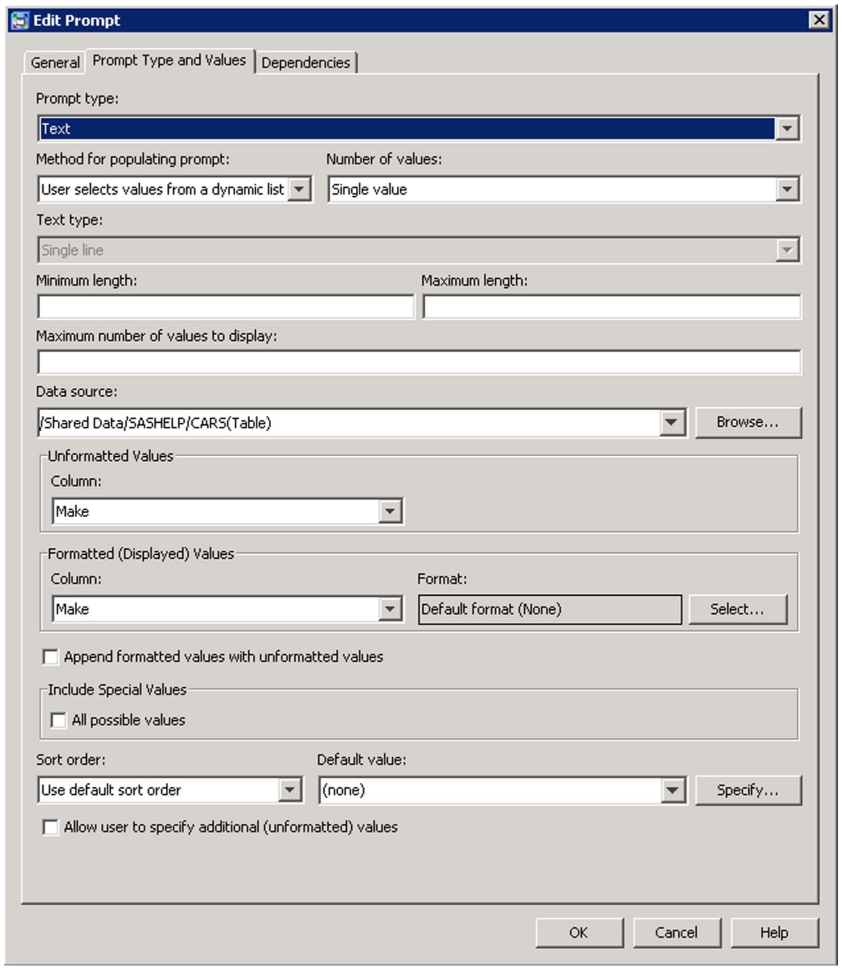
The prompt examples in this book illustrate some of the more advanced prompting framework concepts – cascading prompts and selection-dependent prompt groups. For more information about prompts in general, see the product Help.

Using Cascading Prompts

Cascading prompts populate prompt values based on selections in other prompts. In this example, after you select a value for **Origin**, you can select a value for **Make**, and then **Type**. The values that are available for **Make** depend on the value that was selected for **Origin**, and the value that is selected for **Make** in turn determines which values are available for **Type**. The Sashelp. Cars data set is then subset by these selections, and you see a report of cars that meet all of these criteria.

This stored process, named **`Cascading Prompts`**, is executed on the stored process server and can create package output. Make sure that the Sashelp.Cars data set is registered as a table in SAS Management Console before you define the prompts for this stored process. (You can register libraries and tables using the Data Library Manager plug-in for SAS Management Console.) The prompts are all members of a standard prompt group, titled **Specify Vehicle Information**. The prompts are defined as follows:

Name	Origin	Make	Type
Displayed text	Origin	Make	Type
Options	Requires a non-blank value		
Prompt type	Text		
Method for populating prompt	User selects values from a dynamic list		
Number of values	Single value		
Data source	`/SampleSTPs/CARS(Table)`		
Column	Origin	Make	Type
Formatted (Displayed Values)	Origin	Make	Type
Dependencies tab	Nothing added here, though after Make and Type are defined, they appear at the bottom of this tab in the list of prompts that depend on the value of this prompt.	Add a new dependency where the column Origin contains the value of Origin. After Type is defined, it appears at the bottom of this tab in the list of prompts that depend on the value of this prompt.	Add two new dependencies where the column Origin contains the value of Origin, and the column Make contains the value of Make.



Prompt type: Text

Method for populating prompt: User selects values from a dynamic list

Number of values: Single value

Text type: Single line

Minimum length:

Maximum length:

Maximum number of values to display:

Data source: /Shared Data/SASHELP/CARS(Table)

Unformatted Values

Column: Make

Formatted (Displayed) Values

Column: Make

Format: Default format (None)

☐ Append formatted values with unformatted values

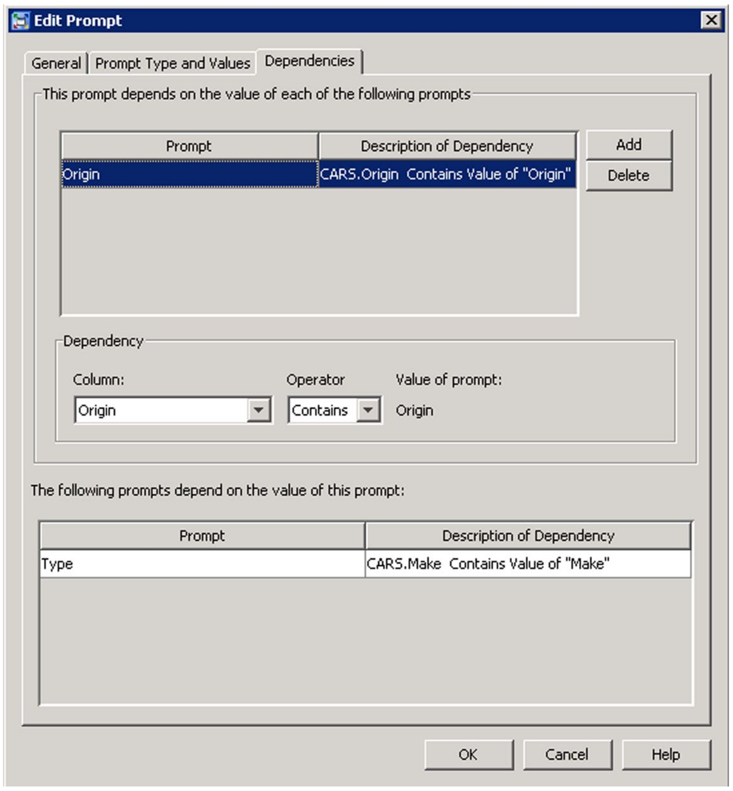
Include Special Values

☐ All possible values

Sort order: Use default sort order

Default value: (none)

