# SAS® 9.4 Stored Processes: Developer's Guide, Third Edition

# Contents

# What's New in SAS 9.4 Stored Processes

## Overview

SAS 9.4 Stored Processes introduces new features for the STP procedure and the SAS Stored Process Web Application, as well as general enhancements.

## STP Procedure Enhancements

The following enhancements have been added to PROC STP:

- Beginning with SAS 9.4M3, you can specify file locations as a fileref of an aggregate storage location and the name of a file or member, enclosed in parentheses, that resides in that location. This is valid for the INPUTFILE, LOG, and OUTPUTFILE statements.

- Beginning with SAS 9.4M3, if you specify a null value for the INPUTPARAM statement, then any default value that was specified for the parameter in metadata is ignored and the value is null. However, if the metadata specifies that the parameter requires a non-blank value, then you will receive an error.

- Beginning with SAS 9.4M3, you can specify multiple lines of text for the value of the input parameter in the INPUTPARAM statement.

- In SAS 9.4M2, the following two automatic macro variables have been added. These macro variables are generated by PROC STP, and are used to communicate status from the stored process.

  _STPCC
      specifies the maximum exit code from the stored process environment.

  _STPRC
      specifies the current return code from the stored process environment.

- Beginning with SAS 9.4M2, the INPUTPARAM statement for PROC STP supports selection groups and an alternate way to specify a range value for an input parameter.

- Stored processes can be executed on a remote Grid or CONNECT server. The SERVER= option specifies whether a stored process executes locally or remotely. Remote execution is the default, starting with SAS 9.4.

- The REMOTE argument on the INPUTDATA, INPUTFILE, OUTPUTDATA, and OUTPUTFILE statements specifies that a data set or file exists on the remote server.

- The MEMTYPE argument on the INPUTDATA and OUTPUTDATA statements specifies whether the member type is a data set, data view, or catalog. Catalog support is new for PROC STP in SAS 9.4.

- User formats can be specified, and you can use the FMTSEARCH system option to search format catalogs.

For more information about PROC STP, see Chapter 8, "STP Procedure," on page 149.

# SAS Stored Process Web Application Enhancements

The following enhancements have been added to the SAS Stored Process Web Application:

- In SAS 9.4M3, the App.AllowGuest initialization parameter has been added. This parameter causes a **Guest** option to be displayed on the SAS Logon Manager sign-in screen. When the AllowGuest parameter is set to `true`, App.AllowGuest must also be set to `true`, or GuestUsername and GuestPassword must be set.

- You can create a NotFound.jsp page to display when the stored process cannot be located.

- The following initialization parameters have been added:

| | |
|---|---|
| AllowBasicAuthentication | enables the SAS Stored Process Web Application to bypass the Logon Manager and to handle user verification through the `http://`*yourserver*`.com:8080/SASStoredProcess/do1` URL. |
| InternalURL | specifies a URL for the SAS Stored Process Web Application to use to retrieve resource files. |
| RTL.SupportedLocales | specifies a comma-delimited list of locales that should be displayed right to left. For SAS 9.4, this is supported only by the SAS Stored Process Web Application prompt page. |

For more information, see "Initialization Parameters".

- Instructions for enabling logging and encoding characters have been added. For more information, see "Debugging in the SAS Stored Process Web Application ".

■   Instead of using a JSP file to create a custom input form, you can use HTML
     following the CARDS4 statement in a DATA step. For more information, see
     "Passing Data through the DATA Step".

# General Enhancements

The following general enhancements have been added to SAS Stored Processes:

■   In SAS 9.4M6, functionality has been added to address security vulnerabilities
     with stored processes. This functionality removes semicolons from most input
     parameters because semicolons in an input parameter can execute a code
     injection. If you have existing code that uses semicolons in input parameters,
     you can use the STP_UNSAFE_DISABLE environment parameter to disable this
     functionality.

■   In SAS 9.4M6, the following _SUPPRESS_MVARS reserved macro variable has
     been added. This macro variable enables you to hide all macro variables and
     their values from the SAS log.

■   In SAS 9.4M3, the following reserved macro variables have been added:

     _HTTP_TOKENAUTH       enables token authentication to a SAS Content
                           Server when _RESULT is set to
                           `PACKAGE_TO_WEBDAV`.

     _TIMEZONE             sets the TIMEZONE SAS system option.

■   You can use the SYSPROCESSMODE automatic macro variable to find out
     which server type the stored process is using to execute. For more information
     about using the automatic macro variable SYSPROCESSMODE, see
     "SYSPROCESSMODE Automatic Macro Variable" in *SAS Macro Language:
     Reference*.

■   The following reserved macro variables have been added:

     _FILE_DESCRIPTION     specifies the description that is to be used for all
                           files that are added to a package.

     _FMTSEARCH            specifies which format catalogs need to be
                           appended to the FMTSEARCH option.

     _REPORTID             specifies the ID of the generation of a stored
                           process report.

     _REQENCODING          specifies the servlet input encoding.

     _USERLOCALE           contains the locale for the user that was set in the
                           user preferences (for web clients).

     The following values have been added for existing reserved macro variables:

     _ACTION=RECALL        can be combined with the PROPERTIES value to
                           retain prompt values throughout a session.

     _RESULT=PACKAGE       specifies that a transient package is returned to the
                           client.

For more information, see "Using Reserved Macro Variables".

# 1

# Overview of SAS Stored Processes

## What Are SAS Stored Processes?

A stored process is a SAS program that is stored on a server and defined in metadata, and which can be executed as requested by client applications. You can use stored processes for web reporting, 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.

## Why Are SAS Stored Processes Important?

The ability to store your SAS programs on the server provides an effective method for change control management. For example, instead of embedding the SAS code into client applications, you can centrally maintain and manage this code from the server. This gives you the ability to change your SAS programs and at the same

time ensure that every client that invokes a stored process always gets the latest version available.

The stored process concept becomes even more powerful when you consider that these SAS programs can be invoked from multiple client contexts. For example, you might deploy Java applets and Windows applications that invoke your stored processes. If your strategy is to use a multi-tiered architecture, you can use Enterprise JavaBeans (EJB) technology, for example, to invoke the same stored processes from an application server.

Using stored processes also enhances security and application integrity because the programs that access your sensitive data are contained on the server instead of being widely distributed with the client applications.

# Which Clients Can Use SAS Stored Processes?

SAS Stored Processes can be used in many different client applications. The following list gives a brief overview of each application so that you can determine which client best meets your needs.

JMP
>   You can use JMP to run stored processes and view results. See *Using JMP* for more information.

SAS Add-In for Microsoft Office
>   The SAS Add-In for Microsoft Office is a Component Object Model (COM) add-in that extends Microsoft Office by enabling you to dynamically execute stored processes and embed the results in Microsoft Word documents, Microsoft Excel spreadsheets, and Microsoft PowerPoint presentations. Also, within Excel, you can use the SAS add-in to access and view SAS data sources or any data source that is available from your SAS server, and analyze SAS or Excel data by using analytic tasks. For more information about using stored processes with the SAS Add-In for Microsoft Office, see the SAS Add-In for Microsoft Office Online Help, which is located within the product.

SAS BI Dashboard
>   You can use SAS BI Dashboard to execute stored processes and to include stored processes or stored process results in a dashboard. When a dashboard has an indicator that was configured with stored process indicator data, users can see output from that stored process if it belongs to the displayed dashboard. See *SAS BI Dashboard: User's Guide* for more information.

SAS BI Web Services
>   SAS BI Web Services provide a web service interface to SAS Stored Processes. Starting with SAS 9.3, all stored processes are available individually for execution using web services, without any action required from the user. For more information about using stored processes with SAS BI Web Services, see the *SAS BI Web Services: Developer's Guide*.

SAS Data Integration Studio
>   SAS Data Integration Studio enables its administrators to publish jobs as stored processes. SAS Data Integration Studio can generate code that converts a job

into a stored process, which is saved to a file and can be executed later by the SAS Stored Process Server. Metadata about the stored process is saved in the current metadata repository. For more information about using stored processes with SAS Data Integration Studio, see the SAS Data Integration Studio product Help.

SAS Enterprise Guide
SAS Enterprise Guide provides an integrated solution for authoring, editing, and testing stored processes. You can create stored processes from existing or new SAS code and create stored processes automatically from SAS Enterprise Guide tasks. Metadata registration and source code management are handled from one interface. SAS Enterprise Guide also has the capability to execute stored processes, which enables you to modify and test your stored process without leaving the SAS Enterprise Guide environment. For more information about using stored processes with SAS Enterprise Guide, see the SAS Enterprise Guide product Help.

SAS Information Delivery Portal
The SAS Information Delivery Portal provides integrated web access to SAS reports, stored processes, information maps, and channels. If you have installed the SAS Information Delivery Portal, you can make stored processes available to be executed from the portal without the need for additional programming. The SAS Information Delivery Portal includes the SAS Stored Process Web Application. For more information about using stored processes with the SAS Information Delivery Portal, see the *SAS Intelligence Platform: Web Application Administration Guide*.

**Note:** Starting with SAS 9.4M8, SAS Information Delivery Portal and SAS BI Portlets are not available from SAS. If you have an existing installation of SAS Information Delivery Portal and SAS BI Portlets in your environment and plan to upgrade or migrate to SAS 9.4M8 or later, SAS recommends that you first unconfigure and uninstall SAS Information Delivery Portal and SAS BI Portlets. A best practice is to unconfigure retired SAS products before you upgrade and to uninstall them after you upgrade. For more information, see Unconfiguring and Uninstalling Retired Products in *SAS Guide to Software Updates and Product Changes*.

SAS Information Map Studio
Stored processes can be used to implement information map data sources. Stored processes can use the full power of SAS procedures and the DATA step to generate or update the data in an information map. For more information about stored process information maps, see the SAS Information Map Studio product Help.

SAS Stored Process Web Application
The SAS Stored Process Web Application is a Java web application that can execute stored processes and return results to a web browser. The SAS Stored Process Web Application is similar to the SAS/IntrNet Application Broker and has the same general syntax and debugging options as the Application Broker. For examples of this component, see "Using the SAS Stored Process Web Application Pages" on page 110. The SAS Stored Process Web Application is included with the SAS Web Infrastructure Platform, which is a component of SAS Integration Technologies.

SAS Web Report Studio
You can use SAS Web Report Studio to execute stored processes and to include stored processes or stored process results in a report. For more information about using stored processes with SAS Web Report Studio, see the *SAS Web*

*Report Studio: User's Guide*, the *SAS Intelligence Platform: Web Application Administration Guide*, and the SAS Web Report Studio product Help.

Stored Process Java API

The Stored Process Java API is a Java application programming interface (API) that enables you to execute stored processes from a Java program. This API is commonly used in JSP pages, but can also be used from servlets, custom tagsets and other Java applications. The Stored Process Java API is part of SAS Foundation Services; you must deploy SAS Foundation Services in order to use the Stored Process Java API. If you want to register new stored processes and modify metadata for existing stored processes programmatically, use the com.sas.services.storedprocess.metadata API. See the API Javadoc at http://support.sas.com/rnd/javadoc/94 for more details.

Stored Process Windows API

The Stored Process Windows API is a Microsoft .NET application programming interface (API) that enables you to execute stored processes from within the .NET framework (using C# or VB.NET, for example). This API is used by both SAS Enterprise Guide and SAS Add-In for Microsoft Office, and can be used to write ASP.NET or Windows applications. The Stored Process Windows API is part of SAS Integration Technologies; you must deploy SAS Integration Technologies in order to use the Stored Process Windows API.

# What Are SAS IOM Direct Interface Stored Processes?

There are two different types of stored processes. A limited form of stored processes, IOM Direct Interface Stored Processes, was introduced in SAS 8. This type of stored process operates on a SAS Workspace Server and produces packages only. IOM Direct Interface Stored Processes are still fully supported. However, the focus of this documentation is on SAS Stored Processes. SAS Stored Processes are new with SAS® 9, and they can be used with either a SAS Workspace Server or a SAS Stored Process Server.

