Paper 024-2008

Adapting Your Programs to the SAS®9 Paradigm

Cynthia L. Zender, SAS Institute, Inc., Cary, NC

ABSTRACT

Do you want to convert existing SAS programs into SAS® Stored Processes? Would you like to learn how to convert and adapt your existing SAS programs so that they can be executed from selected SAS® Business Intelligence client applications? Topics in this paper include basic SAS program conversion to a SAS stored process, conversion of SAS/GRAPH® programs, use of macro variables in program conversion, streaming versus transient package output from stored processes, and permanent result packages.

INTRODUCTION

According to the dictionary, the word “paradigm” was first defined in the fifteenth century as an example or pattern. The purpose of this paper is to provide an example or pattern for you to follow to adapt your programs to work within the context of the SAS® Enterprise Intelligence Platform as SAS Stored Processes. For the sake of brevity and ease of explanation, the ODS options and user-supplied parameters are as simple as possible. Data from the Sashelp library has been used for the demonstration programs.

Security and authorization discussions fall outside the scope of this paper. Not every stored process author has the ability to alter authorization permissions or to move stored processes in a DEV/TEST/PROD environment. The paper does not discuss using the SAS® Enterprise Guide® Stored Process Wizard. The main focus of this paper is on code changes that must be implemented to convert existing SAS programs into SAS Stored Processes.

OVERVIEW OF THE SAS® ENTERPRISE INTELLIGENCE PLATFORM

Stored processes operate within the context of the SAS Enterprise Intelligence Platform. When we talk about adapting programs to the SAS 9 paradigm, we first have to understand what the new paradigm is. Then, we can discuss what a SAS stored process is. And finally, we need to see a concrete example of how you would convert an existing SAS macro program (one that you might run in a production environment) into a SAS stored process.

Let’s look at how business intelligence has changed over the years. In 1907, when my great-grandfather was running a hardware store, he had a bell on the door. His idea of business intelligence was to write in his ledger book how many customers came in the door each day. He updated the inventory by hand at the end of each day based on that day’s receipts. This is the same way that my grandfather ran the hardware store. Both my grandfather and great-grandfather had his own way of gathering information about the business, and of acting on the information or predicting future business based on the trends that he saw.

Computers and computer technology have changed the nature of business intelligence (BI). Companies talk about their flavor of BI, knowledge management, data warehousing, and data mining to track and analyze business data and to make decisions based on that data. And, all of those terms **are part of BI**. The SAS Business Intelligence Architecture, as shown in Figure 1, captures the essence of the SAS Enterprise Intelligence Platform. The client applications allow users to access and analyze the data from the client tier. The client applications use the servers, data, and programs in the other three tiers as defined in the centralized metadata repositories.

SAS Business Intelligence Architecture includes client applications, such as SAS® Add-In for Microsoft Office or SAS® Data Integration Studio, SAS server processes, and the metadata repository. Generally, this architecture is installed on multiple machines.

But, when SAS® Web Report Studio users, Microsoft® Office users, or users of SAS Stored Processes access SAS data, stored processes, or information maps, they are using applications that live on the client-side of the architecture. Through client-side applications, users communicate with the server and get the task they need performed or the question they have answered. Users are aware that work is being done, but they are not concerned with *how* the work is being done. One of the ways work gets performed is with SAS Stored Processes. The stored process is one of the essential building blocks of the SAS Enterprise Intelligence Platform. The new SAS 9 paradigm involves adapting your legacy programs or writing new programs so that they can be submitted from multiple client applications.

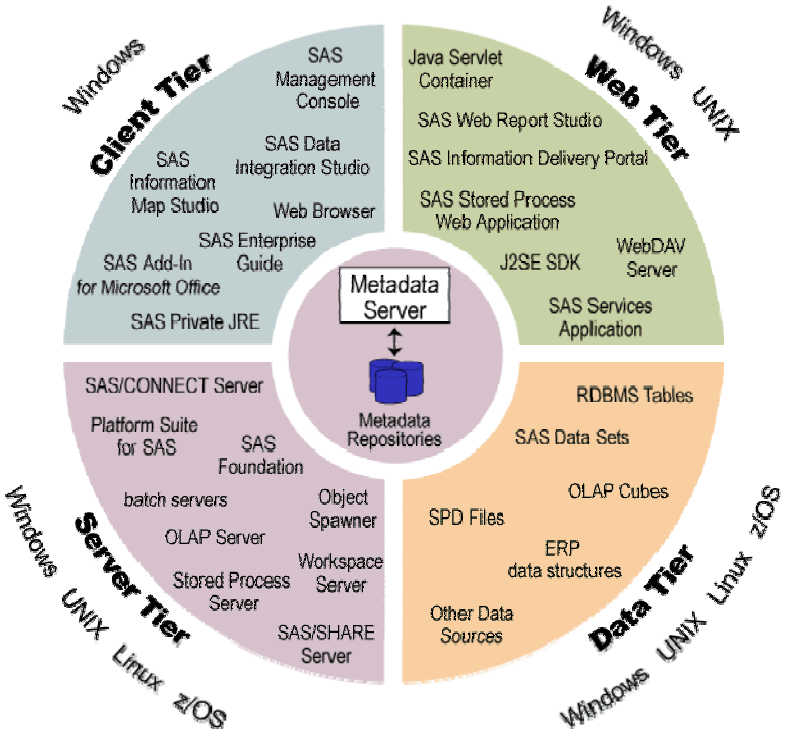

Figure 1: The SAS® Enterprise Intelligence Platform

SAS® STORED PROCESSES

A SAS stored process is just a SAS program stored in a special place called a source code repository. Information about the stored process—such as which server to run the stored process on, what parameters the stored process has, what the name of the .sas program is—is registered in the metadata repository.

A stored process can access any SAS data source or external file, as long as the data source or external file is accessible to the server where the stored process is executed. This includes all of your RDBMS and legacy files. A stored process can create many types of output; the example in this paper focuses on returning report results to the requesting client application. And, the focus is on converting a program that produces graphical and tabular results, such as the kind of output you get from ODS.

The example PROC REPORT and PROC GCHART program can easily be turned into a stored process. It really doesn't matter what the code is, as long as it doesn't open an interactive window (such as the VAR window, a KEYS window, or a BUILD window), you can change the program to be a stored process.

Consider a basic SAS program with a PROC REPORT step followed by a PROC GCHART step, all wrapped in an ODS HTML "sandwich." When you convert this basic program to a SAS stored process, it is likely that only the "wrapper" code or ODS HTML sandwich will have to change to convert the program. In our example, the ODS HTML invocation statement is replaced by the %STPBEGIN macro invocation. The ODS HTML CLOSE statement is replaced by the %STPEND macro invocation. Other changes that might need to be made will be addressed later in a more detailed example.

Basic SAS Program Example	Converted Stored Process Example

`ods html file='original.html';`
 	
`** PROC REPORT step;`
`** PROC GCHART step;`
 	
`ods html close;`
	`*ProcessBody;`
	`%stpbegin;`
	
	`** PROC REPORT step;`
	`** PROC GCHART step;`
	
	`%stpend;`

One of the hallmarks of a SAS stored process is flexibility. For example, if there is a single stored process, each of three different analysts could execute that same stored process. Each analyst can request that the stored process use a different region.

Figure 2: Execution of a Single Stored Process by Three Analysts

Each analyst can supply a different input parameter to the same stored process program that causes it to return customized results to each of them. In addition, each analyst can be working in a different client application. The analyst for Canada might use SAS Enterprise Guide, the analyst for Pacific might use SAS Add-In for Microsoft Office with Microsoft Excel, and the analyst for Asia might need to receive the results in Microsoft PowerPoint. How do you add that flexibility to your stored process programs? Consider the following updated version of the converted program with an input parameter for region.

Converted Stored Process Example	Stored Process with Input Parameter for REGION
	`%global Reg;`
`*ProcessBody;` `%stpbegin;`	`*ProcessBody;` `%stpbegin;`

`** PROC REPORT step;`	`proc report data=sashelp.shoes nowd;`
	`where region = "&Reg";`
	`. . . more code . . .`
	`run;`

`** PROC GCHART step;`	`proc gchart data=sashelp.shoes;`
	`where region = "&Reg";`
	`. . . more code . . .`
	`run;`

`%stpend;`	`%stpend;`

If you have an input parameter, it comes to SAS on the server as a SAS macro variable. The %GLOBAL statement ensures that the input parameter is in the GLOBAL symbol table when the stored process executes. Then, to make your program more flexible, the only change needed in the preceding code is a WHERE statement to limit the number of rows passed to the stored process program, based on each analyst's selection at the stored process execution time. Because your stored process and parameters are defined in the metadata, the stored process can be invoked

from many of the SAS Enterprise Intelligence Platform applications. Or, you can write your own custom application to execute the stored process using the SAS Stored Process Web Application or using .NET or Java technology.