☐ Allow user to specify additional (unformatted) values



This prompt depends on the value of each of the following prompts

Prompt	Description of Dependency
Origin	CARS.Origin Contains Value of "Origin"

Dependency

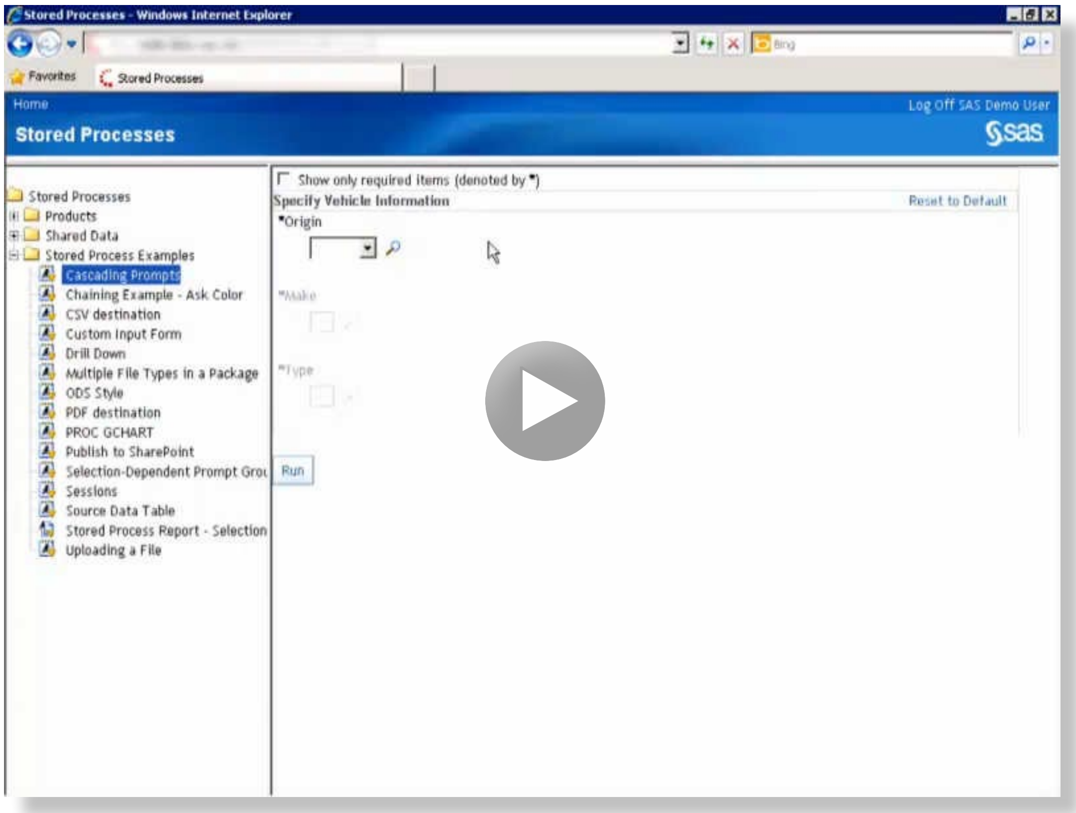
Column: Origin
Operator: Contains
Value of prompt: Origin

The following prompts depend on the value of this prompt:

Prompt	Description of Dependency
Type	CARS.Make Contains Value of "Make"

The code for this stored process is as follows:

```
%STPBEGIN;
   proc print data = sashelp.cars noobs label n;
      where (Origin = "&ORIGIN" AND Make = "&MAKE" AND Type = "&TYPE");
      title "CAR Info - Origin: &origin/Manufacturer: &make/Type: =
         &type)";
   run;
%STPEND;
```



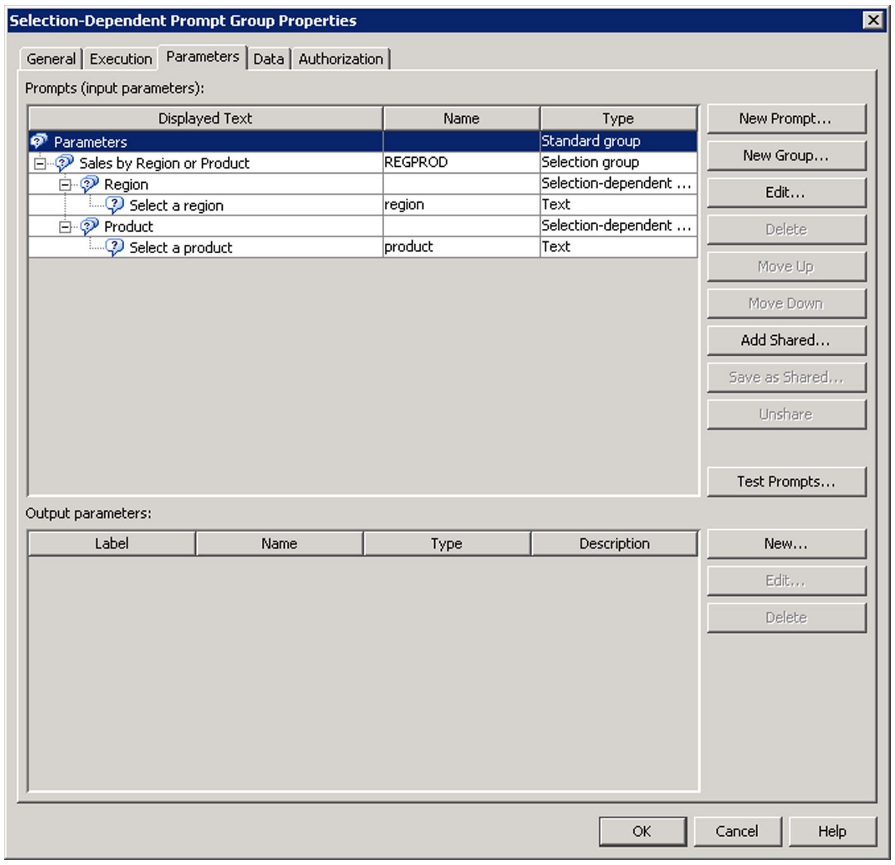
Using Selection-Dependent Prompt Groups

Selection groups enable you to display different prompts based on a condition. In this example, you can select whether to graph shoe sales by region or by product. Selection groups contain selection-dependent groups. After you choose **Region** or **Product**, a second prompt appears, depending on what you selected first. If you selected **Region**, then you are prompted to select a specific region. If you selected **Product**, then you are prompted to select a specific product. When you click **Run**, the total sales are graphed based on your selections.

This stored process, named `Selection-Dependent Prompt Group`, creates package output and is stored on the metadata server. Make sure that the Sashelp.Shoes data set is registered as a table in SAS Management Console before you define the prompts for this stored process.