# 2

# Writing a SAS Stored Process

# Overview of Writing a Stored Process

A stored process is a SAS program that is hosted on a server and described by metadata. Stored processes can be written by anyone who is familiar with the SAS programming language or with the aid of a SAS code generator such as SAS Enterprise Guide. The basic steps for creating a stored process are as follows:

1  Write the stored process.

2  Choose or define a server. For more information, see "Choosing or Defining a Server" on page 59.

3  Register the stored process metadata. For more information, see "Registering the Stored Process Metadata" on page 62.

Almost any SAS program can be a stored process. A stored process can be written using the SAS Program Editor, SAS Enterprise Guide, or any text editor. The following program is a typical stored process:

```
%STPBEGIN;
   title 'Age analysis by sex';
   footnote;
   proc sort data=sashelp.class out=class; by sex age; run;
   proc gchart data=class;
      vbar3d age / group=sex
         discrete
         nozero
         shape=cylinder
         patternid=group;
   run; quit;
   title;
   proc print data=class;
      by sex age;
      id sex age;
      var name height weight;
   run;
%STPEND;
```

The %STPBEGIN and %STPEND macros initialize the Output Delivery System (ODS) and deliver the output to the client. This stored process is capable of generating multiple output formats, including HTML, XML, PDF, CSV, and custom tagsets and then delivering the output through packages or streaming output. For more information, see "Setting Result Capabilities" on page 16.

........................................................................................................................

**Note:** Because the %STPBEGIN and %STPEND macros initialize the Output Delivery System (ODS), you should use them only if your stored process creates ODS output. They are not necessary if the stored process is creating only a table and does not create a report. Another case where they should not be used is when your stored process writes directly to the _WEBOUT fileref, either using the DATA step or some other method. Writing to _WEBOUT is a common technique used in SAS/IntrNet programs.

........................................................................................................................

This sample code generates the following bar chart and table:

*Figure 2.1*   *Stored Process Results: Bar Chart*
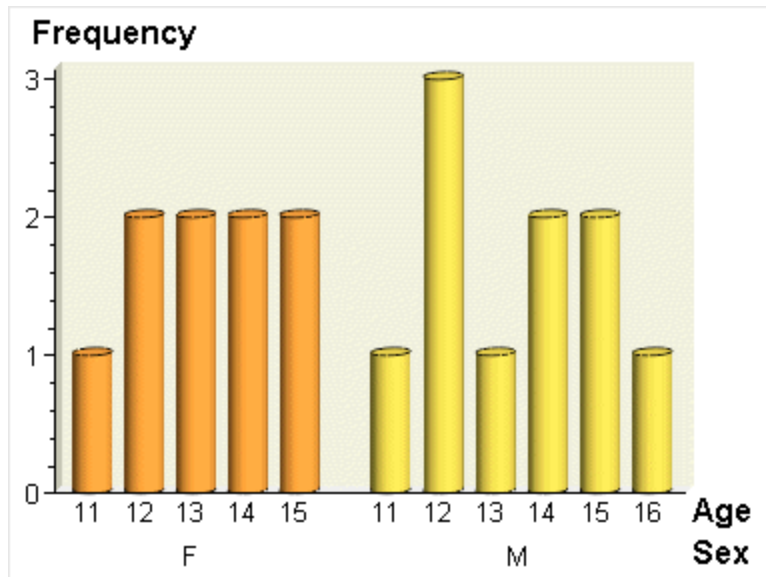
*Figure 2.2*   *Stored Process Results: Table*

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

# Using Input Parameters

## Overview of Input Parameters

Most stored processes require information from the client to perform their intended function. This information can be in the form of presentation options for a report,

selection criteria for data to be analyzed, names of data tables to be used or created, or an unlimited number of other possibilities. Input parameters are the most common way to deliver information from a client to a stored process.

Input parameters are defined as name/value pairs. They appear in a stored process program as global macro variables. For example, if you have a stored process that analyzes monthly sales data, you might accept MONTH and YEAR as input parameters. The stored process program might be similar to the following code:

```
%stpbegin;

title "Product Sales for &MONTH, &YEAR";
proc print data=sales;
where Month eq "&MONTH" and Year eq &YEAR;
var productid product sales salesgoal;
run;

%stpend;
```

Because input parameters are simply macro variables, they can be accessed through normal macro substitution syntax (`&param-name`) or through any other SAS functions that access macro variables (`SYMGET`, `SYMGETC`, or `SYMGETN`). Parameters follow the same rules as SAS macro variables. Names must start with an alphabetic character or underscore and can contain only alphanumeric characters or underscores. The name can be no more than 32 characters long and is not case sensitive. Values can contain any character except a null character and can be up to 65,534 characters in length. For stored processes that are compatible with 9.2 and that run on the workspace server, values are limited to approximately 5950 bytes in length and cannot contain nonprintable characters (including line feeds or carriage returns).

Each stored process client interface provides one or more methods to set input parameters. The Stored Process Java API provides a direct programming interface to set name/value pairs. The SAS Stored Process Web Application allows name/value pairs to be specified directly on a URL or indirectly through posting HTML form data. The SAS Add-In for Microsoft Office provides a property sheet interface to specify parameters.

There are many reserved parameters that are created by the server or the stored process client interface. For a list of these variables, see "Using Reserved Macro Variables" on page 26.

## Standard Header for Parameters

For stored processes that are compatible with 9.2, parameters are not initialized in the same way for the stored process server and the workspace server. The stored process server sets parameter values before the stored process begins to execute. This means the first line of code in the stored process can access any input parameter macro variable. The workspace server does not set input parameters into macro variables until it reaches a `*ProcessBody;` comment line in the stored process:

```
*ProcessBody;
```

A stored process that does not contain this line never receives input parameters when executed on a workspace server. Also, without this comment, the stored process is not able to use reserved macro variables, such as _METAUSER.

It is recommended that you begin all stored processes (regardless of the server types) with %GLOBAL declarations for all of your input parameters followed by the `*ProcessBody;` comment:

```
/*******************************************************
 * Standard header comment documenting your
 * stored process and input parameters.
 * *********************************************** */
%global parmone parmtwo parmthree;
%global parmfour;
*ProcessBody;

... remainder of the stored process ...
```

The %GLOBAL declarations create an empty macro variable for each possible input parameter and enable you to reference the macro variable in the stored process even if it was not set by the stored process client. If you do not declare input parameters in a %GLOBAL statement, then any references to an unset input parameter will result in WARNING messages in the SAS log.

**Note:** Starting with 9.3, you do not need to include the `*ProcessBody;` comment.

# Defining Input Parameters

Most stored process client interfaces allow a client to pass any input parameter. Input parameters are defined in SAS Management Console as prompts. Several types of macro variables are generated from prompts, depending on the type of prompt and what other information is included in the prompt definition. There is no requirement to define parameters before executing the stored process, but there are many advantages to describing parameters in stored process metadata. Here are some of the advantages:

- Parameter definitions can specify labels and descriptive text. This information can be used by client interfaces to present a more attractive and informative user interface. Other presentation options include grouping parameters.

- Default values can be specified. However, if the parameter is not required and the default value is cleared in the client, then the parameter value is empty when passed to the stored process.

  **Note:** The STP procedure uses the default value if a value is not specified.

- Default values can be flagged as read-only to allow a fixed parameter value to always be passed in a stored process. This can be useful when using an existing program that accepts many input parameters. You can register a new, simpler stored process that has some fixed value parameters and fewer client-specified parameters. You can also register multiple stored processes for a single program. Each stored process definition can pass in unique fixed parameter values to the executing program to force a particular operation or otherwise affect the execution of the stored process.

- Parameters can be flagged as required. A stored process does not run unless the client specifies values for these parameters.

- Parameters can be limited to a specific type such as text or date. Defining a parameter type causes certain client user interfaces (such as SAS Add-In for Microsoft Office) to present more appropriate input controls. All interfaces reject stored process requests with input parameters that do not match the specified type.

- Parameter values can be limited by specifying enumerated lists or ranges of valid values for a parameter.

- Dates and times, as well as date ranges and time ranges, can be specified as relative values.

- Input parameters can be shared between stored processes. Other applications or software features that support prompts can also take advantage of these prompts.

- Input parameters can be populated dynamically from a data source.

- Dependencies can be specified between input parameters.

- Selection groups can be used.

Parameter metadata for a stored process can be added or modified using SAS Management Console. To define an input parameter for a stored process, click **New Prompt** in the New Stored Process wizard or on the **Parameters** tab in the Stored Process Properties dialog box. For an example of how to add an input parameter to a stored process definition, see "Adding a Parameter to the Stored Process Definition" on page 203.

For information about using prompt features, see "Using Prompts" on page 70. For more information about how to specify values for prompt, and macro variables that are generated by prompts, see Appendix 3, "Formatting Prompt Values and Generating Macro Variables from Prompts," on page 209. For more information about prompt types and defining prompts, see the product Help.

# Unsafe Character Quoting

Input parameter values are specified by the stored process client at run time. The author of a stored process has little control over the values a client can specify. Setting the values directly into SAS macro variables enables clients to insert executable macro code into a stored process and can lead to unexpected behavior or unacceptable security risks. For example, if an input parameter named COMP was set to `Jones&Comp.` and passed directly into the macro variable, then any references to `&COMP` in the stored process program would lead to an invalid recursive macro reference. To avoid this problem, stored process parameters are masked with SAS macro quoting functions before being set into macro variables. In the Jones&Comp example, the COMP parameter has the following setting:

```
%let COMP=%nrstr(JonesComp.);
```

The stored process can then freely use `&COMP` without special handling for unusual input values. Unsafe characters that are masked for input parameters are the ampersand (&), apostrophe ('), percent sign (%), quotation marks ("), and semicolon (;).

**Note:** If you have existing code that relies on input parameters that contain semicolons or other unsafe characters (that is, %,&,",'), this code should be changed for security purposes.

There might be special cases where you want to unmask some or all of the unsafe characters in an input parameter. The STPSRV_UNQUOTE2 function unmasks only matched apostrophe (') or quotation mark (") characters. For more information, see "STPSRV_UNQUOTE2 Function" on page 57. This can be useful for passing in parameters that are used as SAS options. The %UNQUOTE macro function unquotes all characters in an input parameter, but you should use this function only in very limited circumstances. You should carefully analyze the potential risk from unexpected client behavior before unquoting input parameters. Remember that stored processes can be executed from multiple clients. Some client interfaces perform little or no checking of input parameter values before they are passed in to the stored process.

**Note:** An input parameter to a stored process that is compatible with 9.2 and that is executing on a workspace server cannot contain both apostrophe (') and quotation mark (") characters. Attempting to set such an input parameter results in an error.

In SAS 9.4M6, functionality has been added to address security vulnerabilities with stored processes. This functionality removes semicolons from most input parameters because semicolons in an input parameter can execute a code injection. If you have existing code that uses semicolons in input parameters, you can use the STP_UNSAFE_DISABLE environment variable to disable this functionality. Using this environment variable would enable you to continue using semicolons in input parameters. To set the environment variable, add the following line to the SAS configuration file that is used to start the server:

```
-set STP_UNSAFE_DISABLE true
```

In most cases, this means modifying the sasv9_usermods.cfg file for the logical server (that is, SASApp) that contains the stored process server or workspace server that is used to execute the stored processes.

## Input Parameters with Multiple Values

Parameters with multiple values (or alternatively, multiple input parameters with the same name) can be useful in some stored processes. For example, an HTML input form that is used to drive a stored process might contain a group of four check boxes, each named CBOX. The value associated with each box is optOne, optTwo, optThree, and optFour. The HTML for these check boxes might be

```
<input type="CHECKBOX" name="CBOX" value="optOne">
<input type="CHECKBOX" name="CBOX" value="optTwo">
<input type="CHECKBOX" name="CBOX" value="optThree">
<input type="CHECKBOX" name="CBOX" value="optFour">
```

If you select all four boxes and submit the form to the SAS Stored Process Web Application, then the query string looks like this:

```
&CBOX=optOne&CBOX=optTwo&CBOX=optThree&CBOX=optFour
```

Macro variables cannot hold more than one value. The two types of servers that execute stored processes handle this problem in different ways.