The basic conversion, which requires changing only the wrapper code, is simple to implement. Many sites with the SAS Enterprise Intelligence Platform already have production programs that use the SAS macro facility, so the detailed program conversion example is a SAS macro program conversion.

BASIC PROGRAM CONVERSION STEPS

The basic steps for program conversion are:

1. Have a working SAS program.
2. Convert the SAS program into a stored process by adding the **`*ProcessBody`** comment and using the %STPBEGIN and %STPEND macro invocations instead of the ODS invocation statements. Convert any macro definitions or add macro conditional logic, if required.
3. Register the stored process metadata relevant to the stored process program.
4. Test your stored process in all client applications from which it will be executed.

Part of the conversion process in step 2 and the registration process in step 3 involves understanding how user-supplied parameter values are treated by stored processes and how ODS options need to be supplied to have an effect on stored process results.

STEP 1: HAVE A WORKING SAS® PROGRAM

The program Case_Study1.SAS contains two PROC REPORT steps and a PROC GCHART step within a SAS macro program. The entire program is shown at the end of this paper.

```
%macro RegRept(Reg=Canada);

%let FReg = %sysfunc(compress(&Reg,/));
%let FReg = %sysfunc(compress(&FReg));
%let Ureg = %sysfunc(upcase(&Reg));

proc sort data=sashelp.shoes out=shoes;
by region subsidiary product;
where upcase(Region) = "&UReg";
run;

ods listing close;
ods html path='c:\temp' (url=none)
         file="&FReg..html" style=sasweb;

** First PROC REPORT step;

** Second PROC REPORT step;

** PROC GCHART step;

ods html close;
options missing=.;
title;
%mend RegRept;
```

The SAS macro program sets Canada as the default region. As with a lot of macro programs, the parameter is used to limit the rows that the programs within the macro program will use. Because the region names contain spaces and punctuation, the &FREG macro variable is used in the ODS HTML statement to set the filename for the created output file. Remember, this is the program that you are starting with and will need to be converted into a SAS stored process.

The program is invoked using the following syntax, and generates a report for each REG= value:

```
%RegRept(Reg=Africa);
%RegRept(Reg=Canada);
%RegRept(Reg=Pacific);
%RegRept(Reg=United States);
%RegRept(Reg=Western Europe);
%RegRept(Reg=Central America/Caribbean);
%RegRept(Reg=Eastern Europe);
%RegRept(Reg=Asia);
```

Based on the SAS code, the first invocation with Africa as the region produces the output shown in figures 3 and 4:

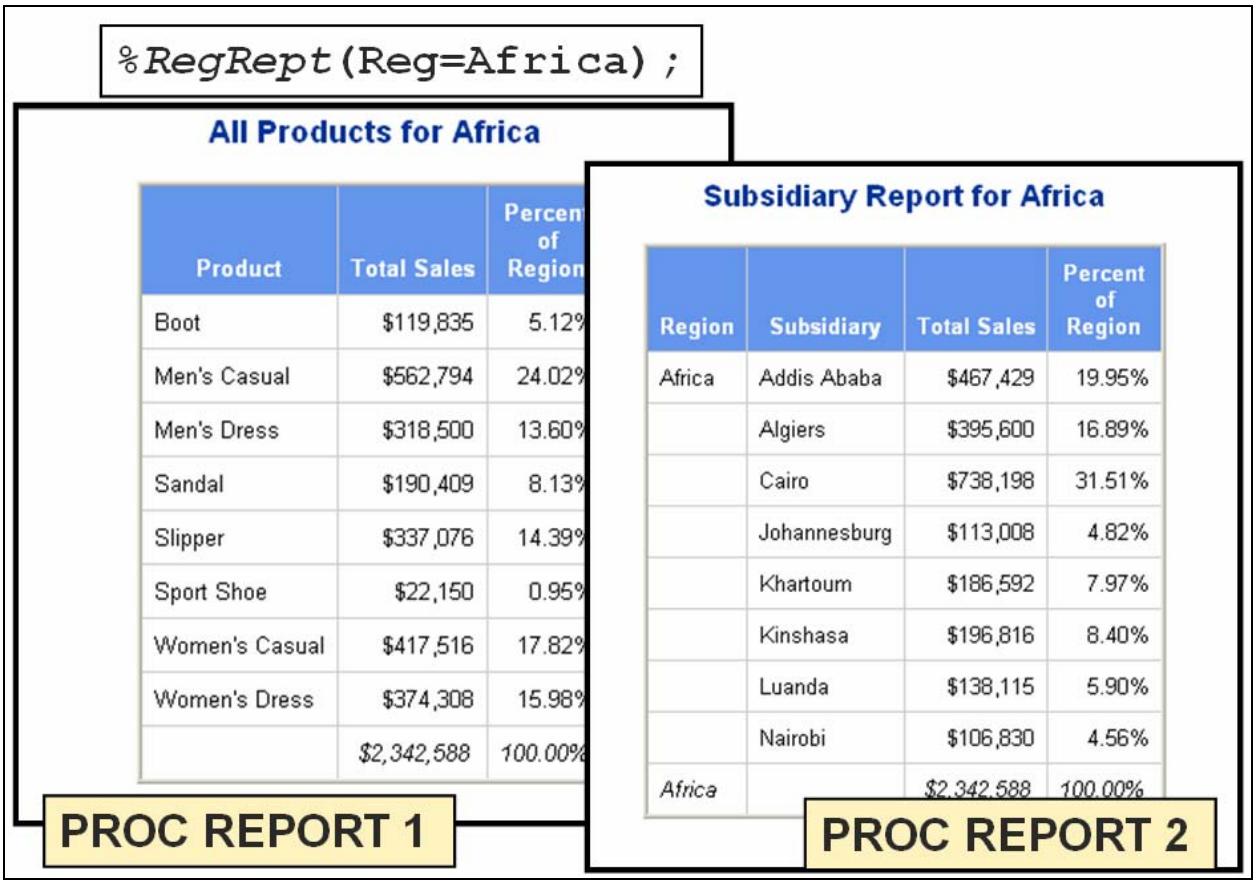

Figure 3: PROC REPORT Output from the %REGREPT Macro Program

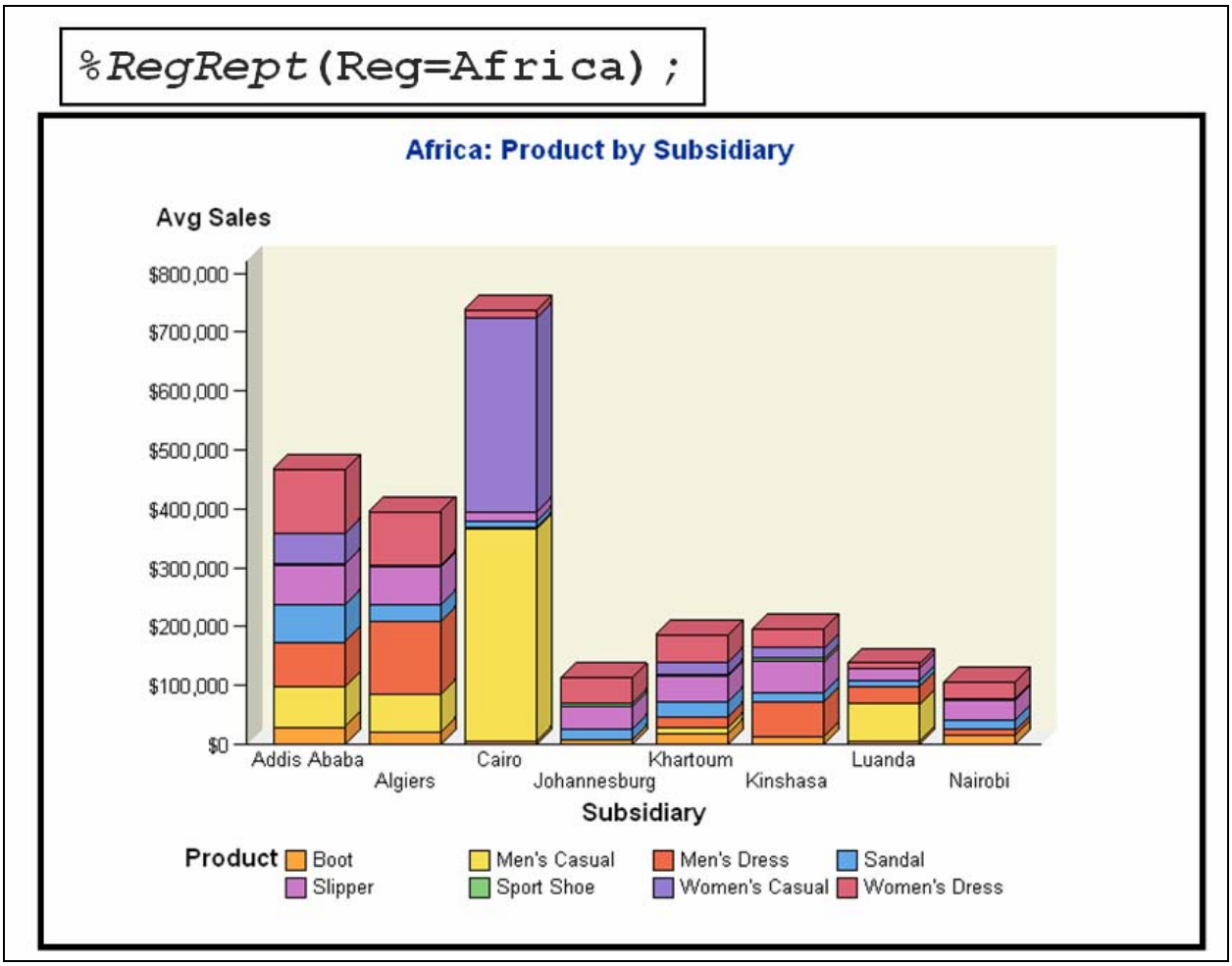

Figure 4: PROC GCHART Output from the %REGREPT Macro Program

STEP 2: CONVERT THE PROGRAM WITH REQUIRED SYNTAX

The SAS macro facility is used in many ways by stored processes. One way is through input parameters. Another way is through the predefined macro programs %STPBEGIN and %STPEND, which become the new wrapper code for the program.

%STPBEGIN AND %STPEND MACRO PROGRAMS

The main part of the program conversion takes place when the %STPBEGIN and %STPEND macro invocation statements replace the ODS invocation statements. %STPBEGIN and %STPEND replace the ODS HTML sandwich. They are a matched set and (in concept) perform the same functions as ODS HTML OPEN and CLOSE statements.

The %STPBEGIN macro initializes ODS to generate output from the stored process. The %STPEND macro ends ODS processing and delivers the results to the client application that executed the stored process. These macro programs use many reserved macro variables to control how they operate. Some variables are available for you to test or even change. Generally, each of the client applications in the SAS Enterprise Intelligence Platform has a preferred ODS destination or result type. HTML is not the default result type for all of the SAS Enterprise Intelligence Platform applications, but, conceptually, the %STPBEGIN call sets up the ODS environment for a client application in much the same way that an ODS HTML invocation statement sets up the ODS environment to create an HTML file.

There is an alternate, more advanced invocation method for stored processes that is generally used with Web-based client applications. This method is similar to the kind of invocation (using FILE=_WEBOUT) that was used with SAS/IntrNet® Application Dispatcher programs. A discussion of this advanced method is outside the scope of this paper.

Although it is not required, the use of the ***ProcessBody** comment is recommended for maximum flexibility. On the SAS Workspace Server, macro variables are not initialized until the ***ProcessBody** comment is encountered. The macro variables used by %STPBEGIN and %STPEND are in the macro global symbol table. The user-defined input parameters need to be in the macro global symbol table. You should use a %GLOBAL statement to put the user-defined input parameters in the table.

PROGRAM CONVERSION

To keep the new stored process simple, there is only one input parameter. This parameter provides a REGION value for the REG macro variable. User-supplied variables like REG could have a hard-coded value in the stored process, could have a value provided using an interface within the client application, or could have a value provided using a custom-coded interface. In the original program code, the value of the REG macro variable was hard-coded with a default in the macro program definition. Versions of the value (UREG and FREG) are assigned values with %LET statements. In the SAS Enterprise Intelligence Platform, the stored process author registers the input parameter in the metadata and can provide valid variable values and a default value. The user is presented with an easy-to-use interface from which he or she can select a valid value.

The advantage of using input parameters is not just the flexibility that enables the stored process author to deliver more generic, reusable programs. The metadata is the “boss” as far as parameter usage is concerned. It can contain information such as whether the parameter is required and modifiable, whether there are parameter value constraints to be enforced by the client interface, and whether the user can make only a single selection for parameter values or multiple selections. Figure 9 shows the prompting interface in SAS Enterprise Guide for the REG parameter. The same interface is used by SAS Add-In for Microsoft Office.

Let’s compare the original program code to the final stored process code. Annotations indicate lines whose changes will be discussed:

Original Case_Study1.SAS Program

```
%macro RegRept(Reg=Canada); (1)

%let FReg = %sysfunc(compress(&Reg,/));
%let FReg = %sysfunc(compress(&FReg)); (2)
%let Ureg = %sysfunc(upcase(&Reg));

proc sort data=sashelp.shoes out=shoes;
by region subsidiary product;
where upcase(Region) = "&UReg";
run;

ods listing close; (3)
ods html path='c:\temp' (url=none)
         file="&FReg..html"
         style=sasweb; (4)

** First PROC REPORT step;

** Second PROC REPORT step;

** PROC GCHART step;

ods html close; (4)
options missing=.;
title;
%mend RegRept;
```

Converted Stored Process SP1.SAS Program

```
%macro RegRept; (1)



%let Ureg = %sysfunc(upcase(&Reg));

proc sort data=sashelp.shoes out=shoes;
by region subsidiary product;
where upcase(Region) = "&UReg";
run;







** First PROC REPORT step;

** Second PROC REPORT step;

** PROC GCHART step;

options missing=.;
title;
%mend RegRept;

*ProcessBody; (5)
%global Reg UReg;
%stpbegin;
%RegRept;
%stpend;
```

1. The macro program definition needs to change. All of the input parameters need to be in the global symbol table. A macro is defined with either a positional or keyword parameter, for example:

   ```
   %macro RegRept(Reg=Canada);
   ```

   The parameters are generally defined in a local symbol table that is unique to the macro program (in this case, REGREPT). Because the stored process input parameters will be put in the global symbol table, if this macro program is left unchanged, it is possible for **`Reg=Canada`** to override the input parameter supplied by the stored process consumer. So, the preferred macro definition is:

   ```
   %macro RegRept;
   ```

   This definition assures that no stored process parameters will be overridden by any default values set at macro definition time.
2. The %LET statements that remove special characters and spaces from the &REG macro variable are no longer needed. So, the two %LET statements are removed from the program.
3. ODS LISTING CLOSE is an extraneous statement that is removed. Even though the stored process is running on a server, the %STPBEGIN and %STPEND macros manage the needed destinations.
4. The SAS Enterprise Intelligence Platform application dynamically requests and dynamically receives the stored process results in the open client application. The user invoking the stored process can save the stored process results using the **`File`** menu of the client application; therefore, the FILE= or BODY= option is not needed for this stored process. The ODS HTML sandwich is removed (which eliminates the need to use the &FREG macro variable).
5. The **`*ProcessBody`** comment starts the stored process. Because the **%REGREPT** macro does not generate output until it is invoked, only the %REGREPT invocation, and not the macro definition section, needs to be within the %STPBEGIN and %STPEND wrapper code. The %GLOBAL statement ensures that both &REG and &UREG are in the global symbol table. As a result, no error messages are issued if the user does not make any choices in the client application, or if the program is invoked through the SAS Stored Process Web Application with no input parameters supplied. If the %REGREPT macro program were stored in the macro autocall library that is defined to the SAS Metadata Server, then the five lines of code above are the only code that would appear in the stored process. If you wanted to override the style to SASWEB, the override would have to be supplied before the %STPBEGIN invocation:

   ```
   *ProcessBody;
   %global Reg UReg;
   %let _odsstyle=sasweb;
   %let _odsstylesheet=;
   %stpbegin;
   %RegRept;
   %stpend;
   ```

OVERRIDING RESERVED MACRO VARIABLES USED FOR ODS OPTIONS

In the preceding example, there are special macro variables that are reserved for use with the %STPBEGIN and %STPEND macro programs. When your stored process is executed by a client application, %STPBEGIN initializes the macro parameters that are appropriate for *that* client application. For example, the default result type or ODS destination for SAS Enterprise Guide is HTML format, and the default result type for SAS Web Report Studio is the SAS report format (XML). The reserved macro variable, _ODSDEST, has the name of the destination. Normally, you do not need to change _ODSDEST if you want your stored process to generate the default result type for the client application, or if you want your users to be able to select a result type using selection methods in the client application. Just like every client application has a default result type or destination, every client application has a default style. To override this default style (if allowed by the client application), either the _ODSSTYLE parameter or the _ODSSTYLESHEET parameter would need to be changed.