To define this set of prompts, first you must define the selection group, which is the parent group of the Region and Product selection-dependent groups. The displayed text is **Sales by Region or Product** and the name of the group is REGPROD. The displayed text for one of the selection-dependent groups is **Region** and the other is **Product**. The values for those groups are REG and PROD, respectively. The prompts in the selection-dependent groups are defined as follows:

Parent group	Region	Product
Name	Region	Product
Displayed text	Select a region	Select a product
Options	None	
Prompt type	Text	
Method for populating prompt	User selects values from a dynamic list	
Number of values	Single value	
Data source	**`/Shared Data/SHOES(Table)`**	
Column	Region	Product
Formatted (Displayed Values)	Region	Product
Dependencies tab	None	

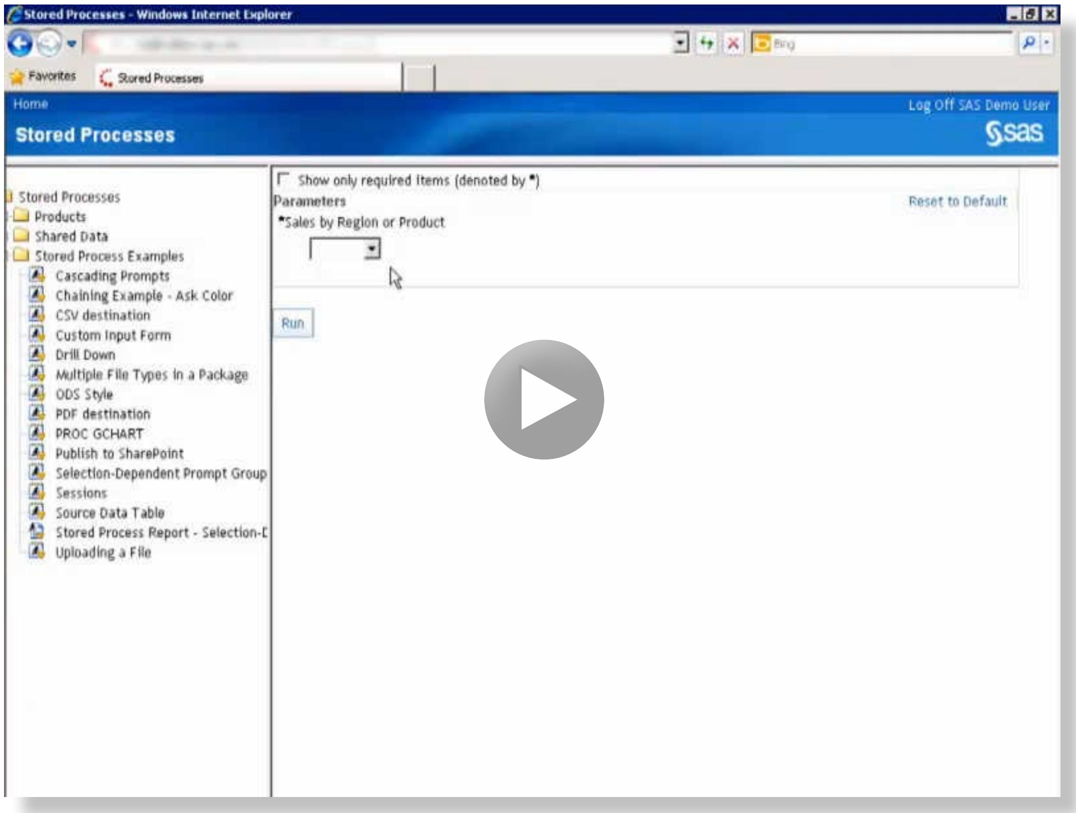


Displayed Text	Name	Type
Parameters		Standard group
Sales by Region or Product	REGPROD	Selection group
Region		Selection-dependent ...
Select a region	region	Text
Product		Selection-dependent ...
Select a product	product	Text

The code for this stored process is as follows:

```
%macro plot;

* Declare local variables;

%local GRAPHVAR TITLE_TEXT WHERE_CLAUSE;

%if (%upcase(&REGPROD) eq REG) %then %do;
   %let GRAPHVAR=product;
   %let WHERE_CLAUSE=%str(region eq "&REGION");
   %let TITLE_TEXT=Total Sales by Product for &REGION;
%end;
%else %do;
```

```
   %let GRAPHVAR=region;
   %let WHERE_CLAUSE=%str(product eq "&PRODUCT");
   %let TITLE_TEXT=Total &PRODUCT Sales by Region;
%end;

proc gchart data=sashelp.shoes;
   where &WHERE_CLAUSE;
   hbar &GRAPHVAR / sumvar=sales
                    clipref
                    frame
                    nostats
                    type=sum;
   title "&TITLE_TEXT";
run; quit;

%mend plot;

* Declare input parameters;

%global PRODUCT REGION REGPROD;

* Create the chart;

%STPBEGIN;
   %PLOT
%STPEND;
```



2.10 Publishing to Microsoft SharePoint

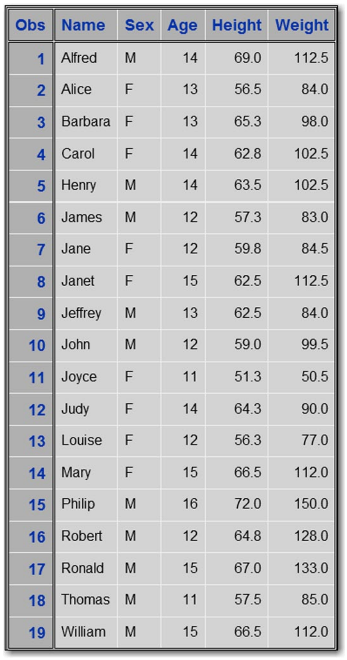
You can use reserved macro variables with %STPBEGIN and %STPEND to publish stored process results to a variety of transports, including archives, e-mail, SharePoint, queues, subscribers, and WEBDAV. Specify the reserved macro variables that are appropriate for the transport that you want to publish to (see the section on using the %STPBEGIN and %STPEND macros in the stored process documentation), and then register the stored process with no result capabilities selected. You also need to set the following input parameters in the metadata:

Parameter Name	Default Value
_RESULT	**PACKAGE_TO_SHAREPOINT**
_SITE_URL	***http://yourSharePointServer***
_LIST_NAME	**Shared Documents**
_COLLECTION_FOLDER	**Stored Process Results**
_HTTP_USER	***userID***
_HTTP_PASSWORD	***Password***
_STATUS_MESSAGE	**<h1>Your request completed.</h1>**

The _STATUS_MESSAGE parameter is not required to publish to SharePoint – it simply displays the status message in the browser after the stored process has executed. The following code is the stored process source code for this example:

```
*ProcessBody;
%STPBEGIN;
   proc print data=sashelp.class;
   run;
%STPEND;
```

When this stored process is executed from the SAS Stored Process Web Application, the client receives a message that says, "Your request completed." The main.html file is published to the **Stored Process Results** subfolder (which is in the **Shared Documents** folder), and the file contains the results of the SAS program. If you open main.html, a printout of the Sashelp.Class data set appears:



Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	56.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
18	Thomas	M	11	57.5	85.0
19	William	M	15	66.5	112.0

Note that before a SharePoint site is used for the first time, it must be initialized with SharePoint content types and column metadata defined by SAS. For more information about initializing the SharePoint site and publishing to different transports, see the *SAS Publishing Framework: Developer's Guide*.