The stored process server uses a macro variable naming convention to pass multiple values to the stored process. A numeric suffix is added to the parameter name to distinguish between values. The number of values is set in *<param-name>* 0, the first value is set in *<param-name>*1, and so on. In the previous example, the following macro variables are set as shown in the following table:

**Table 2.1** *Automatically Generated Variables*

| Name/Value Pair | Description |
| --- | --- |
| CBOX = optOne | Specifies the first value. |
| CBOX0 = 4 | Specifies the number of values. |
| CBOX1 = optOne | Specifies the first value. |
| CBOX2 = optTwo | Specifies the second value. |
| CBOX3 = optThree | Specifies the third value. |
| CBOX4 = optFour | Specifies the fourth value. |

Note that the original parameter macro variable (`CBOX`) is always set to the first parameter value.

Any client application can generate multiple value parameters. The typical uses for multiple values are check box groups in HTML input forms and selection lists that allow multiple selection.

If the parameter name is the same as one of the generated variables, then the following error is returned:

```
Multiple definitions of a prompt name are not allowed.  Certain prompt types
expand to multiple prompt names.
```

# Hiding Passwords and Other Sensitive Data

If you are creating a prompt for a password and want the text to be masked as the user is typing, use a text type prompt, and then select `Masked single line (for password entry)` as the text type. For more information, see the prompt help in SAS Management Console.

Even if you decide not to use a masked prompt, the SAS log exposes programs and input parameters, which could pose a security issue. There are some actions that you can take to hide passwords and other sensitive data from the SAS log.

- Hide password values from the SAS log by using the _PASSWORD suffix anywhere in the input parameter name (for example, ABC_PASSWORD, _PASSWORDABC).

- Disable the SAS log by using the `DebugMask` web application initialization parameter. For more information, see "Debugging in the SAS Stored Process Web Application " on page 145.

- Hide specific request variable values from the SAS log by using the prefix _NOLOG_ with macro variables. The _NOLOG_ prefix enables you to create special macro variables that can be sent to the stored process server without publishing the macro variable values in the SAS log. The special macro variables must start with the prefix _NOLOG_. The prefix is not case sensitive. Here is an example of an input parameter with the _NOLOG_ prefix:

  ```
  http://yourserver/SASStoredProcess/do?
  _program=/WebApps/Sales/Employee+Salary&_nolog_salary=secretpw
  ```

  If _NOLOG_SALARY is displayed in the SAS logs, the log shows the following:

  ```
  _NOLOG_SALARY=XXXXXXXX;
  ```

- Hide all macro variables and their values from the SAS log by using the _SUPPRESS_MVARS input parameter and setting the value to `y` or `yes`. Here is an example that uses _SUPPRESS_MVARS:

  ```
  http://yourserver/SASStoredProcess/do?
  _program=/WebApps/Sales/Employee+Salary&_suppress_mvars=y
  ```

...........................................................................................

**Note:** The _NOLOG_ prefix, _PASSWORD suffix, and _SUPPRESS_MVARS input parameter are effective only if your stored process is running on a stored process server.

...........................................................................................

# Getting Data and Files into and Out of Stored Processes

## Input Files

A stored process can accept input in the form of an input file. An input file is a SAS fileref that is set up before the stored process begins execution. The fileref can point to a local file on the server, a stream fed by the client, a temporary file written by the client, or any other valid fileref that is set up by a stored process client.

Input files are identified by a unique fileref name and are defined in the stored process metadata. The metadata can provide descriptive information or hints to guide clients in the use of input files. Input files can be optional or required. Input files are not assigned if the client does simply corresponding input.

## Input Data

SAS programs frequently process input data sets. An input data set is defined by a macro variable that contains the data set name. Any setup required to access the data set (typically a libref assignment) is handled by the stored process framework.

The following code is an example of a stored process with an input data set named SALESDATA:

```
%stpbegin;

title "Sales for &MONTH, &YEAR";
proc print data=&SALESDATA
  where Month eq "&MONTH" and Year eq &YEAR
  var productid product sales salesgoal;
  run;

%stpend;
```

Different stored process client APIs provide different mechanisms for supplying input data sets to a stored process, including:

Server data sets
    specifies that the macro variable is set to the two-level data set name and, if necessary, the required libref is assigned.

Client data sets
    specifies that the client data set is copied to the WORK library on the server system and the macro variable is set to the temporary data set name.

XML data
    specifies an XML file or data stream as input data to the stored process. The XML data is passed to the stored process via a temporary file and an XML engine libref is assigned to the temporary file. The client can specify an XML map to describe the format of the XML.

Client API access
    specifies direct data access APIs that are appropriate for the technology (for example, a JDBC connection for a Java API.)

# Output Files

A stored process can create output in the form of an output file. An output file is a SAS fileref that is set up before the stored process begins execution. The fileref can point to a local file to be created on the server, a stream consumed by the client, a temporary file that is read by the client after execution is complete, or any other valid fileref that is set up by a stored process client.

Output files are identified by a unique fileref name and are defined in the stored process metadata. The metadata can provide descriptive information or hints to guide clients in the use of output files. Output files can be optional or required. Optional output files are not assigned if the client does not request the output.

The _WEBOUT fileref is frequently used for streaming output from a stored process. The client API can assign the _WEBOUT fileref to a server local file, a temporary file, or any other valid fileref. In the streaming output use case, _WEBOUT is assigned to a stream using the CACHE access method. Then ODS is configured to write to _WEBOUT, and a handle to the _WEBOUT stream is returned to the client web application. The client web application can read output from the stream as it is written by the stored process.

# Output Data

Stored processes support the generation of output data sets. An output data set is defined by a macro variable that contains the data set name. Any setup required to access the data set (typically a libref assignment) is handled by the stored process framework.

The following code is an example of a stored process that generates an output data set named SCORING:

```
%stpbegin;

data &SCORING;
  set MYLIB.RAWDATA;
  score = /* insert scoring algorithm based on input parameters here */;
  run;

%stpend;
```

Different stored process client APIs provide different mechanisms for accessing output data sets, including:

Server data sets
> specifies that the macro variable is set to the two-level data set name and, if necessary, the required libref is assigned.

Client data sets
> specifies that the macro variable is set to a temporary data set name in the WORK library on the server system. After execution is complete, the data set is copied to the client system.

XML data
> assigns a temporary fileref and a corresponding XML engine libref. The macro variable is set to a table in the XML libref. After execution is complete, the resulting XML file is copied to the client system.

Client API access
> specifies direct data access APIs that are appropriate for the technology (for example, a JDBC connection for a Java API.)

# Setting Result Capabilities

A stored process is a SAS program that can produce any type of output that a valid SAS program can produce. Output could include data sets, external files, email messages, SAS catalogs, packages, and many other objects. In some cases, the output (or a result) is delivered to the client application that is executing the stored process. In other cases, the output is generated only on the server.

When you register the stored process, you can specify what type of output the stored process can produce. You can specify **Stream**, **Package**, both output types, or neither output type.

When you run the stored process, the client application chooses the type of output that it prefers. For example, when SAS Web Report Studio runs a stored process, package output is produced. There are four types of client output:

■ The simplest type of output, or result type, is *none*. The client receives no output from the stored process. The stored process is still able to create or update data sets, external files, or other objects, but this output remains on the server. This result type is indicated when the input parameter _RESULT is set to STATUS because only the program status is returned to the client.

■ *Streaming* output delivers a data stream, such as an HTML page or XML document, to the client. This result type is indicated when _RESULT is set to STREAM. The data stream can be textual or binary data and is visible to the stored process program as the _WEBOUT fileref. Any data that is written to the _WEBOUT fileref is streamed back to the client application.

■ Package output can be either transient, meaning that the output is returned only to the client and exists only in the current session, or permanent, meaning that the package is stored or published somewhere and can be accessed even after the session ends.

  □ *Transient package* output returns a temporary package to the client. The package can contain multiple entries, including SAS data sets, HTML files, image files, or any other text or binary files. The package exists only as long as the client is connected to the server. This result type is a convenient way to deliver multiple output objects (such as an HTML page with associated GIF or PNG images) to a client application. Transient package output is indicated when _RESULT is set to PACKAGE_TO_ARCHIVE and the input parameter _ARCHIVE_PATH is set to TEMPFILE.

  □ *Permanent package* output creates a package in a permanent location on a WebDAV server or in the server file system. The package is immediately accessible to the stored process client, but is also permanently accessible to any client with access to WebDAV or the server file system. This result type is a convenient way to publish output for permanent access. Output to WebDAV is indicated when _RESULT is set to PACKAGE_TO_WEBDAV. The input parameter _COLLECTION_URL contains the target location. The input parameters _HTTP_USER and _HTTP_PASSWORD might be set if the WebDAV server is secured and credentials are available. The _HTTP_PROXY_URL parameter is set if an HTTP proxy server is required to access the WebDAV server. Output to the server file system is indicated when _RESULT is set to PACKAGE_TO_ARCHIVE. The input parameters _ARCHIVE_PATH and _ARCHIVE_NAME contain the target repository and filename, respectively.

  Permanent package output can also be published to a channel, an email recipient, or to SharePoint. For more information about the parameters that are used for publishing packages, see "Advanced Package Publishing" on page 21.

......................................................................................................................................

**Note:** Although the result type is chosen when you define a stored process, the result type can be changed by the client application through calls to the Stored Process Service API. Where possible, it is recommended that you write stored processes to support any appropriate client result type. This enables a client application to select the result type most appropriate for that application. The program can determine the desired client result type by examining the _RESULT input parameter. The %STPBEGIN and %STPEND macros include support for any of the four result types. For more information, see "Using the %STPBEGIN and %STPEND Macros" on page 18. The following stored process is capable of

generating streaming, transient package, or permanent package output. (It can also be run with _RESULT set to STATUS, but this would produce no useful result.)

```
%stpbegin;
proc print data=SASHELP.CLASS noobs;
var name age height;
run;
%stpend;
```

The input parameters that were mentioned previously are set by the stored process client APIs and are reserved parameters. They cannot be overridden by passing in new values through the normal parameter interface. Special API methods are provided to set the result type and associated parameters for a stored process. For more information about specific input parameters, see "Using Reserved Macro Variables" on page 26. For more information about developing stored processes that product package results, see "Developing Stored Processes with Package Results " on page 64.

# Using the %STPBEGIN and %STPEND Macros

## Overview of %STPBEGIN and %STPEND

The %STPBEGIN and %STPEND macros provide standardized functionality for generating and delivering output from a stored process. This enables you to write stored processes that generate content in a variety of formats and styles with minimal programming effort. Here is a typical stored process that uses these macros:

```
/* *****************************************************
* Header comment documenting your
* stored process and input parameters.
* ************************************************* */
%global input parameters;

... any pre-processing of input parameters ...

%stpbegin;

... stored process body ...

%stpend;
```

**Note:** You must include a semicolon at the end of the %STPBEGIN and %STPEND macro calls. The %STPBEGIN macro initializes the Output Delivery System (ODS) to generate output from the stored process. The %STPEND macro terminates ODS processing and completes delivery of the output to the client or other destinations.

The macros must be used as a matched pair for proper operation. Streaming output and package output are supported. These macros rely on many reserved macro variables to control their actions. For a more detailed description of each macro variable mentioned in the following sections, see "Using Reserved Macro Variables" on page 26. Because the %STPBEGIN and %STPEND macros initialize the Output Delivery System (ODS), you should use them only if your stored process creates ODS output. For example, the macros are not necessary if the stored process is creating only a table. If you do use the macros, then you should set _ODSDEST to NONE to disable ODS initialization. In these cases, your stored process must explicitly create any output.

# ODS Options

ODS options are specified by various global macro variables. These variables are normally set by input parameters, but can be modified by the stored process. The following variables affect ODS output:

- _ENCODING
- _GOPT_DEVICE
- _GOPT_HSIZE
- _GOPT_VSIZE
- _GOPT_XPIXELS
- _GOPT_YPIXELS
- _GOPTIONS
- _ODSDEST
- _ODSOPTIONS
- _ODSSTYLE
- _ODSSTYLESHEET

The _ODSDEST variable is important because changing this variable enables your stored process to generate HTML, PDF, PostScript, or a variety of other formats, including user-written tagset destinations. Many variables enable you to override ODS options. You must remember to verify whether any options that are specified by the stored process or its clients are compatible with the output destinations that you plan to support.

Some ODS options (for example, BASE) are set based on the result options. For more information, see "Results" on page 20. These options are generally transparent to the stored process author, but they can make it difficult to modify some ODS options in your stored process.