You can invoke user-defined macro programs from within your stored process programs to leverage existing macro code. However, changes might need to be made to your existing macro programs. The macro variable that will become the user-supplied parameter is defined in the original program as a keyword parameter, so let's deal with that parameter first.

You might decide not to override the style to SASWEB, because each client interface provides a way for the stored process consumer to select a style for the output. If you do override the style to SASWEB, then, in addition to overriding the special macro variable &_ODSSTYLE, you should turn off the special macro variable &_ODSSTYLESHEET. As a result, client applications that can use either a style template or a cascading style sheet file use only the style template information. That is the purpose of the null %LET statement for &_ODSSTYLESHEET.

PROGRAM HOUSEKEEPING ISSUES AND OTHER INFORMATION

In addition to the &REG macro variable, you can do more with input parameters, such as providing lists of variables to be analyzed, specifying the type of analysis or statistics to be used, and specifying presentation options for the report. The possibilities are limited only by your imagination (with a few server-specific limitations). No matter what you do with the input parameters, they must follow the same rules in your program as those for SAS macro variables:

1. Macro variables can be accessed through normal macro syntax (&MACVAR) or through functions like SYMGET, SYMGETC, or SYMGETN. Macro variables can be assigned values using CALL SYMPUT or %LET, or through PROC SQL, in addition to having values collected and set via the client application interface.
2. Parameter names can be no longer than 32 characters. They must start with an alphabetic character or underscore and can contain only alphanumeric characters or underscores.

For more information about server-specific rules related to input parameters, refer to “Input Parameters” in the *SAS® 9.1.3 Integration Technologies Developer's Guide*.

You must consider housekeeping issues. This program used the Sashelp data library. However, your stored process might contain a LIBNAME statement, such as the following, which points to data on a network drive:

```
libname wombat 'h:\user\wombatproj\data';
```

Because the stored process works within the context of the SAS Enterprise Intelligence Platform, you will want to use a LIBNAME statement that enables you to take advantage of the security that is part of the metadata for the users, data files, and stored processes. Therefore, the previous LIBNAME statement needs to be converted to use the reference that was defined in the metadata repository by the data administrator or SAS administrator:

```
libname wombat META library='Wombat Data' repname='Foundation' port=8561;
```

The LIBNAME statement does not contain a physical location for the data. Instead, it uses keyword options that use the Metadata LIBNAME Engine (MLE) to point to the data. The data administrator registers libraries and tables in the metadata. The library is given a unique name in the metadata (such as Wombat Data). The data administrator can set access permissions. The library is located in a repository (such as Foundation) and its name is used in the REPNAME option. Finally, a connection to the metadata is made on a particular port number (such as 8561). In fact, a more simplified LIBNAME statement could be issued if all of the defaults for REPNAME and PORT were taken during the platform installation. This simplified LIBNAME statement would be:

```
libname wombat META library='Wombat Data';
```

You should check with your data administrator or SAS administrator to verify that the defaults at platform installation were used.

If you want to use a SAS format library or a macro autocall library for your SAS macro programs, you need to move your formats and your macro code to the application server path location defined in the metadata. For example, on a single machine installation, the location is defined as:

```
C:\SAS\BIArchitecture\Lev1\SASMain\SASEnvironment\
```

Your data administrator or SAS administrator can provide you with the location for your particular installation.

CONVERSION OF SAS/GRAPH® PROGRAMS

When you convert a SAS/GRAPH program, you have to make several decisions that are unique to the production of graphical output.

1. For Step 2: Convert the Program with Required Syntax:
   - What device driver should be used to create my output?
   - What graphics options need to be set in overrides to the %STPBEGIN macro call?
2. For Step 3: Register the Stored Process Metadata:
   - What stored process output type do I choose?

The answers to the first two questions determine how to override the %STPBEGIN reserved macro variables. Macro variables to override, such as &_GOPT_DEVICE, &_GOPT_XPIXELS, or &_GOPT_YPIXELS, are straightforward because they correspond directly to SAS/GRAPH GOPTIONS. The stored process program that is being converted did not use any SAS/GRAPH GOPTIONS. A list of SAS/GRAPH options is provided at the end of the paper.

The answer to the last question affects how the stored process is registered in the metadata.

STEP 3: REGISTER THE STORED PROCESS METADATA

After the program is converted, it must be registered in the metadata. Using SAS® Management Console, the metadata for the stored process was registered with only the REG parameter and the general properties and execution environment settings shown in the following three figures:

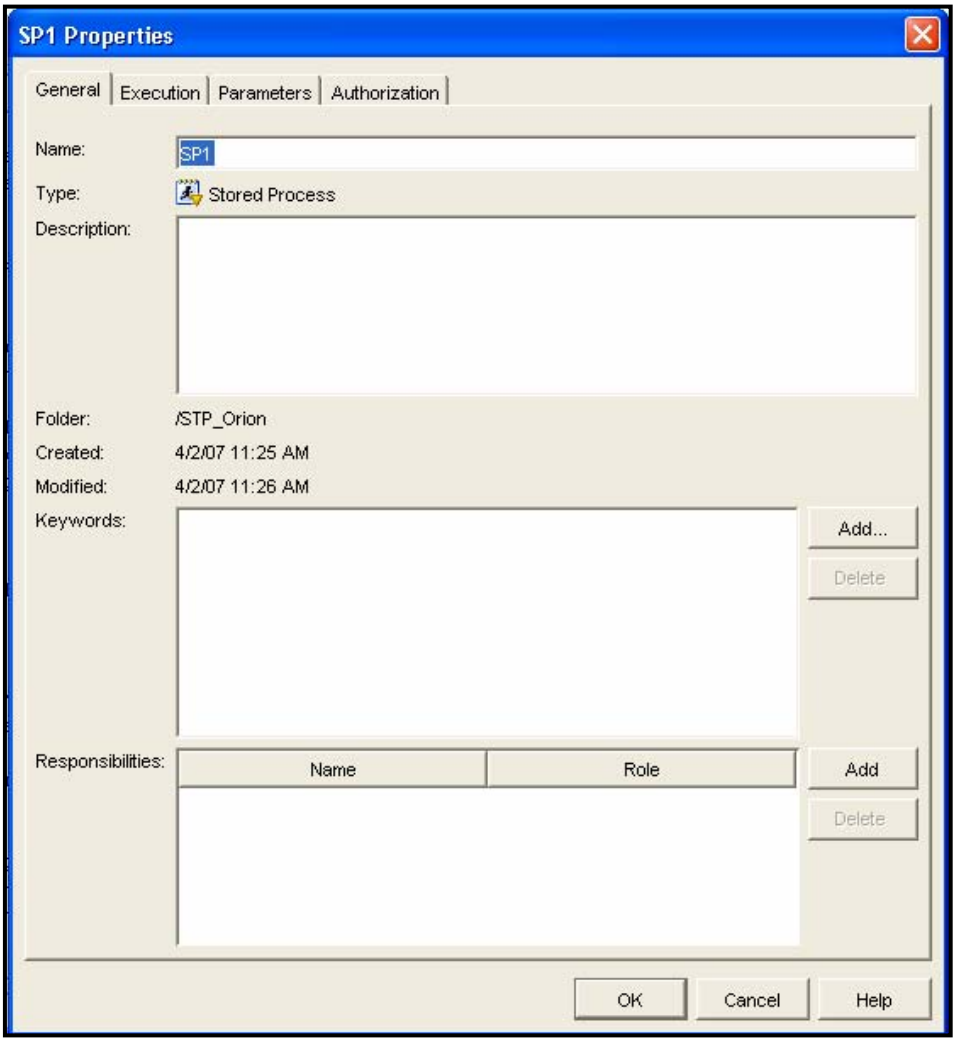

Figure 5: Descriptive Information for the Stored Process

In this example, the stored process name is the same as the SAS program name. In a true production environment, you could name the stored process something like “Performance Report.” After providing the descriptive information, the next dialog box allows you to set the execution environment.