2.11 Using Stored Process Server Functions

Stored process server functions are DATA step functions that you use to define character and numeric strings to generate output in the desired format. The following list of SAS Stored Process Server functions can be used in %LET statements or DATA steps at the top of your stored process code:

- STPSRVGETC, which returns the character value of a server property (for example, the output encoding or version).
- STPSRVGETN, which returns the numeric value of a server property (for example, the session timeout or maximum concurrent requests).
- STPSRVSET, which sets the value of a server property (such as the session timeout or program error).
- STPSRV_HEADER, which adds or modifies a header.
- STPSRV_SESSION, which creates or deletes a session or sets the session timeout value.
- STPSRV_UNQUOTE2, which unmasks quotation marks in an input parameter.

For example, to increase the session timeout (the default is 900 seconds, or 15 minutes), you can include the following line of code:

```
%let rc=%sysfunc(stpsrvset(session timeout,2700));
```

This line of code uses the STPSRVSET function to increase the session timeout to 2,700 seconds, or 45 minutes. Another way to do this is to include the following DATA step at the top of your stored process code:

```
data _null_;
   rc=stpsrvset('session timeout',2700);
run;
```

2.12 Using the STP Procedure

Starting with SAS 9.3, you can use the STP procedure to execute a stored process from a SAS program. PROC STP can be executed in an interactive, batch, or server SAS session. It can also be executed by another stored process.

The following example subsets a data set and then prints it. The original data set, the output data set, the filter conditions, and an output parameter that prints the subset size are all specified by the statements in the PROC STP program, which provide values for the macros in the stored process code. To create the stored process that the PROC STP code runs, register a stored process and name it **PROC STP**. Use the following code for the stored process:

```
%STPBEGIN;

data &_target_OUTSUBSET (drop=nobs);
   set &_source_INDS (where=(%unquote(&filter))) end=last;
   nobs+1;
   if ( last ) then
      call symput('subset_sz', nobs);
run;

proc print data=&_target_OUTSUBSET;
run;

%STPEND;
```

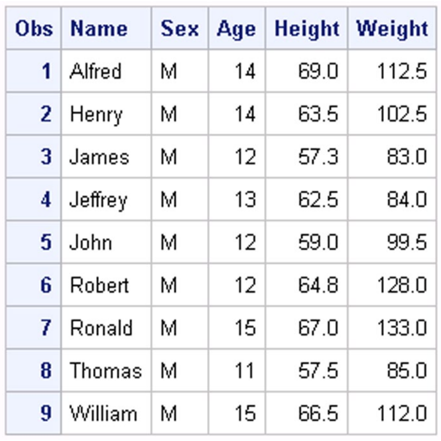
The following code uses PROC STP to run the stored process that you just created. Note that before running PROC STP, you must specify some system options to connect to the metadata server. (These system options are not needed if the PROC STP code is also being executed as a stored process and both stored processes access the same metadata server.)

```
options metaserver = 'your-metadata-server'
        metaport   =  your-metadata-server-port
        metauser   = 'your-userid'
        metapass   = 'your-password';

proc stp program="/Stored Process Examples/PROC STP" odsout=replay;
   inputdata     INDS      = sashelp.class;
   outputdata    OUTSUBSET = work.subset_class;
   inputparam    FILTER    = "(age<=15) and (sex='M')";
   outputparam   SUBSET_SZ = subset_n;
run;

%put SUBSET SIZE is &subset_n;
```

The **ODSOUT=REPLAY** option specifies that the ODS output appears in the Output window of the SAS windowing environment. The following table show the names and values for each of the PROC STP statements and the macro variables that are used in the stored process code to call these values. PROC STP prepends the INPUTDATA and OUTPUTDATA names with _SOURCE_ and _TARGET_, respectively.

PROC STP Statement	Name	Value	Macro Variable in Stored Process	Result
INPUTDATA	INDS	Sashelp.Class	_SOURCE_INDS	Sashelp.Class is used as the input data set for the stored process.
OUTPUTDATA	OUTSUBSET	Work.Subset_ Class	_TARGET_ OUTSUBSET	A new data set, Work.Subset_ Class, is created, which is a subset of the input data set.
INPUTPARAM	FILTER	`"(age<=15) and (sex='M')"`	FILTER	This input parameter is used to filter and subset the data, creating an output data set that contains males who are 15 years old or younger.
OUTPUTPARAM	SUBSET_SZ	SUBSET_N	SUBSET_SZ	The number of observations (the subset size) is output to the log.

Running this code from the SAS windowing environment yields the following results:



Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Henry	M	14	63.5	102.5
3	James	M	12	57.3	83.0
4	Jeffrey	M	13	62.5	84.0
5	John	M	12	59.0	99.5
6	Robert	M	12	64.8	128.0
7	Ronald	M	15	67.0	133.0
8	Thomas	M	11	57.5	85.0
9	William	M	15	66.5	112.0