# Overriding Input Parameters

Macro variables that are recognized by the %STPBEGIN macro can be set or modified by the stored process. This is usually done to deny or limit client choices

for that variable. For example, a stored process that requires the use of a particular style might begin with the following statements:

```
%global _ODSSTYLE;

%let _ODSSTYLE=MyStyle;

%stpbegin;
```

Any client-specified value for the _ODSSTYLE variable is ignored and the `MyStyle` style is always used. A more elaborate implementation might validate an input parameter against a list of supported values and log an error or choose a default value if the client input is not supported.

A stored process can modify the macro variables that are used by the %STPBEGIN macro at any time until %STPBEGIN is called. Modifying these reserved macro variables after %STPBEGIN has been called is not recommended.

# Results

The %STPBEGIN and %STPEND macros implement several options for delivering results. For an introduction to the standard options for stored process results, see "Setting Result Capabilities" on page 16. In most cases, a stored process that uses these macros can support all the standard result types with no special coding. The _RESULT variable defines the result type. The following values are supported:

STATUS
    returns only a completion status. An ODS destination is not opened, but the ODS LISTING destination is closed.

STREAM
    returns the body or file output from ODS as a stream. This is the default result type if _RESULT is not set.

There are several values for _RESULT that generate packages. Packages can be delivered directly to the client and published to a more permanent location on the server file system, a WebDAV server, or other destinations. Package creation and delivery are controlled by many reserved macro variables. Here are the variables that are valid for all package destinations:

- _ABSTRACT
- _DESCRIPTION
- _EXPIRATION_DATETIME
- _FILE_DESCRIPTION
- _NAMESPACES
- _NAMEVALUE

Here are additional variables that are recognized for specific _RESULT settings:

PACKAGE_TO_ARCHIVE
    creates an archive package on the server file system that contains the generated output. The _ARCHIVE_PATH and _ARCHIVE_NAME variables specify where the package is created. In addition, _ARCHIVE_FULLPATH is set by %STPEND to hold the full pathname of the created archive package.

PACKAGE_TO_REQUESTER

returns a package to the stored process client. It can also simultaneously create an archive package on the server file system if _ARCHIVE_PATH and _ARCHIVE_NAME are set. This option is valid only on the workspace server, and only for stored processes that are compatible with 9.2.

PACKAGE_TO_WEBDAV

creates a package as a collection on a WebDAV-compliant server. The location of the package is defined by _COLLECTION_URL or _PARENT_URL. Other relevant variables include _HTTP_PASSWORD, _HTTP_PROXY_URL, _HTTP_TOKENAUTH, _HTTP_USER, and _IF_EXISTS.

The %STPBEGIN macro configures ODS to create output files in a temporary working directory. The %STPEND macro then creates the package from all of the files in this temporary directory. The temporary directory is defined by the _STPWORK variable. This variable should not be changed by the stored process, but new entries can be added to the output package by creating files in this directory. For example, the XML LIBNAME engine might be used to create one or more XML files that would be included in the package along with any output that was created by ODS. The temporary directory and any files that are contained in it are automatically deleted when the stored process completes. No cleanup is required in the stored process program.

**Note:** If the environment variable STPWORK is not set when the server is started, then STPBEGIN determines a temporary directory based on the operating system, and places that value in the _STPWORK reserved macro variable. If the environment variable STPWORK is set when the server is started, then STPBEGIN uses the directory specified as a starting point to create the temporary directory. For example, the STPWORK environment variable is set to /usrs/. STPBEGIN creates a temporary subdirectory under /usrs/ and places the full path to the temporary directory in the _STPWORK reserved macro variable.

# Errors

Errors in the %STPBEGIN and %STPEND macros are reported in the _STPERROR macro variable. A value of `0` indicates that the macro completed successfully. A nonzero value indicates that an error occurred.

Because these macros enable clients or stored processes to submit SAS language options (for example, the _ODSOPTIONS variable), it is possible for the macros to fail in unusual ways. Invalid input parameters can cause the stored process to go into syntaxcheck mode (when the SAS OBS option is set to `0`) or to terminate immediately.

# Advanced Package Publishing

The %STPBEGIN and %STPEND macros support some package publishing options that are not recognized by the stored process metadata framework. These options are generally accessed by registering a stored process with no output type.

This causes the stored process to be executed with _RESULT set to STATUS. The stored process can then set _RESULT to one of the following values:

PACKAGE_TO_ARCHIVE
provides several new options when used in this way. Archive packages can be created on HTTP servers that support updates and FTP servers. Variables that control this option include the following:

- _ARCHIVE_NAME
- _ARCHIVE_PATH
- _FTP_PASSWORD
- _FTP_USER
- _GENERATED_NAME
- _HTTP_PASSWORD
- _HTTP_PROXY_URL
- _HTTP_USER

PACKAGE_TO_EMAIL
creates a package and mails it to one or more email addresses. An actual archive package can be mailed, or the package can be created in a public location and a reference URL can be mailed. Variables that control this option include the following:

- _ADDRESSLIST_DATASET_LIBNAME
- _ADDRESSLIST_DATASET_MEMNAME
- _ADDRESSLIST_VARIABLENAME
- _APPLIED_TEXT_VIEWER_NAME
- _APPLIED_VIEWER_NAME
- _ARCHIVE_NAME
- _ARCHIVE_PATH
- _COLLECTION_URL
- _DATASET_OPTIONS
- _EMAIL_ADDRESS
- _FTP_PASSWORD
- _FTP_USER
- _FROM
- _HTTP_PASSWORD
- _HTTP_PROXY_URL
- _HTTP_USER
- _IF_EXISTS
- _PARENT_URL
- _PROCESS_VIEWER
- _REPLYTO
- _SENDER

- _SUBJECT
- _TARGET_VIEW_MIMETYPE
- _TARGET_VIEW_NAME
- _TEXT_VIEWER_NAME
- _VIEWER_NAME

PACKAGE_TO_QUEUE
> creates a package and sends it to one or more message queues. An actual archive package can be sent, or the package can be created in a public location and a reference URL can be sent. Variables that control this option include the following:

- _ARCHIVE_NAME
- _ARCHIVE_PATH
- _CORRELATIONID
- _FTP_PASSWORD
- _FTP_USER
- _HTTP_PASSWORD
- _HTTP_PROXY_URL
- _HTTP_USER
- _MESSAGE_QUEUE

PACKAGE_TO_SHAREPOINT
> creates a package and sends it to a Microsoft SharePoint server. Variables that control this option include the following:

- _ARCHIVE_NAME
- _ARCHIVE_PATH
- _APPLIED_VIEWER_NAME
- _COLLECTION_FOLDER
- _COLLECTION_URL
- _DEBUG_FILE
- _GENERATED_NAME
- _HTTP_PASSWORD
- _HTTP_USER
- _IF_EXISTS
- _INITIALIZE_SITE
- _LIST_NAME
- _PARENT_FOLDER
- _PARENT_URL
- _SITE_URL
- _TARGET_VIEW_MIMETYPE
- _TARGET_VIEW_NAME
- _VIEWER_NAME

PACKAGE_TO_SUBSCRIBERS
creates a package and sends it to a subscriber channel. An actual archive package can be sent, or the package can be created in a public location and a reference URL can be sent. Variables that control this option include the following:

- _APPLIED_TEXT_VIEWER_NAME

- _APPLIED_VIEWER_NAME

- _ARCHIVE_NAME

- _ARCHIVE_PATH

- _CHANNEL

- _CHANNEL_STORE

- _COLLECTION_URL

- _CORRELATIONID

- _FOLDER_PATH

- _FROM

- _FTP_PASSWORD

- _FTP_USER

- _HTTP_PASSWORD

- _HTTP_PROXY_URL

- _HTTP_USER

- _IF_EXISTS

- _PARENT_URL

- _PROCESS_VIEWER

- _REPLYTO

- _SUBJECT

- _TARGET_VIEW_MIMETYPE

- _TARGET_VIEW_NAME

- _TEXT_VIEWER_NAME

- _VIEWER_NAME

PACKAGE_TO_WEBDAV
creates a package and sends it to a WebDAV-compliant server. Variables that control this option include the following:

- _ARCHIVE_NAME

- _ARCHIVE_PATH

- _COLLECTION_URL

- _GENERATED_NAME

- _HTTP_PASSWORD

- _HTTP_PROXY_URL

- _HTTP_TOKENAUTH

- _HTTP_USER

- ■ _IF_EXISTS
- ■ _PARENT_URL
- ■ _TARGET_VIEW_MIMETYPE
- ■ _TARGET_VIEW_NAME
- ■ _TEXT_VIEWER_NAME
- ■ _VIEWER_NAME

Almost all of these package option variables have directly equivalent properties in the package publishing API. For more information about these properties, see the PACKAGE_PUBLISH documentation in the *SAS Publishing Framework: Developer's Guide*. The property names are the same as the variable names with the underscore prefix removed.

# Using Output Parameters

Output parameters enable stored processes to return SAS macro variables upon successful execution, and to pass one or more values back to the client. Output parameters are used mainly with SAS BI Web Services and with stored processes that are called using the Stored Process Java API.

Output parameters are defined as part of the stored process metadata. Metadata for output parameters includes the following information: name, type, label, and an optional description. The name of the output parameter is the name of the SAS macro variable that is associated with the output parameter. The label specifies the output parameter name that is displayed to the user. The output parameter can be any of the following types: Date, Double, Integer, Time, TimeStamp, or String.

Recommended formats for each output parameter type are as follows:

*Table 2.2    Formats for Output Parameters*

| Output Parameter Type | Format |
|---|---|
| Date | DATE*w.* |
| Double | E*w.* |
| Integer | 11.0<br>**Note:** Integer values can range from -2,147,483,648 to 2,147,483,647. |
| Time | TIME*w.d* |
| Timestamp | DATETIME*w.d* |
| String | No format needed. |

For more information about any of these formats, see *SAS Formats and Informats: Reference*.

You can use the PUTN function to specify which numeric format you want the output parameter to have. The following example shows how to set the format for a timestamp type output parameter:

```
*  Input value from the prompt;
%let timestamp1 = 17OCT1991:14:45:32;


*  Format SAS timestamp value;
%let TimestampOut1 = %sysfunc(putn("&timestamp1"dt, DATETIME.));


%put TimestampOut1 parameter is &TimeStampOut1;
```

The following result is written to the SAS log:

```
    TimestampOut1 parameter is 17OCT91:14:45:32
```

The following example shows how to set the format for a time type output parameter:

```
*  Input value from the prompt;
%let time1 = 19:35;


*  Format SAS time value;
%let TimeOut1 = %sysfunc(putn("&time1"t, TIME8.));


%put TimeOut1 parameter is &TimeOut1;
```

The following result is written to the SAS log:

```
    TimeOut1 parameter is 19:35:00
```

The following example shows how to set the format for a date type output parameter:

```
*  Input value from the prompt;
%let date1 = 5Dec07;


*  Format SAS time value;
%let DateOut1 = %sysfunc(putn("&date1"d, DATE9.));


%put DateOut1 parameter is &DateOut1;
```

The following result is written to the SAS log:

```
    DateOut1 parameter is 05DEC2007
```

# Using Reserved Macro Variables

Many macro variables are reserved for special purposes in stored processes. Reserved names generally are prefixed with an underscore character. To avoid conflicts, do not use the underscore prefix for any application variables. Some reserved macro variables are created automatically for all stored processes that are running on a particular server. Some are created by specific stored process client or

middle-tier interfaces and are not created or available when other clients call the stored process.