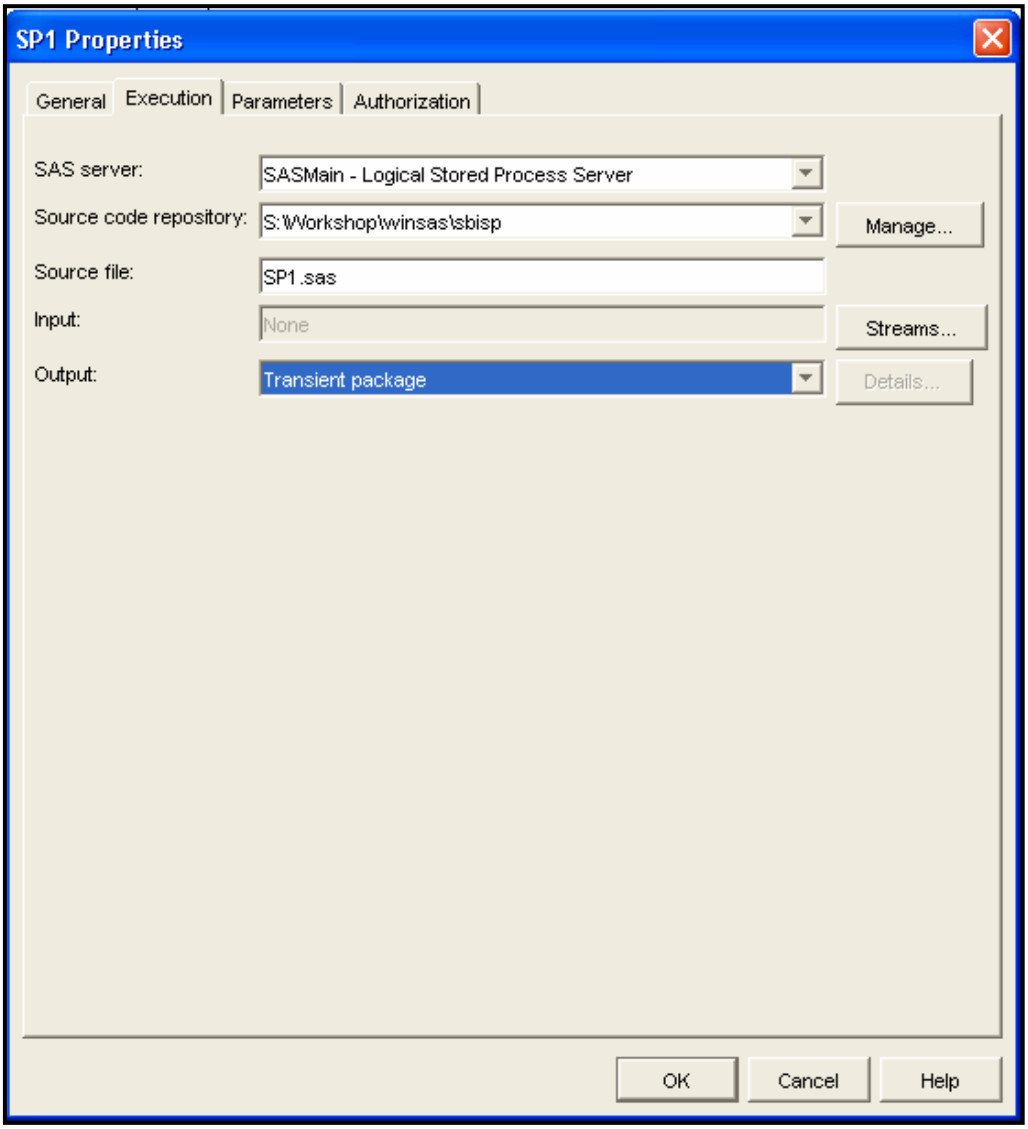

Figure 6: Execution Properties for the Stored Process

The name of the server, such as SASMain, is set by your system administrator. For simple tabular output that returns results to the client application, the SAS Stored Process Server and streaming are good choices for **`SAS server`** and **`Output`**. However, for the example program, because the results contain tables and graphical output, Transient package is a better choice for output type.

The differences between streaming, transient package, and permanent package output types are discussed in a subsequent section. After setting the execution environment properties, the next step is to register the input parameter. As part of this process, you have to know the name of the SAS macro variable used in your program, whether you want a single-selection or multiple-selection parameter and whether you want to set a default value for the parameter.

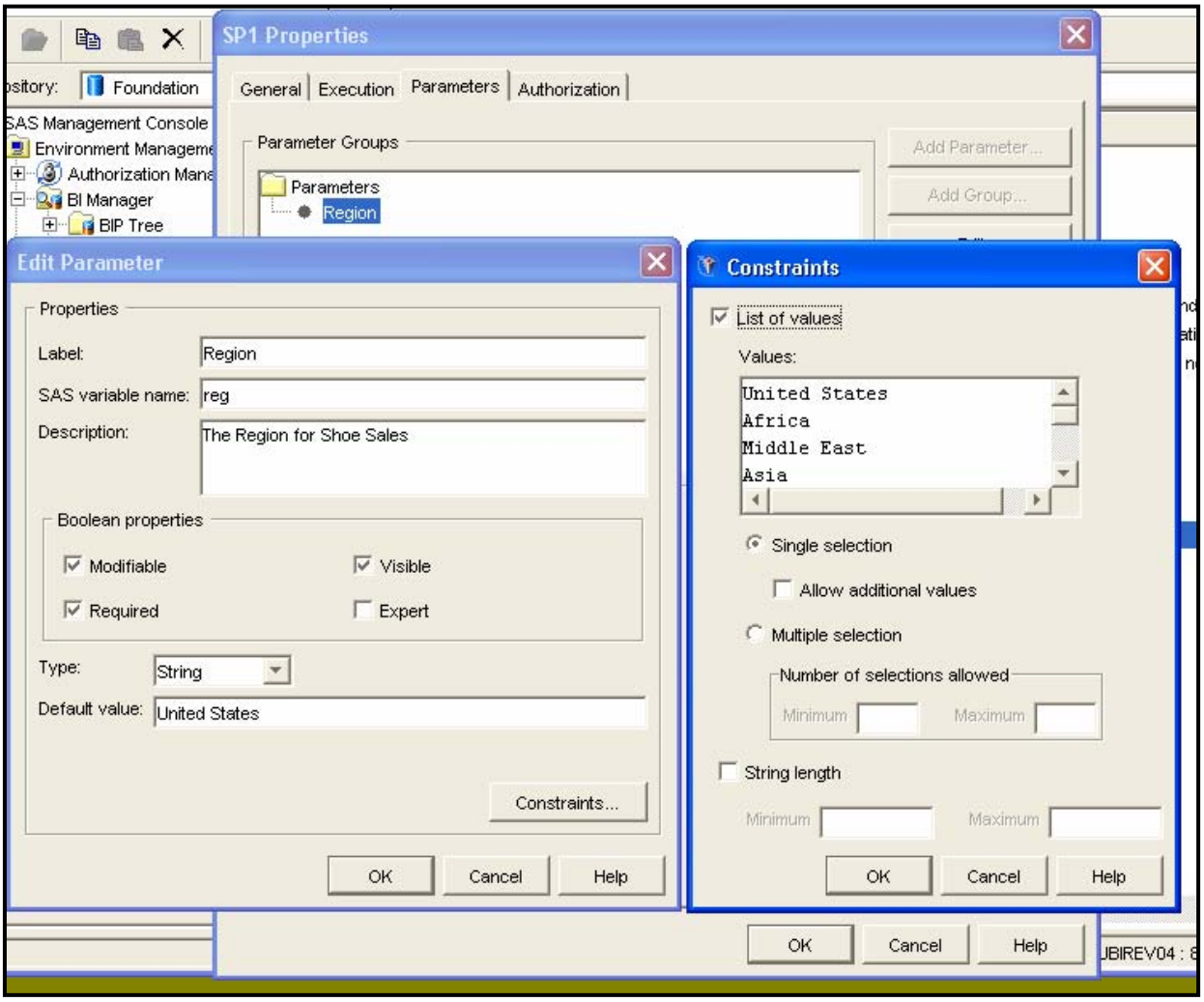

Figure 7: Input Parameter Properties for the Stored Process

The choices for **`Boolean properties`** set the &REG parameter to **`Modifiable`**, **`Required`**, and **`Visible`**. These choices ensure that the stored process consumer selects a region from the list of values that are entered under constraints. Either SAS Management Console or SAS Enterprise Guide allows you to set a default value to be used if the stored process is submitted via batch or by the SAS Stored Process Web Application.