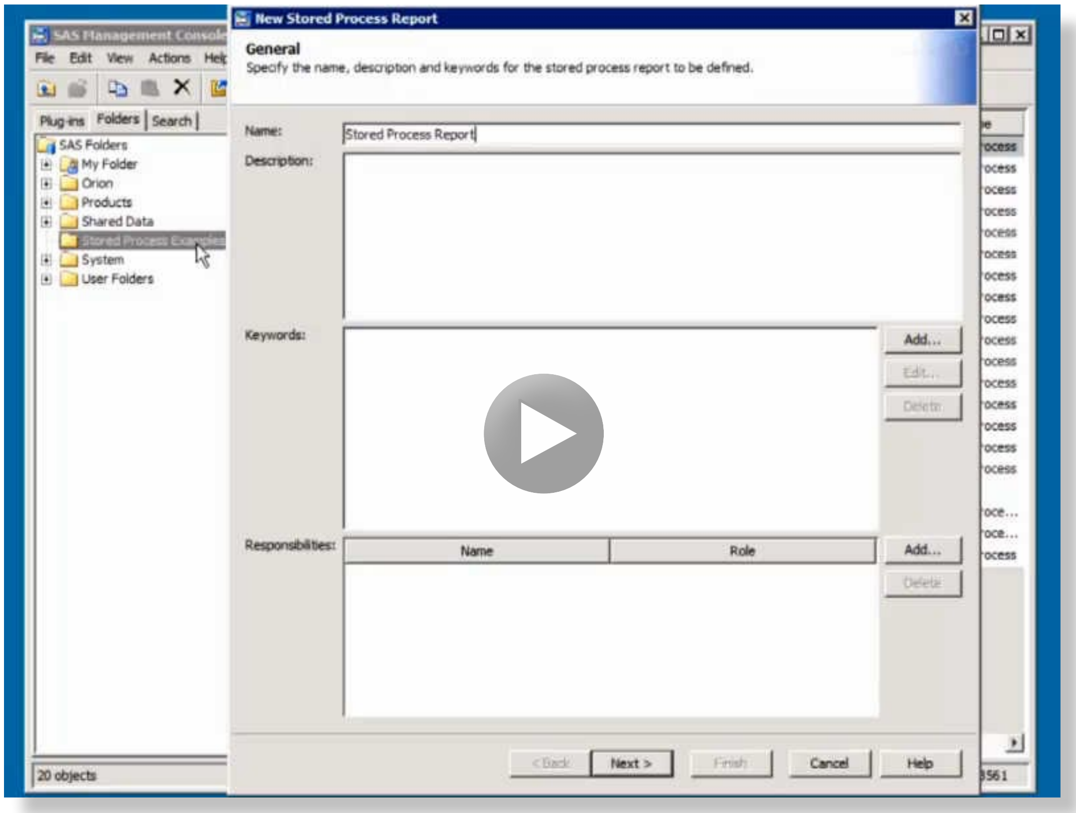
For more information about PROC STP and the various statements that you can use with this procedure, see the *SAS Stored Processes: Developer's Guide*.

2.13 Using Stored Process Reports

A *stored process report* consists of stored process output that is cached. You can use a stored process report to maintain access to stored process results without having to rerun the stored process (which is helpful, especially when a stored process might take a long time to run).

To create a stored process report, use the New Stored Process Report wizard in SAS Management Console. The following example bases a stored process report on the **`Selection-Dependent Prompt Group`** stored process. (Make sure that the stored process that you select produces streaming output.) When you select a stored process that contains prompts, you must also specify values for any required prompts. The prompt values that you select are used every time the stored process runs for this report, unless someone changes the prompt value in the Stored Process Report Properties dialog box. In this example, select **Region**, and then select **United States**.

For this example, leave the maximum retained generations setting at 1 (you can read more about this setting and others in the *SAS Stored Processes: Developer's Guide* or in the product Help). Let's specify a weekly expiration policy of 8 AM EST on Mondays. This means that if a client accesses the stored process report before it expires, say on Friday afternoon, the cached stored process results appear in the client. If the client accesses the stored process report on Monday at 8:01 (in other words, if the client is the first to access the stored process report after the last one expired), then the stored process re-executes and a new output generation is created for the stored process report. (Because you maintained the default of one generation, the old output generation is replaced.)

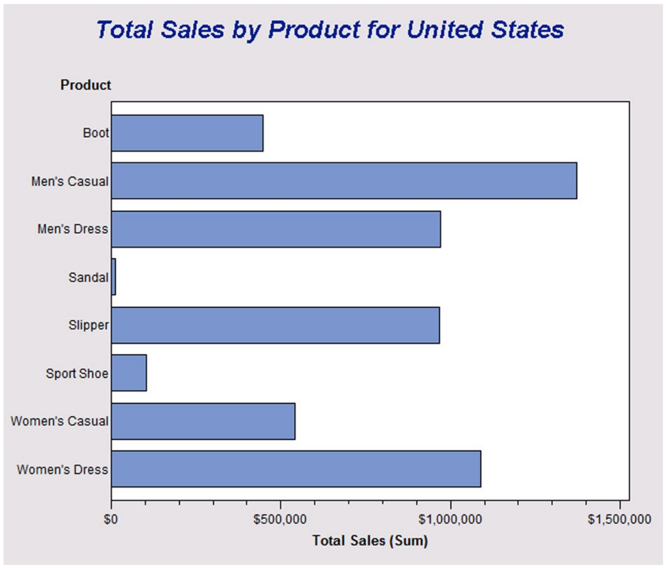

New Stored Process Report

General

Specify the name, description and keywords for the stored process report to be defined.

Name: Stored Process Report

Accessing this stored process report on the SAS Stored Process Web Application yields the following result:



2.14 Using Web Services

Starting with SAS 9.3, you can invoke any SAS Stored Process as a SAS BI Web Service over SOAP or RESTful HTTP. Every stored process is available for execution as a web service simply by accessing the endpoint for the web service from an HTTP or SOAP client. For example, you can access the following URL to execute the ODS Style stored process example as a RESTful web service and retrieve the contents of the _WEBOUT stream:

```
http://yourserver.com:8080/SASBIWS/rest/storedProcesses/Stored%20Process%20
Examples/ODS%20Style/streams/_WEBOUT
```

The results are the same as when the stored process runs using the web application. You can use web services if you are writing your own code client and want to access stored processes. For more information about SAS BI Web Services, see the *SAS BI Web Services: Developer's Guide*.

3

Client Examples

3.1 About the Clients used in this Book

Aside from the SAS Stored Process Web Application, stored processes can be used with a variety of clients, including but not limited to the following (listed here with the versions used in this book):

- SAS Add-In 5.1 for Microsoft Office
- SAS Information Delivery Portal 4.31
- SAS Information Map Studio 4.32
- JMP Pro 10
- SAS Web Report Studio 4.31
- SAS BI Dashboard 4.31
- SAS Enterprise Guide 5.1
- SAS Data Integration Studio

This chapter does not address SAS Enterprise Guide (there is a video in the first chapter that shows how to create stored processes using SAS Enterprise Guide). SAS Data Integration Studio is not addressed here, either. Creating a stored process with this product is exactly the same as creating a stored process using SAS Management Console.

The following table shows which clients can be used to create stored process code, register stored process metadata, and execute stored processes:

	Stored Processes		
Client Applications	**Create**	**Register**	**Execute**
SAS Add-In for Microsoft Office	N	N	Y
SAS Information Delivery Portal	N	N	Y
SAS Information Map Studio	N	N	Y
JMP	Y	N	Y
SAS Web Report Studio	N	N	Y
SAS BI Dashboard	N	N	Y
SAS Enterprise Guide	Y	Y	Y
SAS Data Integration Studio	Y	Y	N