The following table shows the reserved macro variables that can be used in your stored processes:

***Table 2.3*** *Reserved Macro Variables*

| Variable Name | Used By | Description |
| --- | --- | --- |
| _ABSTRACT | %STPBEGIN<br>%STPEND | Is the text string that briefly describes a package that was created by %STPBEGIN and %STPEND. |
| _ACTION | Web Clients | Specifies an action for the web application to take. Possible values for this variable are as follows:<br><br>BACKGROUND<br>    executes the stored process in the background.<br><br>DATA<br>    displays a summary of general stored process data.<br><br>EXECUTE<br>    executes the stored process.<br><br>FORM<br>    displays a custom input form if one exists. If FORM is the only value for _ACTION, and no form is found, then an error is generated. You can use the _FORM reserved macro variable in conjunction with _ACTION=FORM if you want to specify custom input forms.<br><br>INDEX<br>    displays a tree of all stored processes. For _ACTION=INDEX, three HTML frames are created with the top frame being the banner frame.<br><br>LOGOFF<br>    causes the web application to terminate the active session and to display a logoff screen.<br><br>NEWWINDOW<br>    displays results in a new window.<br><br>NOALERT<br>    suppresses generation of alerts. |

| Variable Name | Used By | Description |
| --- | --- | --- |
| | | NOBANNER<br>displays results without adding a banner. |
| | | PROPERTIES<br>displays the prompt page, which enables you to set input parameters and execution options and to execute the stored process. |
| _ACTION (cont'd.) | Web Clients | RECALL<br>can be combined with the PROPERTIES value to retain prompt values throughout a session. After you execute the stored process, you can go back to the input form and it will be populated with the prompt values that you previously specified. |
| | | SEARCH<br>displays a search page that enables you to search for a stored process or stored process report. You can combine this value with values for the _MATCH, _FIELD, _COLUMNS, _TYPE, or _PATH variables in order to specify search parameters. |
| | | TREE<br>displays a tree of all stored processes. You can combine this value with values for the _MATCH, _FIELD, _COLUMNS, _TYPE, or _PATH variables in order to specify display parameters for the tree view. |
| | | XML<br>can be combined with other _ACTION values to return XML data. You can use the following combinations: |
| | | _ACTION=TREE,XML<br>returns a stored process and stored process report tree list in XML. |
| | | _ACTION=DATA,XML<br>returns stored process data in XML. |
| | | _ACTION= PROPERTIES,XML<br>returns stored process prompts in XML. |

| Variable Name | Used By | Description |
|---|---|---|
| | | _ACTION=SEARCH,XML returns search results in XML. |
| _ACTION (cont'd.) | Web Clients | Values for _ACTION are case insensitive. Multiple values can be combined (except when using INDEX or DATA). Two common combinations are: |
| | | _ACTION= FORM,PROPERTIES displays a custom input form if one exists, otherwise displays the prompt page. |
| | | _ACTION=FORM,EXECUTE displays a custom input form if one exists, otherwise executes the stored process. |
| _ADDRESSLIST_DATASET_LIBN AME<br>_ADDRESSLIST_DATASET_ME MNAME<br>_ADDRESSLIST_VARIABLENAM E<br>_DATASET_OPTIONS | %STPBEGIN<br>%STPEND | Specifies a data set that contains email addresses when _RESULT is set to `PACKAGE_TO_EMAIL`. |
| _APPLIED_TEXT_VIEWER_NAM E<br>_APPLIED_VIEWER_NAME | %STPBEGIN<br>%STPEND | Specifies the name of the rendered package view when _RESULT is set to `PACKAGE_TO_SUBSCRIBERS` or `PACKAGE_TO_EMAIL`. |
| _APSLIST | Stored Process Server | Specifies the list of the names of all the parameters that were passed to the program. |
| _ARCHIVE_FULLPATH | %STPBEGIN<br>%STPEND | Specifies the full path and name of an archive package that was created by %STPEND when _RESULT is set to `PACKAGE_TO_ARCHIVE` or `PACKAGE_TO_REQUESTER`. This value is set by %STPEND and is an output value only. Setting it before setting %STPEND has no effect. |
| _ARCHIVE_NAME | %STPBEGIN<br>%STPEND | Specifies the name of the archive package to be created when _RESULT is set to `PACKAGE_TO_ARCHIVE`. If this |

| Variable Name | Used By | Description |
| --- | --- | --- |
| | | value is not specified or is blank and _RESULT is set to `PACKAGE_TO_ARCHIVE` or `PACKAGE_TO_REQUESTER`, then the package is created with a new, unique name in the directory that is specified by _ARCHIVE_PATH. This value is set through the Stored Process Service API and cannot be directly overridden by a client input parameter. |
| _ARCHIVE_PATH | %STPBEGIN %STPEND | Specifies the path of the archive package to be created when _RESULT is set to `PACKAGE_TO_ARCHIVE` or `PACKAGE_TO_REQUESTER`. This value is set through the Stored Process Java API and cannot be directly overridden by a client input parameter. The special value `TEMPFILE` causes the archive package to be created in a temporary directory that exists only until the stored process completes executing and the client disconnects from the server. |
| _AUTHTYP | Web Clients | Specifies the name of the authentication scheme that is used to identify a web client (for example, BASIC or SSL), or "null" (no authentication.) This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _BASEURL | Web Clients | Overrides the default value for _URL. This macro variable is used mainly with _REPLAY to return to the correct location. |
| _CHANNEL | %STPBEGIN %STPEND | Specifies a subscriber channel when _RESULT is set to `PACKAGE_TO_SUBSCRIBERS`. For more information about channel names, see PACKAGE_PUBLISH in the |

| Variable Name | Used By | Description |
| --- | --- | --- |
| | | *SAS Publishing Framework: Developer's Guide*. |
| _CLIENT | All | Identifies the client and version number, as follows: `_CLIENT=Client_Name; JVM java_version; operating_environment (operating_environment_architecture) operating_environment_version` The *Client_Name* is automatically obtained through the Stored Process Java API. If the API cannot obtain the value for *Client_Name*, then the default value is **StoredProcessService 9.4** (for example, `_CLIENT=StoredProcessService 9.4; JVM 1.6.0_25; Windows 7 (x86) 6.1)`. |
| _COLLECTION_FOLDER | %STPBEGIN %STPEND | Specifies both the parent folder and the collection folder together. This variable is relative to the list name when _RESULT is set to **PACKAGE_TO_SHAREPOINT**. |
| _COLLECTION_URL | %STPBEGIN %STPEND | Specifies the URL of the WebDAV collection to be created when _RESULT is set to **PACKAGE_TO_WEBDAV**. See also _IF_EXISTS. This value is set through the Stored Process Java API and cannot be directly overridden by a client input parameter. |
| _COLUMNS | Web Clients | Specifies which columns are to be displayed and the order in which they are presented. You can specify one or more of the following values: **description**, **keywords**, **createDate** (creation date), or **modDate** (modified date). The creation date and modified date are displayed as *year-month-day*. The default is the name column only. |
| _DEBUG | Web Clients | Specifies the debugging flags. For information about setting the |

| Variable Name | Used By | Description |
|---|---|---|
| | | default value of _DEBUG, see "Setting the Default Value of _DEBUG" on page 146. |
| _DEBUG_FILE | %STPBEGIN<br>%STPEND | Specifies the name of the file that contains debug wire trace output when _RESULT is set to `PACKAGE_TO_SHAREPOINT`. |
| _DESCRIPTION | %STPBEGIN<br>%STPEND | Descriptive text that is embedded in a package that was created by %STPBEGIN and %STPEND. |
| _DOMAIN | Web Clients | Specifies the authentication domain for the SAS Stored Process Web Application. |
| _EMAIL_ADDRESS | %STPBEGIN<br>%STPEND | Specifies destination email addresses when _RESULT is set to `PACKAGE_TO_EMAIL`. Multiple addresses can be specified using the multiple value convention for stored process parameters. |
| _ENCODING | %STPBEGIN<br>%STPEND | Sets the encoding for all ODS output. |
| _EXPIRATION_DATETIME | %STPBEGIN<br>%STPEND | Specifies the expiration datetime that is embedded in a package that was created by %STPBEGIN and %STPEND. Must be specified in a valid SAS datetime syntax. |
| _FIELD | Web Clients | Specifies which field to search for the match string. The possible fields are `name`, `description`, or `keywords`. The keywords field matches only complete keywords. The default field is the name field. |
| _FILE_DESCRIPTION | %STPBEGIN<br>%STPEND | Specifies the description that is to be used for all files that are added to a package. You can add the following values to the description: |

| Variable Name | Used By | Description |
|---|---|---|
| | | _FILENAME_<br>    specifies the name of the file that is being added to the package.<br><br>_DATETIME_<br>    specifies the date and time of package creation.<br><br>These values are case sensitive. For example, you can specify the following description:<br><br>`%let _FILE_DESCRIPTION= Date, Time,`<br>`    and Filename: _DATETIME_ _FILENAME_;` |
| _FMTSEARCH | PROC STP | Specifies which format catalogs need to be appended to the FMTSEARCH option in the stored process environment. |
| _FOLDER_PATH | %STPBEGIN<br>%STPEND | Specifies the folder path for the channel of interest when _RESULT is set to `PACKAGE_TO_SUBSCRIBERS`. |
| _FORM | Web Clients | Specifies the location of a custom input form JSP file for a stored process. This value is used only if _ACTION=FORM is also present. If the value starts with a slash (/), then the JSP file is assumed to be located relative to the SAS Stored Process Web Application root. Otherwise, it is assumed to be a complete URL and a redirect is performed to that value. |
| _FROM | %STPBEGIN<br>%STPEND | Specifies the name or email address of the sender when _RESULT is set to `PACKAGE_TO_EMAIL`. This value is the name or email address that the email appears to be from. |
| _GENERATED_NAME | %STPBEGIN<br>%STPEND | Returns the name of the package or the name of the folder that contains the package, whether this value was generated by SAS or specified by another variable. This variable is returned when _RESULT is set to `PACKAGE_TO_ARCHIVE`, |

| Variable Name | Used By | Description |
|---|---|---|
| | | `PACKAGE_TO_SHAREPOINT`, or `PACKAGE_TO_WEBDAV`. |
| _GOPT_DEVICE<br>_GOPT_HSIZE<br>_GOPT_VSIZE<br>_GOPT_XPIXELS<br>_GOPT_YPIXELS | %STPBEGIN<br>%STPEND | Sets the corresponding SAS/GRAPH option. For more information, see the DEVICE, HSIZE, VSIZE, XPIXELS, and YPIXELS options in "Graphics Options and Device Parameters Dictionary" in the *SAS/GRAPH: Reference* in SAS Help and Documentation. |
| _GOPTIONS | %STPBEGIN<br>%STPEND | Sets any SAS/GRAPH option that is documented in "Graphics Options and Device Parameters Dictionary" in the *SAS/GRAPH: Reference* in SAS Help and Documentation. You must specify the option name and its value in the syntax that is 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. |
| _GRAFLOC | Web Clients | Specifies the URL for the location of SAS/GRAPH applets. This variable is set to `/sasweb/graph` for most installations. |
| _HTACPT | Web Clients | Specifies the MIME types that are accepted by the stored process client. This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _HTCOOK | Web Clients | Specifies all of the cookie strings that the client sent with this request. This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _HTREFER | Web Clients | Specifies the address of the referring page. This variable is not set by default but can be enabled. For more information, |

| Variable Name | Used By | Description |
|---|---|---|
| | | see "Web Application Properties" on page 88. |
| _HTTP_PASSWORD | %STPBEGIN %STPEND | Specifies the password that is used (with _HTTP_USER) to access the WebDAV server when _RESULT is set to `PACKAGE_TO_WEBDAV`. This value is set through the Stored Process Java API and cannot be directly overridden by a client input parameter. |
| _HTTP_PROXY_URL | %STPBEGIN %STPEND | Specifies the Proxy server that is used to access the WebDAV server when _RESULT is set to `PACKAGE_TO_WEBDAV`. This value is set through the Stored Process Java API and cannot be directly overridden by a client input parameter. |
| _HTTP_TOKENAUTH | %STPBEGIN %STPEND | Enables token authentication to a SAS Content Server when _RESULT is set to `PACKAGE_TO_WEBDAV`. Value must be set to `TRUE`, `FALSE`, `YES`, or `NO`. Do not specify values for _HTTP_USER or _HTTP_PASSWORD if you are using this reserved macro variable. |
| _HTTP_USER | %STPBEGIN %STPEND | Specifies the user name that is used (with _HTTP_PASSWORD) to access the WebDAV server when _RESULT is set to `PACKAGE_TO_WEBDAV`. This value is set through the Stored Process Java API and cannot be directly overridden by a client input parameter. |
| _HTUA | Web Clients | Specifies the name of the user agent. This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _IF_EXISTS | %STPBEGIN | Can be NOREPLACE, UPDATE, or UPDATEANY. For more |