A single-selection choice means that the macro variable name &REG will be used for the single choice that the user makes. If multiple-selection choices are allowed (remember that this kind of stored process can execute only on the SAS Workspace Server), then macro variables are created as shown in Figure 8.

There are SAS macro coding issues if you enable multiple selections. What if the stored process consumer selects one value or two values? The answer is the SAS Stored Process Server creates a different set of macro variables and you might need to use macro conditional logic to perform the processing that you want. Table 1 outlines the way the SAS Stored Process Server creates macro variables when there are multiple selections.

Table 2 shows examples of coding techniques that you might use in a multiple-selection stored process. Other types of SAS macro conditional logic could be used within your stored process.

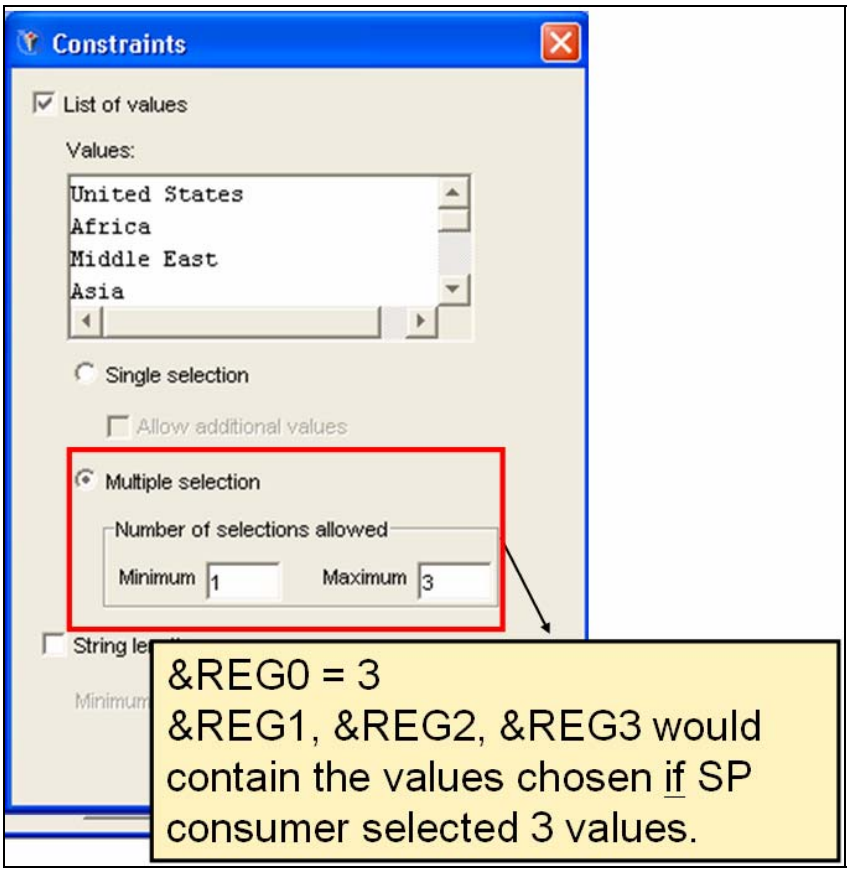

Figure 8: Macro Variable Values for the Stored Process with Multiple Selections Enabled

Stored Process Input Parameter Name	Macro Variable Created by SAS Stored Process Server	If Two or More Selections, the Macro Variables Contain…	If One Selection, the Macro Variables Contain…	If No Selection, the Macro Variables Contain…
REG	&REG0	number of selections	no value; if you are using a %DO loop, then you need to set &REG0=1	no value; if you are using a %DO loop, then you need to set &REG0=0
	&REG	one of the selections	the selection value; if you want to use a %DO loop, then you must create &REG1 and assign it the value that is held in &REG	nothing because they are not created; if &REG is NULL and &REG0 is NULL, then the user did not make a selection; the %DO loop will not process so you should perform a default process or write a message to the log
	&REG1-&REG*n*	all selected values	nothing because they are not created; you need to assign &REG1 the value that is held in &REG	nothing because they are not created

Table 1: Macro Variables Created with Multiple Selections Enabled Depending on User Choices

Code Example If Two or More Selections	Code Example If One Selection	Code Example If No Selections

`%do i = 1 %to &reg0;`
`  ** code to execute`
`  for every choice;`
`%end;`
	`%if &reg0 = %then %do;`
	`  %let &reg0 = 1;`
	`  %let &reg1 = &reg;`
	`  %do i = 1 %to &reg0;`
	`  ** %do loop will`
	`  execute 1 time;`
	`  %end;`
	`%end;`
		`%if &reg = %then %do;`
		`  %let &reg0 = 0;`
		`  %do i = 1 %to &reg0;`
		`  ** %do loop will not`
		`  execute;`
		`  %end;`
		`  ** Error Handling or set`
		`  default here;`
		`%end;`

Table 2: Macro Code Handling Examples for Multiple Selections

Because the SP1 stored process allows only a single selection for the &REG input parameter, multiple selection code is not needed for this example. The final dialog box for the properties sets authorization; there is no special authorization for this stored process, so the next step is to test the stored process.

STEP 4: TEST THE STORED PROCESS

When the SPI stored process is added to a SAS Enterprise Guide project, and you then run the stored process, the prompting interface opens with a single tab showing the Region input parameter. The list of constraints are the possible choices. In this case, the user can select only one choice for region.