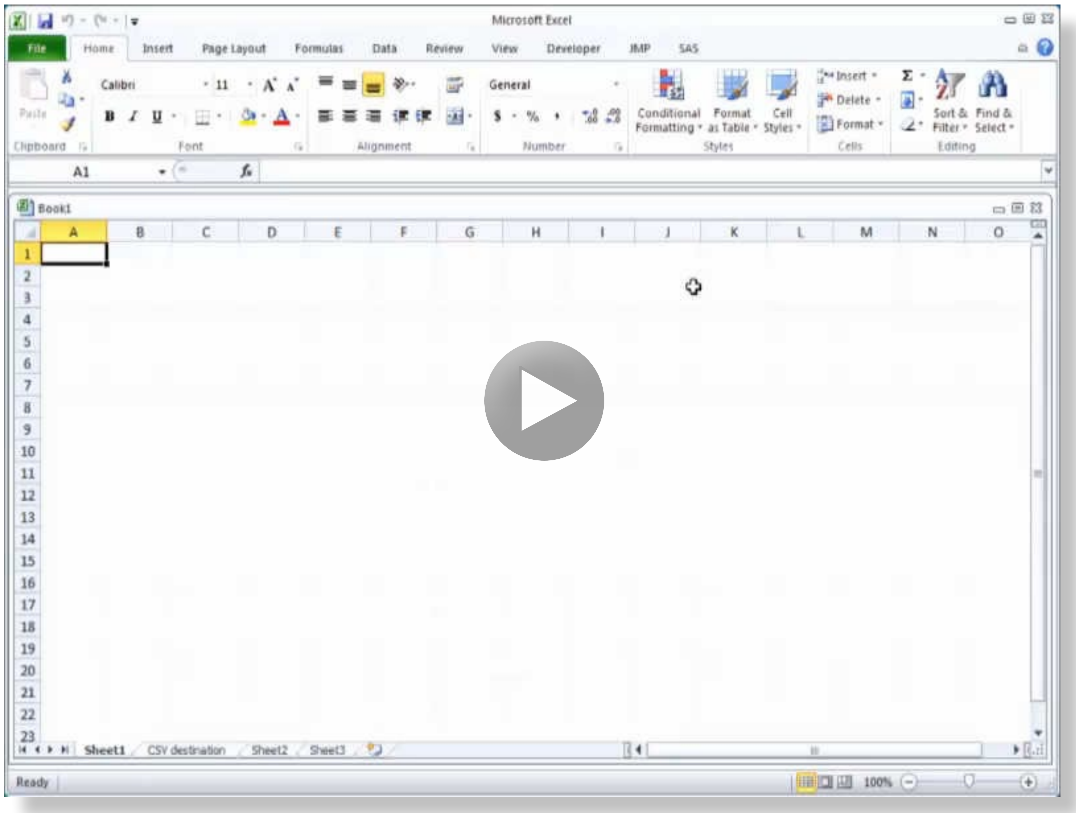
The information in this chapter is meant only as an introduction to the way that each client works with stored processes. Refer to the product Help and documentation for each client for more information.

3.2 SAS Add-In for Microsoft Office

The SAS Add-In for Microsoft Office enables you to include stored process results in spreadsheets, documents, presentations, and e-mails. To run a stored process in a Microsoft Office application, click the **SAS** tab and then click **Reports**. Navigate to the stored process that you want to run and open it. The stored process results appear.

The stored process results are static – they are not updated automatically. If you want to refresh the results later (for example, to include current data in your presentation), then click the **Manage Content** button.

From the Manage Content window, you can perform a variety of tasks, such as deleting SAS content, refreshing analyses and data sources, and sending the results from an analysis to another Microsoft Office application. All of the SAS content in your current document is listed in the Manage Content window.



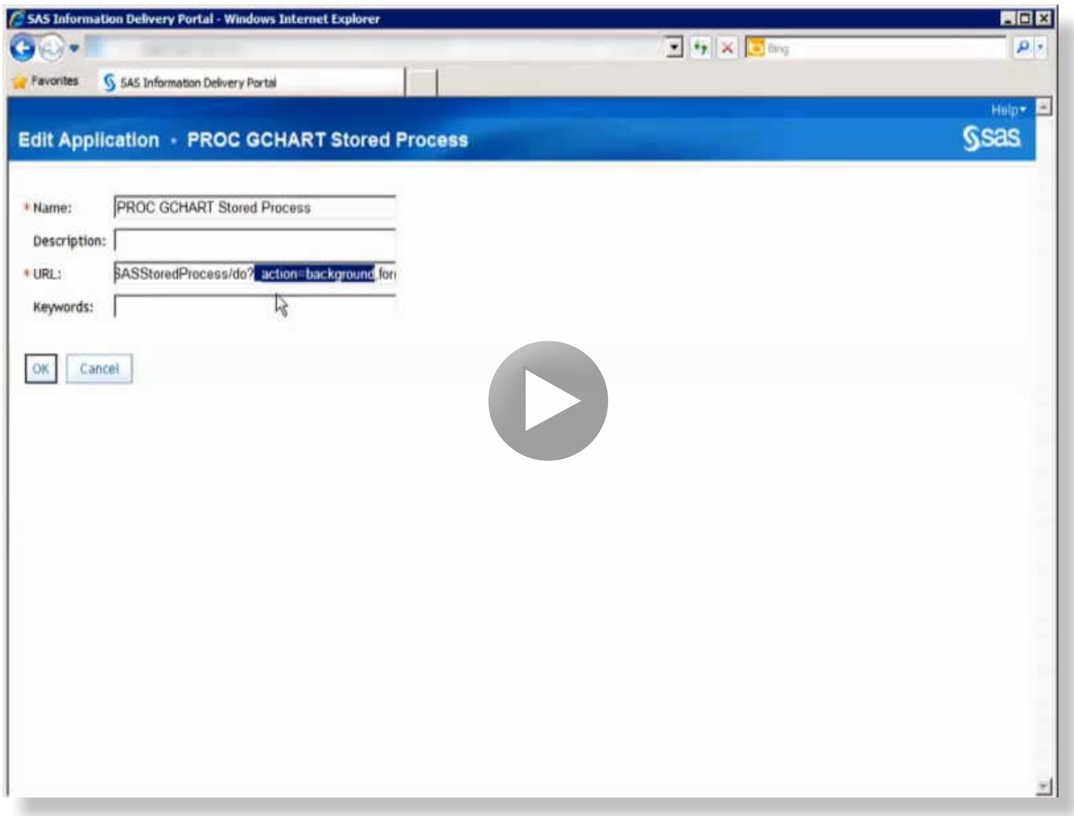
You can also use the Scheduled Tasks functionality in the Windows operating environment to schedule when the SAS content in a workbook, document, or presentation is automatically refreshed. See the SAS Add-In for Microsoft Office product Help for more information.

3.3 SAS Information Delivery Portal

In the SAS Information Delivery Portal, you can execute a stored process from a collection portlet or a navigator portlet that contains a stored process. You can also use the search tool to find the stored process that you want. Streaming output from a stored process appears directly in the portal. If a stored process produces package output, then a message appears in your Stored Process Alerts portlet once the stored process has executed (if the stored process was set up to run in the background). You can click the alert message to see the results. From the viewer, you can bookmark, e-mail, or publish the stored process.