| Variable Name | Used By | Description |
| --- | --- | --- |
| | %STPEND | information, see the PACKAGE_PUBLISH options in the *SAS Publishing Framework: Developer's Guide*. |
| _INITIALIZE_SITE | %STPBEGIN %STPEND | Enables an administrator to initialize a SharePoint site when _RESULT is set to `PACKAGE_TO_SHAREPOINT`. |
| _LIST_NAME | %STPBEGIN %STPEND | Specifies a document library in the SharePoint site when _RESULT is set to `PACKAGE_TO_SHAREPOINT`. |
| _MATCH | Web Clients | Specifies a search string. If no string is specified, then everything is a match. |
| _MESSAGE_QUEUE | %STPBEGIN %STPEND | Specifies a target queue when _RESULT is set to `PACKAGE_TO_QUEUE`. For more information about queue names, see the PACKAGE_PUBLISH documentation in the *SAS Publishing Framework: Developer's Guide*. Multiple queues can be specified using the multiple value convention for stored process parameters. |
| _METAFOLDER | All | Contains the name or path of the folder for the stored process that is being executed. For example, for the stored process: `_PROGRAM=/Sales/Southwest/ Quarterly Summary` the value of _METAFOLDER would be: `_METAFOLDER=/Sales/Southwest/` . |
| _METAPERSON | All | Specifies the person metadata name that is associated with the _METAUSER login variable. The value of this variable can be `UNKNOWN`. This variable cannot be modified by the client. |
| _METAUSER | All | Specifies the login user name that is used to connect to the |

| Variable Name | Used By | Description |
|---|---|---|
| | | metadata server. This variable cannot be modified by the client. |
| _MSOFFICECLIENT | SAS Add-In for Microsoft Office | Specifies the Microsoft application that is currently executing the stored process. Valid values for this macro variable are `Excel`, `Word`, `PowerPoint`, and `Outlook`. |
| _NAMESPACES | %STPBEGIN %STPEND | Applies to packages only. For more information about this variable, see the PACKAGE_BEGIN documentation in the *SAS Publishing Framework: Developer's Guide*. |
| _NAMEVALUE | %STPBEGIN %STPEND | Specifies a list of one or more name or value pairs that are used for filtering when generating packages. For more information about this variable, see the PACKAGE_BEGIN documentation in the *SAS Publishing Framework: Developer's Guide*. |
| _ODSDEST | %STPBEGIN %STPEND | Specifies the ODS destination. The default ODS destination is HTML if _ODSDEST is not specified. Valid values of _ODSDEST include the following:<br>▪ CSV<br>▪ CSVALL<br>▪ TAGSETS.CSVBYLINE<br>▪ HTML<br>▪ LATEX<br>▪ NONE (no ODS output is generated)<br>▪ PDF<br>▪ PS<br>▪ RTF<br>▪ SASREPORT<br>▪ WML<br>▪ XML |

| Variable Name | Used By | Description |
| --- | --- | --- |
| | | ■ any tagset destination |
| _ODSDOC | %STPBEGIN %STPEND | Contains the two-level name of the ODS Document file that was created by the STP procedure. |
| _ODSOPTIONS | %STPBEGIN %STPEND | Specifies options that are to be appended to the ODS statement. Do not use this macro to override options that are defined by a specific macro variable. For example, do not specify ENCODING=*value* in this variable because it conflicts with _ODSENCODING. |
| | | Note that NOGTITLE and NOGFOOTNOTE are appended to the ODS statement as default options. You can override this behavior by specifying GTITLE or GFOOTNOTE for _ODSOPTIONS. |
| _ODSSTYLE | %STPBEGIN %STPEND | Sets the ODS STYLE= option. You can specify any ODS style that is valid on your system. |
| _ODSSTYLESHEET | %STPBEGIN %STPEND | Sets the ODS STYLEHEET= option. To store a generated style sheet in a catalog entry and automatically replay it by using the SAS Stored Process Web Application, specify `myfile.css (url="myfile.css")`. |
| _PARENT_FOLDER | %STPBEGIN %STPEND | Specifies the parent folder of a generated collection name when _RESULT is set to **PACKAGE_TO_SHAREPOINT**. This variable is relative to the list name. |
| _PATH | Web Clients | Specifies the starting level for the _ACTION=INDEX display. The value of _PATH is a folder name, such as `/Sales/Southwest` . If this variable is used with _ACTION=SEARCH, then |

| Variable Name | Used By | Description |
|---|---|---|
| | | _PATH limits searching to the specified path and below. |
| _PROCESS_VIEWER | %STPBEGIN %STPEND | Specifies a character string of `yes` to indicate that the rendered view is delivered in email when _RESULT is set to `PACKAGE_TO_EMAIL` or `PACKAGE_TO_SUBSCRIBERS`. |
| _PROGRAM | All | Specifies the name of the stored process. The value of _PROGRAM is a path, such as `/Sales/Southwest/Quarterly Summary`. |
| _QRYSTR | Web Clients | Specifies the query string that is contained in the request URL after the path. This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _REPLAY | Stored Process Server Web Clients | Specifies a complete URL for use with programs that use the Output Delivery System (ODS). This URL consists of the values of _THISSESSION and _TMPCAT. ODS uses this URL to create links that replay stored output when they are loaded by the user's web browser. This variable is created by the stored process server and is not one of the symbols that is passed from the SAS Stored Process Web Application. The _REPLAY variable is set only if the _URL variable is passed in from the client or middle tier.<br><br>If you are using the _REPLAY macro variable with Microsoft Office, then you build a URL that uses the _OUTPUTAPP= parameter. Supported values for the _OUTPUTAPP= parameter include EXCEL, WORD, and POWERPOINT. For example, if you specify _OUTPUTAPP=EXCEL in the URL, then the content type for |

| Variable Name | Used By | Description |
|---|---|---|
| | | the replayed output is `application/vnd.ms-excel`. |
| | | If you need to specify the name of the file that the _REPLAY macro variable returns, then you can use the _CONTDISP parameter in the URL. The value of this parameter is echoed back as a Content-disposition header. |
| _REPLYTO | %STPBEGIN %STPEND | Specifies a designated email address to which package recipients might respond when _RESULT is set to `PACKAGE_TO_EMAIL`. |
| _REPORT | All | Specifies the name of the stored process report. The value of _REPORT is a path, such as `/myfolder/myreport` . |
| _REPORTID | All | Specifies the ID of the generation of the stored process report. The value of _REPORTID is a number, such as 3. |
| _REPOSITORY | Web Clients | Specifies the metadata repository where the stored process is registered. The default repository value is `METASERVER` if _REPOSITORY is not specified. |
| _REQENCODING | Web Clients | Specifies the servlet input encoding. The default encoding is `UTF-8`. |
| _REQMETH | Web Clients | Specifies the name of the HTTP method with which this request was made (for example, GET, POST, or PUT). This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _RESULT | All | Specifies the type of client result that is to be created by the stored process. For more information, see "Setting Result |

| Variable Name | Used By | Description |
|---|---|---|
| | | Capabilities" on page 16. Possible values for this variable are as follows: |

STATUS
    no output to the client.

STREAM
    output is streamed to the client through _WEBOUT fileref.

PACKAGE_TO_ARCHIVE
    package is published to an archive file.

PACKAGE_TO_REQUESTER
    package is returned to the client. The package can also be published to an archive file in this case. This option is valid only on the workspace server, and only for stored processes that are compatible with 9.2.

PACKAGE_TO_WEBDAV
    package is published to a WebDAV server.

PACKAGE
    transient package is returned to the client. This applies only to the SAS Stored Process Web Application, and can be used as a shortcut for specifying `_RESULT=PACKAGE_TO_ARCHIVE` and `_ARCHIVE_PATH=TEMPFILE` in the URL.

The _RESULT value is set through the Stored Process Service API and cannot be directly overridden by a client input parameter. The value can be overridden in the stored process program to use these additional values:

PACKAGE_TO_EMAIL
    package published to one or more email addresses.

PACKAGE_TO_QUEUE
    package published to a message queue.

PACKAGE_TO_SHAREPOINT
    package published to SharePoint.

PACKAGE_TO_SUBSCRIBERS
    package published to a subscriber channel.

| Variable Name | Used By | Description |
|---|---|---|
| _RESULT (cont'd.) | All | For more information about these options, see "Using the %STPBEGIN and %STPEND Macros" on page 18. |
| _RMTADDR | Web Clients | Specifies the Internet Protocol (IP) address of the client that sent the request. For many installations with a firewall between the client and the web server or servlet container, this value is the firewall address instead of the web browser client. This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _RMTHOST | Web Clients | Specifies the fully qualified name of the client that sent the request, or the IP address of the client if the name cannot be determined. For many installations with a firewall between the client and the web server or servlet container, this value is the firewall name instead of the web browser client. This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _RMTUSER | Web Clients | Specifies the login ID of the user making this request if the user has been authenticated, or indicates null if the user has not been authenticated. This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _SECUREUSERNAME | Web Clients | Contains the value for the user name that is obtained from web client authentication. The _SECUREUSERNAME macro variable is created when the application server executes a stored process. The value for |

| Variable Name | Used By | Description |
|---|---|---|
| | | _SECUREUSERNAME is written into the _username macro variable if _username does not already contain a value. |
| _SENDER | %STPBEGIN %STPEND | Specifies the email address of the sender when _RESULT is set to `PACKAGE_TO_EMAIL`. A valid email address should be specified. This address receives any bounced or undeliverable email. This value is the actual email address that the email is sent from. |
| _SESSIONID | Stored Process Server | Specifies a unique identifier for the session. The _SESSIONID variable is created only if a session has been explicitly created. |
| _SITE_URL | %STPBEGIN %STPEND | Specifies the transfer protocol (HTTP or HTTPS), the host name, and the SharePoint site when _RESULT is set to `PACKAGE_TO_SHAREPOINT`. |
| _SRVNAME | Web Clients | Specifies the host name of the server that received the request. |
| _SRVPORT | Web Clients | Specifies the port number on which this request was received. |
| _SRVPROT | Web Clients | Specifies the name and version of the protocol that the request uses in the form protocol/majorVersion.minorVersion (for example, HTTP/1.1). This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |
| _SRVSOFT | Web Clients | Identifies the web server software. This variable is not set by default but can be enabled. For more information, see "Web Application Properties" on page 88. |

| Variable Name | Used By | Description |
|---|---|---|
| _STATUS_MESSAGE | Web Clients | Returns the value of the SAS macro variable to the client after the stored process has been executed. This macro variable is useful for returning debugging information or informational messages (for example, when packages are created that are not displayed). |
| _STPERROR | %STPBEGIN %STPEND | Specifies a global error variable. This variable is set to `0` if %STPBEGIN and %STPEND complete successfully. This variable is set to a nonzero numeric value if an error occurs. |
| _STPWORK | %STPBEGIN %STPEND | Specifies a temporary working directory to hold files that are published in a package. This variable is set by %STPBEGIN and is not modified by the stored process. |
| _SUBJECT | %STPBEGIN %STPEND | Specifies a subject line when _RESULT is set to `PACKAGE_TO_EMAIL`. |
| _SUPPRESS_MVARS | Stored Process Server | Hides all macro variables and their values from the SAS log when set to `yes` or `y`. |
| _TARGET | Web Clients | Specifies the fixed HTML form target value to use in displays that are generated by the SAS Stored Process Web Application. Use `_blank` to always force a new window. |
| _TARGET_VIEW_MIMETYPE | %STPBEGIN %STPEND | Specifies the MIME type of the rendered view when _RESULT is set to `PACKAGE_TO_EMAIL`, `PACKAGE_TO_SHAREPOINT`, `PACKAGE_TO_SUBSCRIBERS`, or `PACKAGE_TO_WEBDAV` |
| _TARGET_VIEW_NAME | %STPBEGIN %STPEND | Specifies the name of the rendered view when _RESULT is set to `PACKAGE_TO_EMAIL`, `PACKAGE_TO_SHAREPOINT`, |