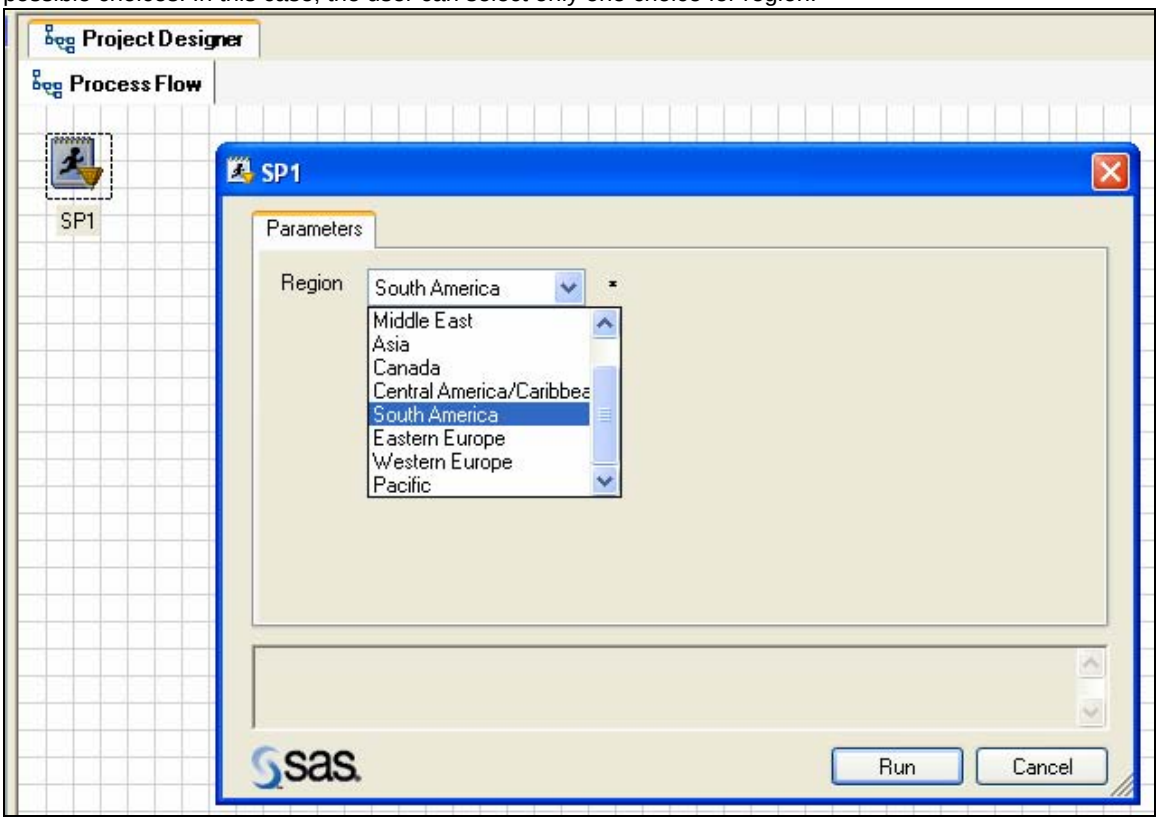

Figure 9: Prompting Interface for the Stored Process in SAS Enterprise Guide

After you click **Run**, the SAS Stored Process Server is sent instructions on where to find the SP1 program. Then, the program is run on the server. The partial results for South America are shown in Figures 10, 11 and 12 using the EGDEFAULT style, which is the default for SAS Enterprise Guide output.

Product	Total Sales	Percent of Region
Boot	$245,675	10.09%
Men's Casual	$544,950	22.38%
Men's Dress	$425,669	17.48%
Sandal	$165,925	6.81%
Slipper	$462,651	19.00%
Sport Shoe	$33,061	1.36%
Women's Casual	$179,227	7.36%
Women's Dress	$377,625	15.51%
	$2,434,783	*100.00%*

Figure 10: First PROC REPORT Output in EGDEFAULT Style

Subsidiary Report for South America

Region	Subsidiary	Total Sales	Percent of Region
South America	Bogota	$206,234	8.47%
	Buenos Aires	$118,283	4.86%
	Caracas	$789,323	32.42%
	La Paz	$530,506	21.79%
	Montevideo	$318,646	13.09%
	Santiago	$104,956	4.31%
	Sao Paulo	$366,835	15.07%
South America		*$2,434,783*	*100.00%*

Figure 11: Second PROC REPORT Output in EGDEFAULT Style

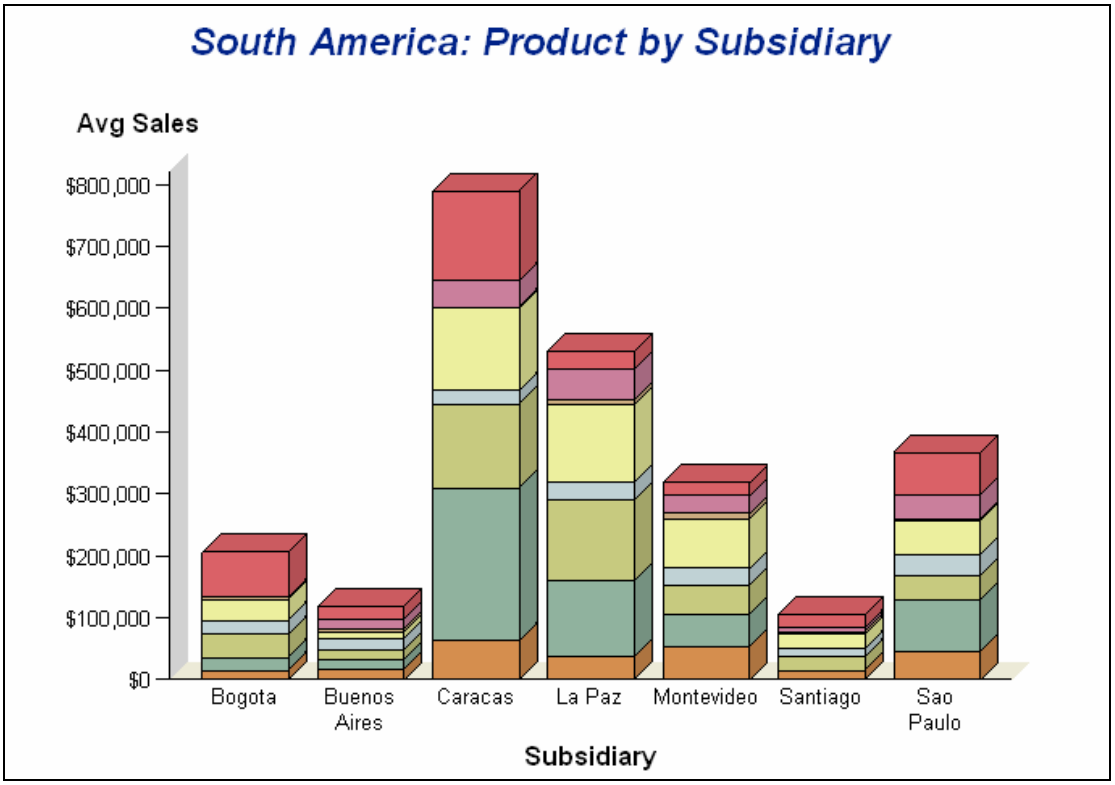

Figure 12: PROC GCHART Output in EGDEFAULT Style

STREAMING, TRANSIENT PACKAGE, AND PERMANENT PACKAGE RESULTS

The default type of output that is created by the SAS Enterprise Guide Stored Process Wizard for returning tabular results is streaming. This kind of output uses HTTP to return results from the server to the client application that made the request. When streaming results are received by the client application, they are held in a temporary cache location and must be explicitly saved. Because our program contained tables and graphs, streaming is not an appropriate choice.

If you have a simple text-based report (such as tabular results), the streaming output type is appropriate. However, a rule of streaming output is that only one content type can be delivered via the client/server connection. SAS Enterprise Guide follows this rule by having an option that forces streaming output to be transient package output (see Figure 13). Other client applications do not have this option, so you must set the output type appropriately.

Figure 13 shows the option that makes graphical output presentable in SAS Enterprise Guide, but not in Microsoft Word. The **`Force Streaming to Transient`** option protects you from setting streaming as the stored process output type.

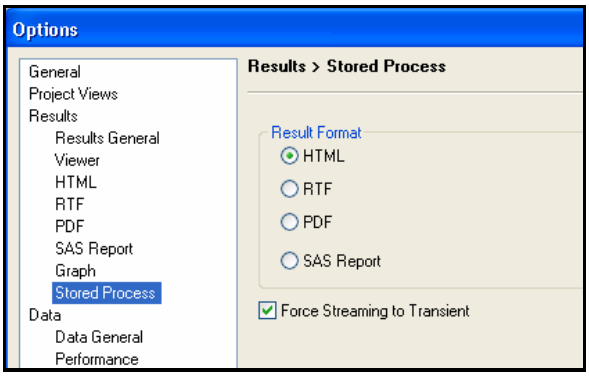

Figure13: Force Streaming to Transient Option

To get to this option, select **Tools → Options**. The default SAS Enterprise Guide settings for a stored process are displayed. If you turned off this option (by deselecting it), then SAS Enterprise Guide will have a red X indicating that there were problems with the graphical output.

Figure14: Tables and Graphs Stored Process Returns Streaming Output, But No Graphical Output

The SAS Enterprise Intelligence Platform can produce two other kinds of output. Transient package output is returned immediately to the client application, and images are held in a temporary cache location on the client machine. When the client application closes, all the transient package output files are cleared. Transient package output is

appropriate for the converted stored process in the example because the report contains tabular output from the two PROC REPORT steps, and it contains a graphical image from the PROC GCHART step.

Permanent package output is returned immediately to the client application; however, a stored process that creates permanent package output can either write the output to a permanent location on the server, or it can write the output to a WebDAV repository. Permanent package output is written not as an HTML or PDF file, but as a package of files in a special container called a SAS package file, which is similar to a ZIP file or a TAR file. The SAS package file permanently stores separate files, which do not have to be the same file type. For an explanation of the possible file types that you can store in a SAS package file, see "About Packages: Package Content" in the *SAS 9.1.3 Integration Technologies Developer's Guide.* For more information about administering and configuring Xythos to publish output to a WebDAV repository, see "Implementing Authentication and Authorization for the Xythos WFS WebDAV Server" in the *SAS 9.1.3 Integration Technologies Server Administrator's Guide*.