Stored process alerts notify you when certain types of SAS Stored Processes have finished executing and the results are ready to view. These alerts are displayed in the Stored Process Alerts portlet on one of your personal portal pages. You can then click the alert message to see the results of the stored process. To run a stored process in the background, make sure that it produces package output, and specify `_ACTION=BACKGROUND` in the URL for the stored process.

The following video shows an example of a stored process called from a collection portlet and run in the background:


SAS Information Delivery Portal - Windows Internet Explorer
Favorites
SAS Information Delivery Portal
Help
Edit Application • PROC GCHART Stored Process
OK
Cancel

Name: PROC GCHART Stored Process

Description:

URL: SASStoredProcess/do?_action=background,for

Keywords:

3.4 SAS Information Map Studio

You can add a SAS Stored Process to any relational information map. The stored process can be used to generate or refresh data for a query, acting as a dynamic data source for the information map. When an information map with a stored process is used in a query, the stored process runs before the query executes. You can add only one stored process to an information map. Before you can add a stored process to an information map, you must first add at least one table as a data source for the information map.

Ensure that each table that your stored process writes to is defined in the metadata repository. A table and its library must be defined in metadata before the table can be used as a data source for an information map. Ensure that the data source that you specify for the information map is the table that your stored process writes to. For example, if your stored process output is written to the RESULT_SET table in the Stpout library, then you must use that table as the data source for your information map.

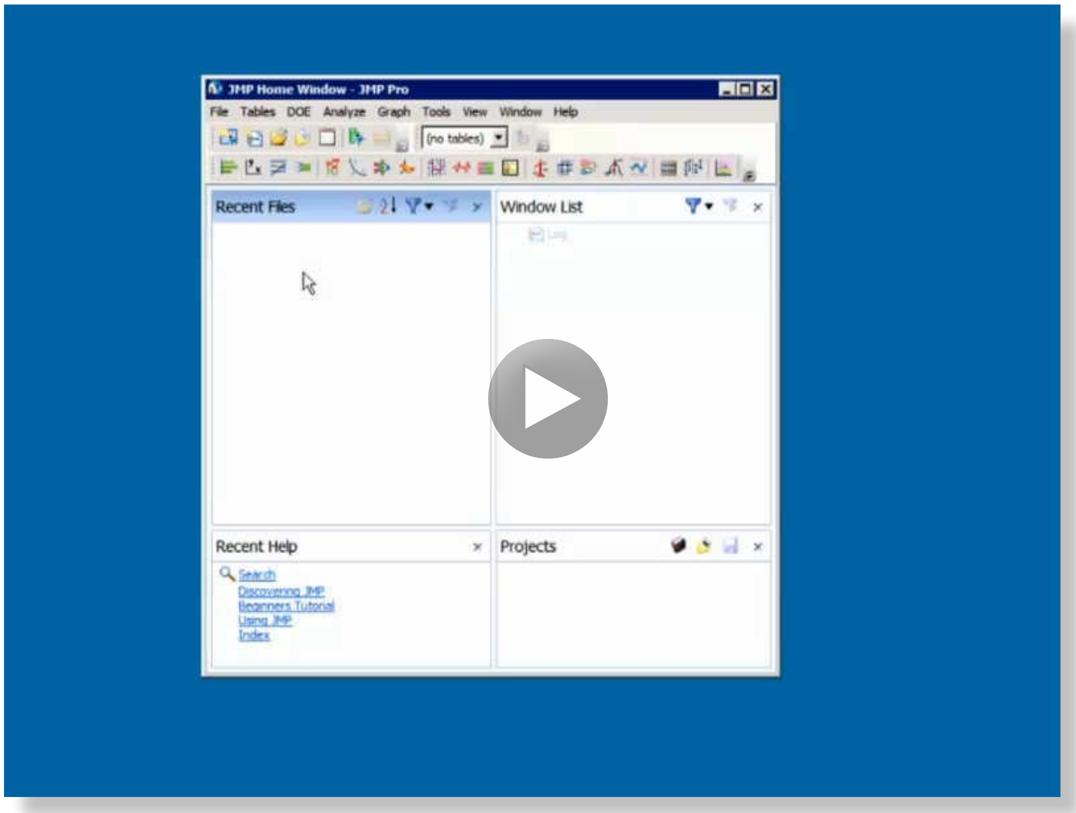
3.5 JMP

You can use JMP to run stored processes and display results. These results can include SAS output and graphics in a variety of formats (for example, PDF, HTML, and TEXT) and SAS data sets.

To run a stored process using JMP, you must first connect to a metadata server. After you have connected to the metadata server, you can browse through the SAS folders to find the stored process that you want to run.

You can enhance a stored process with JMP functionality by adding data sets and JSL scripts to the results package for the stored process. This enhanced functionality is available only when you run the stored process in JMP; other clients ignore these additions. You can use the JMP Stored Process Packager in SAS Enterprise Guide to add these items to a stored process results package. The SAS program that is generated by the JMP Stored Process Packager is not run as part of the SAS Enterprise Guide project.

You can also use the SAS Program Editor in JMP to create code to use as a stored process. However, you cannot use JMP to register the metadata for a stored process.