| Variable Name | Used By | Description |
|---|---|---|
| | | `PACKAGE_TO_SUBSCRIBERS`, or `PACKAGE_TO_WEBDAV` |
| _TEXT_VIEWER_NAME | %STPBEGIN %STPEND | Specifies the name of a text viewer template that formats package content for viewing in email when _RESULT is set to `PACKAGE_TO_EMAIL`, `PACKAGE_TO_SUBSCRIBERS`, or `PACKAGE_TO_WEBDAV`. |
| _THISSESSION | Stored Process Server Web Clients | Specifies a URL that is composed from the values of _URL and _SESSIONID. This variable is created by the stored process server and is used as the base URL for all URL references to the current session. The _THISSESSION variable is created only if the _URL variable is passed in and a session has been explicitly created. |
| _TIMEZONE | %STPBEGIN %STPEND | Sets the TIMEZONE SAS system option. For more information about this system option and valid values, see *SAS System Options: Reference*. |
| _TMPCAT | Stored Process Server | Specifies a unique, temporary catalog name. This catalog can be used to store temporary entries to be retrieved later. In socket servers, the _TMPCAT catalog is deleted after a number of minutes that are specified in the variable _EXPIRE. This variable is created by the stored process server and is not one of the symbols that is passed from the SAS Stored Process Web Application. |
| _TYPE | Web Clients | Indicates what type of object to search for. You can specify `storedprocess` or `report` (stored process report) to limit the search to one of these object |

| Variable Name | Used By | Description |
|---|---|---|
| | | types. The default is to search for both. |
| _URL | Web Clients | Specifies the URL of the web server middle tier that is used to access the stored process. |
| _USERLOCALE | Web Clients | Contains the locale for the user that was set in the user preferences. If this value was not set, it contains the locale sent in the HTTP request Accept-Language header. |
| _username | Web Clients | Specifies the value for the user name that is obtained from web client authentication. |
| _VERSION | Web Clients | Specifies the SAS Stored Process Web Application version and build number. |
| _VIEWER_NAME | %STPBEGIN<br>%STPEND | Specifies the name of the HTML viewer template to be applied when _RESULT is set to `PACKAGE_TO_EMAIL`, `PACKAGE_TO_SHAREPOINT`, `PACKAGE_TO_SUBSCRIBERS`, or `PACKAGE_TO_WEBDAV`. |
| _WELCOME | Web Clients | Specifies an initial page to display in the web application. If the value starts with a slash (/), then the Welcome page is relative to the web application root context, and the web browser is forwarded to that page. Otherwise, a redirect command is sent to the web browser for the specified page. |
| SYSPROCESSMODE | All | returns the server type that the stored process uses to execute. For stored processes, one of the following values is returned:<br><br>■ `SAS Workspace Server`<br><br>■ `SAS Pooled Workspace Server`<br><br>■ `SAS Stored Process Server`<br><br>See "SYSPROCESSMODE Automatic Macro Variable" in |

| Variable Name | Used By | Description |
|---|---|---|
| | | *SAS Macro Language: Reference* for more information about using the automatic macro variable SYSPROCESSMODE. |

Most of the reserved macro variables that are related to package publishing have an equivalent property or parameter in SAS Publishing Framework. For a description of these variables, see the documentation for PACKAGE_PUBLISH and PACKAGE_BEGIN in the *SAS Publishing Framework: Developer's Guide*.

# Using Sessions

## Overview of Sessions

The web is a stateless environment. A client request to a server does not know about preceding requests. The web is a simple environment for client and server developers, but it is difficult for application programmers. Often, programmers want to carry information from one request to the next. This is known as maintaining state. Sessions provide a convenient way to maintain state across multiple stored process requests.

A session is the data that is saved from one stored process execution to the next. The session data consists of macro variables and library members (data sets and catalogs) that the stored process has explicitly saved. The session data is scoped so that all users have independent sessions. For more information, see "Using Sessions in a Sample Web Application " on page 134.

## Creating a Session

The stored process must explicitly create a session with the STPSRV_SESSION function, as follows:

In macro:

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

In DATA step or SCL:

```
rc=stpsrv_session('create');
```

Creating a session sets the global macro variables _SESSIONID and _THISSESSION and creates the SAVE session library.

# Using the Session

A session saves all global macro variables whose names begin with SAVE_. For example, the following statements cause the macro variable `save_mytext` to be available in subsequent stored processes that share the same session:

```
%global save_mytext;
%let save_mytext="Text to be saved
for the life of the session";
```

Data sets and catalogs can also be saved across program requests using the SAVE library. Data sets and catalogs that are created in or copied to this library are available to all future stored processes that execute in the same session.

Creating a session causes the automatic variables _THISSESSION and _SESSIONID to be set. Sample values for these variables are as follows:

```
%let rc=%sysfunc(stpsrv_session(create));
%put _SESSIONID=&_SESSIONID;
_SESSIONID=7CF645EB-6E23-4853-8042-BBEA7F866B55
%put _THISSESSION=&_THISSESSION;
_THISSESSION=/SASStoredProcess/do?_sessionid=
   7CF645EB-6E23-4853-8042-BBEA7F866B55
```

These variables can be used to construct URLs or HTML forms that execute another stored process in the same session. For example:

```
%let rc=%sysfunc(stpsrv_session(create));
data _null;
file _webout;
put '<HTML>';
put '<BODY>';
put '<H1>Session Test Page</H1>';

/* Link to another stored process in the same session */
put '<A HREF="' "&_THISSESSION"
'&_PROGRAM=/Test/Test2">Test</A>';
put '</BODY>';
put '</HTML>';
run;
```

**Note:** The _THISSESSION variable is not identical to the _THISSESSION variable that is used in SAS/IntrNet. If you are converting an existing SAS/IntrNet program to a stored process, any references to `symget('_THISSESSION')` should generally be replaced with `"&_THISSESSION"`. For more information, see Appendix 2, "Converting SAS/IntrNet Programs to SAS Stored Processes," on page 185.

# Deleting the Session

Sessions expire after a period of inactivity. The default expiration time is 15 minutes. The expiration time can be changed using the STPSRVSET function, as follows where the time-out is specified in seconds:

In macro:

```
%let rc=%sysfunc(stpsrvset(session timeout,300));
```

In DATA step or SCL:

```
rc=stpsrvset('session timeout',300);
```

If the session is not accessed for the length of the time-out, the server deletes the session, the associated SAVE library, and all associated macro values. Any further attempts to access the session result in an invalid or expired session error.

Sessions can be explicitly destroyed using the STPSRV_SESSION function, as follows:

In macro:

```
%let rc=%sysfunc(stpsrv_session(delete));
```

In DATA step or SCL:

```
rc=stpsrv_session('delete');
```

Submitting this code does not immediately delete the session. The session is marked for deletion only at the completion of the stored process. For this reason, a stored process cannot delete a session and create a new session.

# Limitations

Stored process sessions are supported only by the stored process server. Stored processes that execute on a workspace server cannot create or access sessions.

A session exists in the server process where it was created. All stored processes that access that session must execute in the same server process. Load balancing and other execution dispatching features are typically ignored when using sessions that might have an impact on application performance and scalability. Sessions are not recommended for applications with small amounts of state information; use a client-based method for maintaining state instead.

# 3

# Stored Process Server Functions

# Using Stored Process Server Functions

Stored process server functions are DATA step functions that you use to define character, numeric, and alphanumeric strings to generate output in the desired format. The SAS Stored Process Server functions can be used to return the correct character, numeric, or alphanumeric value of a parameter setting.

**Note:** You can also use APPSRV syntax from the Application Dispatcher in place of these functions. For more information, see the SAS/IntrNet: Application Dispatcher documentation.

# Dictionary

# STPSRVGETC Function

Returns the character value of a server property.

Category: Character

## Syntax

**STPSRVGETC**(*valuecode*)

## Required Argument

***valuecode***
    is the character string name of the property.

## Details

The STPSRVGETC function takes one character string property and returns a character string result.

......................................................................................................

**Note:** The APPSRVGETC function can be used instead of STPSRVGETC. This feature is provided in order to enable you to convert existing SAS/IntrNet programs to stored processes.

......................................................................................................

## Example

*Table 3.1   Sample Statements and Results*

| SAS Statements | Results |
| --- | --- |
| `sencoding=stpsrvgetc('Default Output Encoding'); put sencoding=;` | `sencoding=WLATIN1` |
| `version=stpsrvgetc('version'); put version=;` | `version=SAS Stored Processes Version 9.4 (Build 330)` |

# STPSRVGETN Function

Returns the numeric value of a server property.

Category: Numeric

## Syntax

**STPSRVGETN**(*valuecode*)

## Required Argument

**valuecode**
is the character string name of the property.

## Details

The STPSRVGETN function takes one character string property and returns a numeric string result.

Note: The APPSRVGETN function can be used instead of STPSRVGETN. This feature is provided in order to enable you to convert existing SAS/IntrNet programs to stored processes.

## Example

*Table 3.2    Sample Statements and Results*

| SAS Statements | Results |
|---|---|
| `dsesstimeout=stpsrvgetn('default session timeout');`<br>`put dsesstimeout=;` | `dsesstimeout=900` |
| `sessmaxtimeout=stpsrvgetn('maximum session timeout');`<br>`put sessmaxtimeout=;` | `sessmaxtimeout=3600` |
| `session=stpsrvgetn('session timeout');`<br>`put session=;` | `session=900` |

| SAS Statements | Results |
|---|---|
| ```maxconreqs=stpsrvgetn('maximum`<br>` concurrent requests');`<br>`put maxconreqs=;``` | `maxconreqs=1` |
| ```deflrecl=stpsrvgetn('default`<br>` output lrecl');`<br>`put deflrecl=;``` | `deflrecl=65535` |
| ```version=stpsrvgetn('version');`<br>`put version=;``` | `version=9.4` |

# STPSRVSET Function

Sets the value of a server session time-out.

Category:     Character

## Syntax

**STPSRVSET**(SESSION TIMEOUT, *newvalue*)

### Required Arguments

**SESSION TIMEOUT**
    is the character string name of the property that specifies that the server session time-out value is being set.

*newvalue*
    specifies the number of seconds that elapse before a session expires. The default session time-out is 900 (15 minutes).

## Details

The STPSRVSET function takes one character string property (`SESSION TIMEOUT`) and one numeric string property (the number of seconds before the session expires) and returns a numeric string result. The return code is zero for success, nonzero for failure.

**Note:** The APPSRVSET function can be used instead of STPSRVSET. This feature is provided in order to enable you to convert existing SAS/IntrNet programs to stored processes. The following Application Dispatcher properties are not supported

by the SAS Stored Process Server: REQUIRE COOKIE, REQUEST TIMEOUT, and AUTOMATIC HEADERS.

## Example

This example shows how to set the session time-out to 900 seconds:

```
rc=stpsrvset('session timeout',900);
```

# STPSRV_HEADER Function

Adds or modifies a header.

Category:        Character

## Syntax

**STPSRV_HEADER**(*Header Name*,*Header Value*)

### Required Arguments

***Header Name***
    is the name of the header to set or reset.

***Header Value***
    is the new value for the header.

## Details

The STPSRV_HEADER function enables automatic header generation. You can add a header to the default list or modify an existing header from the list. When you modify the value of an existing header, the old value becomes the return value of the function.

The automatic HTTP header generation feature recognizes Output Delivery System (ODS) output types and generates appropriate default content-type headers. If no content type is specified with STPSRV_HEADER, then ODS is not used and no HTTP header is written to _WEBOUT. A default `Content-type: text/html` header is generated. For a list of commonly used HTTP headers, see "Using HTTP Headers" on page 120.

**Note:** The APPSRV_HEADER function can be used instead of STPSRV_HEADER. This feature is provided in order to enable you to convert existing SAS/IntrNet programs to stored processes.

## Example

*Table 3.3* *Sample Statements and Resulting Headers*

| SAS Statements | Resulting Headers |
| --- | --- |
| *No calls to stpsrv_header* | Content-type: text/html |
| ```/* add expires header */```<br>```rc = stpsrv_header('Expires','Thu,```<br>``` 18 Nov 1999 12:23:34 GMT');``` | Content-type: text/html<br>Expires: Thu, 18 Nov 1999<br> 12:23:34 GMT |
| ```/* add expires header */```<br>```rc = stpsrv_header('Expires','Thu,```<br>``` 18 Nov 1999 12:23:34 GMT');```<br>```/* add pragma header*/```<br>```rc = stpsrv_header('Cache-control',```<br>``` 'no-cache');``` | Content-type: text/html<br>Expires: Thu, 18 Nov 1999<br> 12:23:34 GMT<br>Cache-control: no-cache |
| ```/* add expires header */```<br>```rc = stpsrv_header('Expires','Thu,```<br>``` 18 Nov 1999 12:23:34 GMT');```<br>```/* add pragma header*/```<br>```rc = stpsrv_header('Cache-control',```<br>``` 'no-cache');```<br>```...```<br>```/* remove expires header, rc```<br>``` contains old value */```<br>```rc = stpsrv_header('Expires','');``` | Content-type: text/html<br>Cache-control: no-cache |

# STPSRV_SESSION Function

Creates or deletes a session.