CONCLUSION

Once you understand how to convert your SAS programs into stored processes using `*ProcessBody` and %STPBEGIN and %STPEND, you are well on your way to understanding the new SAS 9 paradigm. Even though some changes are required, these changes are easy to understand. Most times, all you have to change is the wrapper code in your program—either the ODS sandwich or the macro definition.

Of course, more planning has to take place, such as making sure that the libraries are defined and the tables are visible in the libraries. You might not be working solo anymore; you might have to take chocolate to the data administrator or the SAS administrator to get the library names, formats, and autocall macros that have been defined in the metadata repository. It's not hard, it's just different. And, given that you will be able to deploy your stored processes across multiple client applications and across different groups of users, the trade-offs are worth it!

It is easy to convert your legacy programs into stored processes within the context of the SAS Enterprise Intelligence Platform. Further study of issues, such as selecting output types, publishing packages to a WebDAV repository, or writing a custom Web application (from which you could call a stored process via URL), will make your stored processes even more flexible and powerful.

APPENDIX

PARTIAL LIST OF RESERVED MACRO VARIABLES USED BY %STPBEGIN MACRO PROGRAM

Variable Name	Used By	Description
_GOPT_DEVICE, _GOPT_HSIZE, _GOPT_VSIZE, _GOPT_XPIXELS, and _GOPT_YPIXELS	%STPBEGIN and %STPEND	Sets the corresponding SAS/GRAPH option. See the DEVICE, HSIZE, VSIZE, XPIXELS, and YPIXELS options in "Graphics Options and Device Parameters Dictionary," in *SAS/GRAPH Reference*, in SAS Help and Documentation, for more information.
_GOPTIONS	%STPBEGIN and %STPEND	Sets any SAS/GRAPH option documented in "Graphics Options and Device Parameters Dictionary," in *SAS/GRAPH Reference*, in SAS Help and Documentation. You must specify the option name and its value in the syntax used for the GOPTIONS statement. For example, set _GOPTIONS to ftext=Swiss htext=2 to specify the Swiss text font with a height of 2.
_ODSDEST	%STPBEGIN and %STPEND	Specifies the ODS destination. The default ODS destination is HTML if _ODSDEST is not specified.
_ODSOPTIONS	%STPBEGIN and %STPEND	Specifies options to be appended to the ODS statement. Do not use this macro variable to override options defined by a specific macro variable. For example, do not specify ENCODING=*value* in this macro variable because it conflicts with _ODSENCODING. NOGTITLE and NOGFOOTNOTE are appended to the ODS statement as default options. You can override this by specifying GTITLE or GFOOTNOTE for _ODSOPTIONS.
_ODSSTYLE	%STPBEGIN and %STPEND	Sets the ODS STYLE= option. You can specify any ODS style that is valid on your system and is available on the server where the stored process is executing.

_ODSSTYLESHEET	%STPBEGIN and %STPEND	Sets the ODS STYLEHEET= option. To store a generated style sheet in a catalog entry and automatically replay it using the SAS Stored Process Web Application, specify myfile.css (url="myfile.css").

For the complete list of reserved macro variables, see “Reserved Macro Variables,” in the *SAS 9.1.3 Integration Technologies Developer's Guide*.

FULL TEXT OF BASIC SAS PROGRAM CASE_STUDY1.SAS

```
%global Reg FReg UReg;

%macro RegRept(Reg=Canada);

%let FReg = %sysfunc(compress(&Reg,/));
%let FReg = %sysfunc(compress(&FReg));
%let Ureg = %sysfunc(upcase(&Reg));

proc sort data=sashelp.shoes out=shoes;
by region subsidiary product;
where upcase(Region) = "&UReg";
run;

ods listing close;
ods html path='c:\temp' (url=none)
         file="&FReg..html" style=sasweb;

options center missing=' ';
proc report data=shoes nowd
     style(header)={vjust=b};
  title "All Products for &Reg";
  column region product sales sales=ps_sales ;
  define region /group noprint;
  define product /group;
  define sales/ sum f=dollar14.;
  define ps_sales / pctsum f=percent10.2 "Percent/of/Region";
  break after region /summarize skip;
run;

proc report data=shoes nowd
     style(header)={vjust=b};

  title "Subsidiary Report for &Reg";
  column region subsidiary sales sales=ps_sales;
  define region /group;
  define subsidiary /group;
  define sales/ sum f=dollar14.;
  define ps_sales / pctsum f=percent10.2 "Percent/of/Region";
  break after region /summarize skip;
run;

goptions reset=all device=actximg;
axis1 minor=none  label=("Subsidiary") ;
axis2 minor=none  label=("Avg Sales") ;

proc gchart data=shoes;
  title "&Reg: Product by Subsidiary";
  vbar3d Subsidiary /
        sumvar=sales type=mean
```

```
        name="Sgrf" subgroup=Product
        maxis=axis1 raxis=axis2;
run;
quit;

ods html close;
options missing=.;
title;

%mend RegRept;

%RegRept(Reg=Africa);
%RegRept(Reg=Canada);
%RegRept(Reg=Pacific);
%RegRept(Reg=United States);
%RegRept(Reg=Western Europe);
%RegRept(Reg=Central America/Caribbean);
%RegRept(Reg=United Kingdom);
%RegRept(Reg=Eastern Europe);
%RegRept(Reg=Asia);
```

FULL TEXT OF CONVERTED STORED PROCESS SP1.SAS

```
%macro RegRept;

%let Ureg = %sysfunc(upcase(&Reg));

proc sort data=sashelp.shoes out=shoes;
by region subsidiary product;
where upcase(Region) = "&UReg";
run;

options center missing=' ';
proc report data=shoes nowd
     style(header)={vjust=b};
  title "All Products for &Reg";
  column region product sales sales=ps_sales ;
  define region /group noprint;
  define product /group;
  define sales/ sum f=dollar14.;
  define ps_sales / pctsum f=percent10.2 "Percent/of/Region";
  break after region /summarize skip;
run;

proc report data=shoes nowd
     style(header)={vjust=b};
  title "Subsidiary Report for &Reg";
  column region subsidiary sales sales=ps_sales;
  define region /group;
  define subsidiary /group;
  define sales/ sum f=dollar14.;
  define ps_sales / pctsum f=percent10.2 "Percent/of/Region";
  break after region /summarize skip;
run;

goptions reset=all device=actximg;
axis1 minor=none  label=("Subsidiary") ;
axis2 minor=none  label=("Avg Sales") ;
```

```
proc gchart data=shoes;
  title "&Reg: Product by Subsidiary";
  vbar3d Subsidiary /
        sumvar=sales type=mean
        name="Sgrf" subgroup=Product
        maxis=axis1 raxis=axis2;
run;
quit;
options missing=.;
title;
%mend RegRept;

*ProcessBody;
%global Reg UReg;
%stpbegin;
%RegRept;
%stpend;
```

REFERENCES

SAS Institute Inc. 2007. *SAS 9.1.3 Integration Technologies Developer's Guide.* Cary, NC: SAS Institute Inc. Available at support.sas.com/rnd/itech/doc9/dev_guide.

SAS Institute Inc. 2007. *SAS 9.1.3 Integration Technologies Server Administrator's Guide.* Cary, NC: SAS Institute Inc. Available at support.sas.com/rnd/itech/doc9/admin_oma.

Weinstein, Heather. 2007. “Converting SAS/IntrNet Programs to SAS Stored Processes.” *Proceedings of the SAS Global Forum 2007 Conference*. Cary, NC: SAS Institute Inc.

RESOURCES

You can download the programs and stored process in this paper from the SAS Web site for Technical Papers and Presentations for SAS Global Forum 2008 Conference at support.sas.com/rnd/papers/.

ACKNOWLEDGMENTS

The author thanks Chris Hemedinger, Stephen McDaniel, and Stacey Syphus for graciously answering all SAS Enterprise Guide questions. In addition, this paper would not have been possible without the encouragement and support of Herb Kirk, Larry Stewart, and Eric Rossland. Thanks are also due to Diane Hatcher, Jane Stroupe, Linda Jolley, Christine Riddiough, and Rick Bell, who helped review this paper.

CONTACT INFORMATION

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

Cynthia L. Zender
SAS Institute, Inc.
Work Phone: 575-522-3803
E-mail: Cynthia.Zender@sas.com