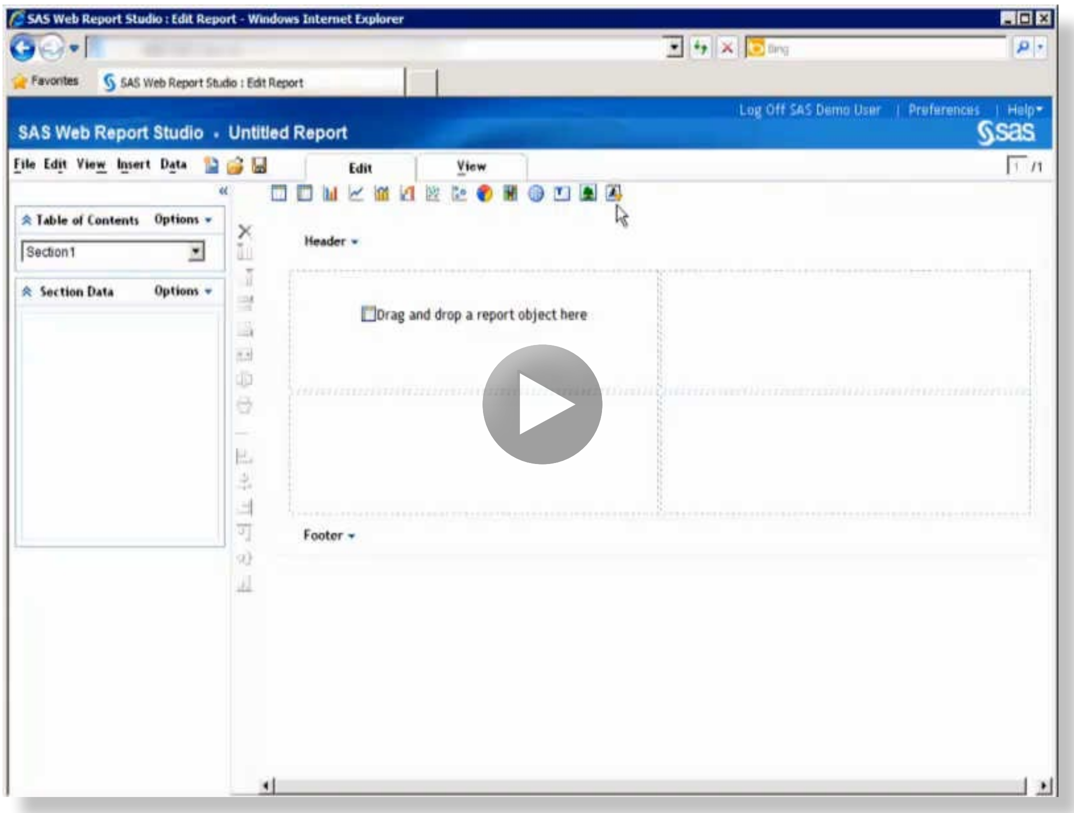
3.6 SAS Web Report Studio

Stored process output in a report can include queries, prompted filters, titles, images, and statistical analyses. Some stored process output might support chart tips, tooltips, and drilling.

There are two ways to use stored processes in reports:

- Stored processes can be included as part of a larger saved report. When you include a stored process in a saved report, you can perform some additional layout design tasks that are independent of the stored process output. For example, you can add text, images, headers, and footers. In the same report section, you can also include the output of a query that uses data items from a data source. If you add an additional data source (other than the stored process), then you can also add graphs, tables, and, possibly, geographical maps.
- Stored processes can be rendered directly. When a stored process is viewed directly in SAS Web Report Studio, it is a stored process report (note that this is different from the type of stored process report that is created in SAS Management Console). Stored process reports contain only one section and are automatically refreshed. All users can run a shared stored process.

The following video shows both methods of using stored processes in reports:


SAS Web Report Studio : Edit Report - Windows Internet Explorer
Favorites
SAS Web Report Studio : Edit Report
Log Off SAS Demo User | Preferences | Help
SAS Web Report Studio • Untitled Report
File Edit View Insert Data
Edit
View
Table of Contents Options
Section1
Section Data Options
Header
Drag and drop a report object here
Footer

3.7 SAS BI Dashboard

SAS BI Dashboard can use stored processes in two ways – either within a custom graph indicator or as a data source for an indicator. When you are using stored processes with SAS BI Dashboard, remember not to include the %STPBEGIN and %STPEND macros. Also, the stored process should not produce HTML output. SAS BI Dashboard does not support the display of HTML content.

You can create a custom graph indicator that uses a stored process. By using a stored process, you can create dashboards that display images created by using SAS/GRAPH software. The stored process must

- create an image file in one of the following formats: PNG, JPG, or GIF (but not animated GIF)
- use streaming output
- write directly to _WEBOUT

The stored process that you want to use must include GOPTIONS statements to write an image to _WEBOUT. Add the following code at the beginning of the SAS/GRAPH code that creates the image:

```
goptions gsfname=_webout gsfmode=replace;
goptions device=png;
```

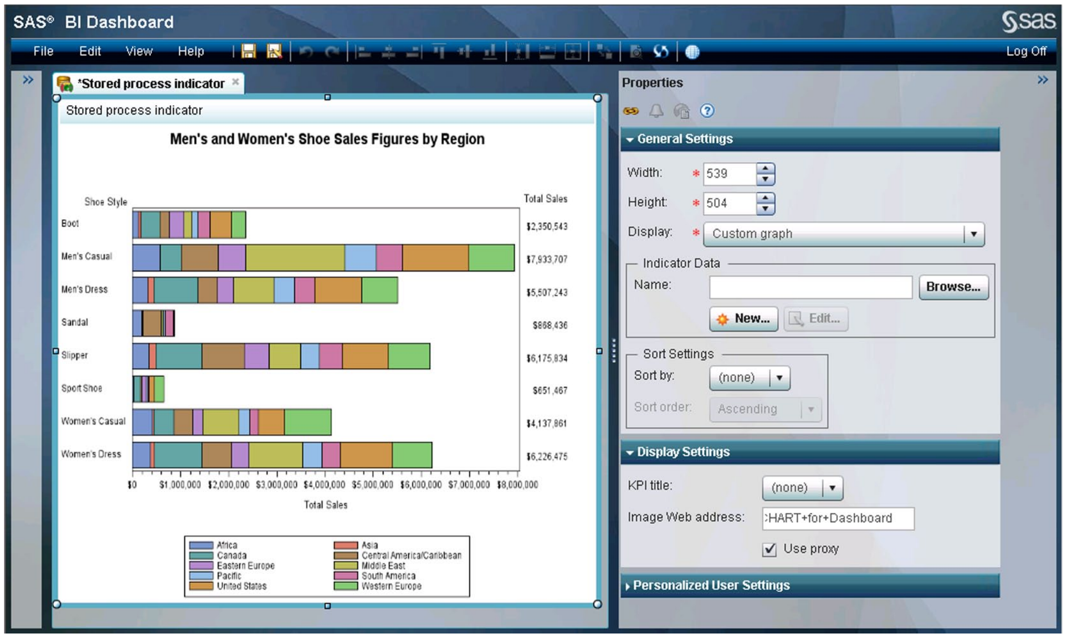
The following code shows the GCHART procedure example that you created earlier, modified for use in SAS BI Dashboard:

```
legend1 label=none across=2 frame;

title "Men's and Women's Shoe Sales Figures by Region";
footnote;

goptions gsfname=_webout gsfmode=replace;
goptions device=png;

proc gchart data=sashelp.shoes;
   hbar product / sumvar=sales
                  subgroup=region
                  sum
                  patternid=subgroup
                  legend=legend1;
   label product='Shoe Style';
run;
quit;
```

Run the stored process using the SAS Stored Process Web Application to make sure the output file is created.

In the SAS BI Dashboard designer, create a new custom graph indicator. In the Display Settings section of the Properties pane, in the **`Image Web address`** field, paste the stored process web address (copy the URL from the SAS Stored Process Web Application). Note that this field does not support relative pathnames. Save the custom graph indicator.

The following screen shows the new stored process indicator:



Width: 539
Height: 504
Display: Custom graph
Name:
Sort by: (none)
Sort order: Ascending
KPI title: (none)
Image Web address: :HART+for+Dashboard
✓ Use proxy

You can create a SAS stored process that you can use as a source of indicator data in SAS BI Dashboard. These stored processes do not produce visual output. The stored process must

- create a SAS data set in the Work library
- publish the data to a SAS package file using the SAS Publishing Framework
- set the macro variable _ARCHIVE_FULLPATH to the path of the archive file that the stored process generates