Category:          Character

## Syntax

**STPSRV_SESSION**('*command*',<*timeout*>)

## Required Argument

**command**
is the command to be performed. Allowed values are `CREATE` and `DELETE`.

## Optional Argument

**timeout**
is the optional session time-out in seconds. This property is valid only when you specify the value `CREATE` for the command property.

# Details

The STPSRV_SESSION function creates or deletes a session. The function returns zero for a successful completion. A nonzero return value indicates an error condition.

**Note:** The APPSRV_SESSION function can be used instead of STPSRV_SESSION. This feature is provided in order to enable you to convert existing SAS/IntrNet programs to stored processes.

# Example

*Table 3.4   Sample Statements*

| SAS Statements |
| --- |
| `rc=stpsrv_session('create', 600);` |
| `rc=stpsrv_session('delete');` |

# STPSRV_UNQUOTE2 Function

Unmasks quotation mark characters in an input parameter.

Category:          Character

# Syntax

**STPSRV_UNQUOTE2**(*paramname*)

## Required Argument

***paramname***
> is the character string name of the parameter.

## Details

The STPSRV_UNQUOTE2 CALL routine takes the name of an input parameter (or any global macro variable) and unmasks matched pairs of single or double quotation marks. The CALL routine does not return a value. Instead, it modifies the specified macro variable. This CALL routine can be used to selectively remove quotation marks from stored process input parameters so that they can be used in statements that require quotation marks.

## Example

This CALL routine is typically called with %SYSCALL in open macro code, as follows:

```
/* MYOPTIONS is an input parameter and might contain quotation
marks, for example: dashline='c000000000000000'x */
%SYSCALL STPSRV_UNQUOTE2(MYOPTIONS);

/* Quote characters are now interpreted as expected */
goptions &MYOPTIONS;
...
```

# 4

# Managing Stored Process Metadata

# Choosing or Defining a Server

## Types of Servers That Host Stored Processes

You must choose a server (for stored processes that are compatible with 9.2) or application server context to host your stored process. Servers are defined in metadata and are actually logical server definitions that can represent one or more physical server processes. There are many options, including pre-started servers, servers that are started on demand, and servers that are distributed across multiple hardware systems. You can use the Server Manager in SAS Management Console to create or modify server definitions. For more information about server

configurations, see the *SAS Intelligence Platform: Application Server Administration Guide*.

Because the logical server description in metadata hides the server implementation details, a stored process can be moved to or associated with any appropriate server without modifying the stored process. Moving a stored process from one server to another requires changing only the metadata association and moving the source code, if necessary. A stored process is the combination of a SAS program, the server that hosts that program, and the metadata that describes and associates the two. For stored processes that are compatible with 9.2, it is not possible to create a stored process that is associated with more than one server, although it is possible to create stored processes that share the same SAS program or source file. Starting with 9.3, stored processes can be run from multiple application servers.

Stored processes can be hosted by two types of servers: SAS Stored Process Servers and SAS Workspace Servers. The two servers are similar, but they have slightly different capabilities and they are targeted at different use cases.

Starting with 9.3, stored processes are associated with an application server context instead of a specific logical server. The application server context defines the environment in which the stored process executes. Application server contexts typically contain multiple server definitions. Stored processes can be executed on several server types. The server type is selected at run-time based on client preferences and constraints defined in the stored process metadata. You can choose whether to restrict the stored process to run on a stored process server only or on a workspace server only, or you can choose to allow the client application to run the stored process on the default server type. In this case, the stored process server is used unless the client application specifies to use the workspace server.

---

**Note:** You can use the SYSPROCESSMODE automatic macro variable to find out which server type the stored process is using to execute. See"SYSPROCESSMODE Automatic Macro Variable" in *SAS Macro Language: Reference*for more information about using the automatic macro variable SYSPROCESSMODE.

---

# SAS Stored Process Server

The SAS Stored Process Server is a multi-user server. A single server process can be shared by many clients. The recommended load-balancing configuration enables client requests to be serviced by multiple server processes across one or more hardware systems. This approach provides a high-performance, scalable server, but it imposes some restrictions. Because the same server handles requests from multiple users, it cannot easily impersonate a user to perform security checks. By default, the server runs under a single, shared user identity (defined in metadata) for all requests. All security checks based on client identity must be performed in the stored process. For more information about stored process server security, see the *SAS Intelligence Platform: Application Server Administration Guide*.

The stored process server implements some features that are not available on the workspace server, including replay (such as graphics in streaming output) and sessions (see "Using Sessions" on page 47).

## SAS Workspace Server

The SAS Workspace Server is a single-user server. A new server process is started for each client, then terminated after the stored process completes execution. This approach is not as scalable as the load-balanced stored process server, but it has a major security advantage. Each server is started under the client user identity and is subject to host operating environment permissions and rights for that client user. The workspace server also provides additional functionality, including data access and execution of client-submitted SAS code. For more information about workspace server security, see the *SAS Intelligence Platform: Application Server Administration Guide*.

Starting with 9.3, the workspace server supports streaming output and web services. If you use the workspace server to execute a stored process with streaming output and static images are missing from the output, you can try making any of the following changes:

- Change the graph format to ActiveX or Java.

- Change the result format to PDF or RTF.

- Update the stored process to produce package output.

**Note:** Information map stored processes are supported only on the workspace server.

The pooled workspace server is very similar to the standard workspace server, but it provides higher performance by reusing existing server instances instead of starting a new server for each client. Pooled workspace servers must run under group identities to allow reuse across groups of users.

# Using Source Code Repositories

Starting with 9.3, stored process source code can be stored in metadata. If the code is not stored in metadata, then stored processes are stored in external files with a `.sas` extension. The `.sas` file must reside in a directory that is registered with the server that executes the stored process. These directories are known as *source code repositories*. Source code repositories are managed using the New Stored Process wizard or the Stored Process Properties dialog box in SAS Management Console. After you choose a server for your stored process in the New Stored Process wizard or in the Stored Process Properties dialog box, you can choose whether to store the source code in metadata (starting with 9.3) or on the server. If you store the source code on the server, you are presented with a list of available source code repositories. You can choose an existing source code repository or click **Manage** to add or modify source code repositories.

For z/OS, the program can be contained in an HFS `.sas` file or in a member of a partitioned data set (PDS). Source code repositories can be either HFS directories or a partitioned data set.

# Registering the Stored Process Metadata

After you write the stored process and define or choose a server, you must register the metadata by using the New Stored Process wizard in SAS Management Console. (SAS Enterprise Guide users can perform the same steps within the SAS Enterprise Guide application.) The New Stored Process wizard can be used to create new stored processes, or you can use the Stored Process Properties dialog box in SAS Management Console to modify existing stored processes. You can specify and manage the following information for stored processes:

**Folder**
specifies the metadata location where the stored process is registered. The folders are defined in metadata and do not correspond to any physical location. The folder hierarchies that are used for stored processes can also hold other objects such as SAS reports, information maps, and administrative metadata. You can create and modify folders using SAS Management Console.

**Name**
specifies the stored process name, which acts as both a display label and as part of the URI for the stored process.

**Description**
specifies an optional text description of the stored process.

**Keywords**
specifies an optional list of keywords to associate with the stored process. Keywords are arbitrary text strings that are typically used for searching or to indicate specific capabilities. For example, the keyword *XMLA Web Service* is used to indicate a stored process that can be executed by SAS BI Web Services by using the XMLA calling convention.

**Responsibilities**
specifies one or more users who are responsible for the stored process. This information is optional.

**Hide from user**
enables you to hide the stored process from the folder view and search results in a client application. The stored process is hidden only in clients that support this feature. The stored process is hidden from all users, even unrestricted users. The stored process is not hidden in SAS Management Console.

**SAS server** or **Application server**
specifies the server or application server context, respectively, that executes the stored process. For more information, see "Choosing or Defining a Server" on page 59.

**Server type** (available starting with 9.3)
specifies the type of server that runs the stored process that you are defining. If you choose **Default server**, then either a SAS Stored Process Server or a SAS Workspace Server can be used, depending on which one is available to the client running the stored process. If this option is selected, then the stored process server is used unless the client application specifies to use a workspace server. Select **Stored process server only** if the stored process uses sessions or if it uses replay (for example, to produce graphics in streaming output). Select

**Workspace server only** if the stored process must be run under the client identity.

**Source location and execution** (available starting with 9.3)
specifies whether the stored process can be executed on other application servers or only on the selected application server, and where the source code is stored. If you allow the stored process to be executed on other application servers, then the source code is stored in metadata. If you allow execution on the selected application server only, then you must specify whether the source code is stored in metadata or on the application server. If the source code is stored on the application server, then you must specify the source code repository and source file.

**Source Code Repository** and **Source File**
specifies the directory and source file that contain the stored process. For more information, see "Using Source Code Repositories" on page 61.

**Edit Source Code** (or **Edit**)
enables you to add or edit the source code for the stored process.

**Result capabilities** (or **Results**)
specifies the type of output that the stored process can produce. For more information, see "Setting Result Capabilities" on page 16.

**Parameters**
specifies input parameters or output parameters for the stored process. Parameters are optional. For more information, see "Using Input Parameters" on page 8 or "Using Output Parameters" on page 25.

**Data Sources** and **Data Targets**
specifies an optional list of data sources and data targets for the stored process. Streams can be used to send data that is too large to be passed in parameters between the client and the executing stored process. Definitions for data sources and data targets can also include an XML schema specification or a data table.

**Authorization**
specifies access controls for the stored process. Currently only the *ReadMetadata* and *WriteMetadata* permissions are honored. A user must have *ReadMetadata* permission to execute the stored process. *WriteMetadata* permission is required to modify the stored process definition.

You cannot specify authorization information from the New Stored Process wizard. To specify authorization information, you must open the Stored Process Properties dialog box for an existing stored process.

This metadata is stored on the SAS Metadata Server so that it can be accessed by client applications. For more information about using the New Stored Process wizard or the Stored Process Properties dialog box to create and maintain the metadata defining a stored process, see the Help in SAS Management Console.

**Note:** Starting with SAS 9.2, you can register and modify the metadata for stored processes programmatically by using a Java API.

# Developing Stored Processes with Package Results

## Overview

Prior to SAS 9.2, when the user created a stored process by using SAS Management Console and chose to create permanent package results, there was a Permanent Package Details dialog box for providing the required information.

Starting with SAS 9.2, the Permanent Package Details dialog box has been removed and the user is now responsible for providing this information by using the new prompt framework instead. A set of shared prompt groups that contain this information has been created for convenience.

If you have stored processes that were created in SAS 9.1.3 with package results, you can migrate or promote these to SAS 9.4. Any necessary hidden prompts are created automatically by the import process.

## Create Permanent Package Results

To create a stored process with a permanent result package, perform the following steps:

1   In the New Stored Process wizard, when you are defining the stored process, select the **Package** check box to specify result capabilities for the stored process.

2   On the Parameters page of the New Stored Process wizard, click **Add Shared** to load one of the predefined shared prompt groups for package result parameters.

3   In the Select a Shared Group or Prompt dialog box, navigate to the `Products/SAS Intelligence Platform/Samples/` folder. Select the appropriate shared prompt group. The names of these all begin with `Package`. Some of these are SAS server specific (that is, stored process server versus workspace server). So if you chose a stored process server as the SAS server, you should choose a prompt group ending in `(STP Servers)`.

In the Add Prompt dialog box, click **OK** to accept the prompt group displayed text. You have included a reference to the shared prompt group in your stored process. You are not currently allowed to edit the shared prompts.

**4** The prompts must be unshared to make them editable. To unshare the prompts, select the prompt group (for example, `Package – File System with Archive Name`), and click **Unshare**.
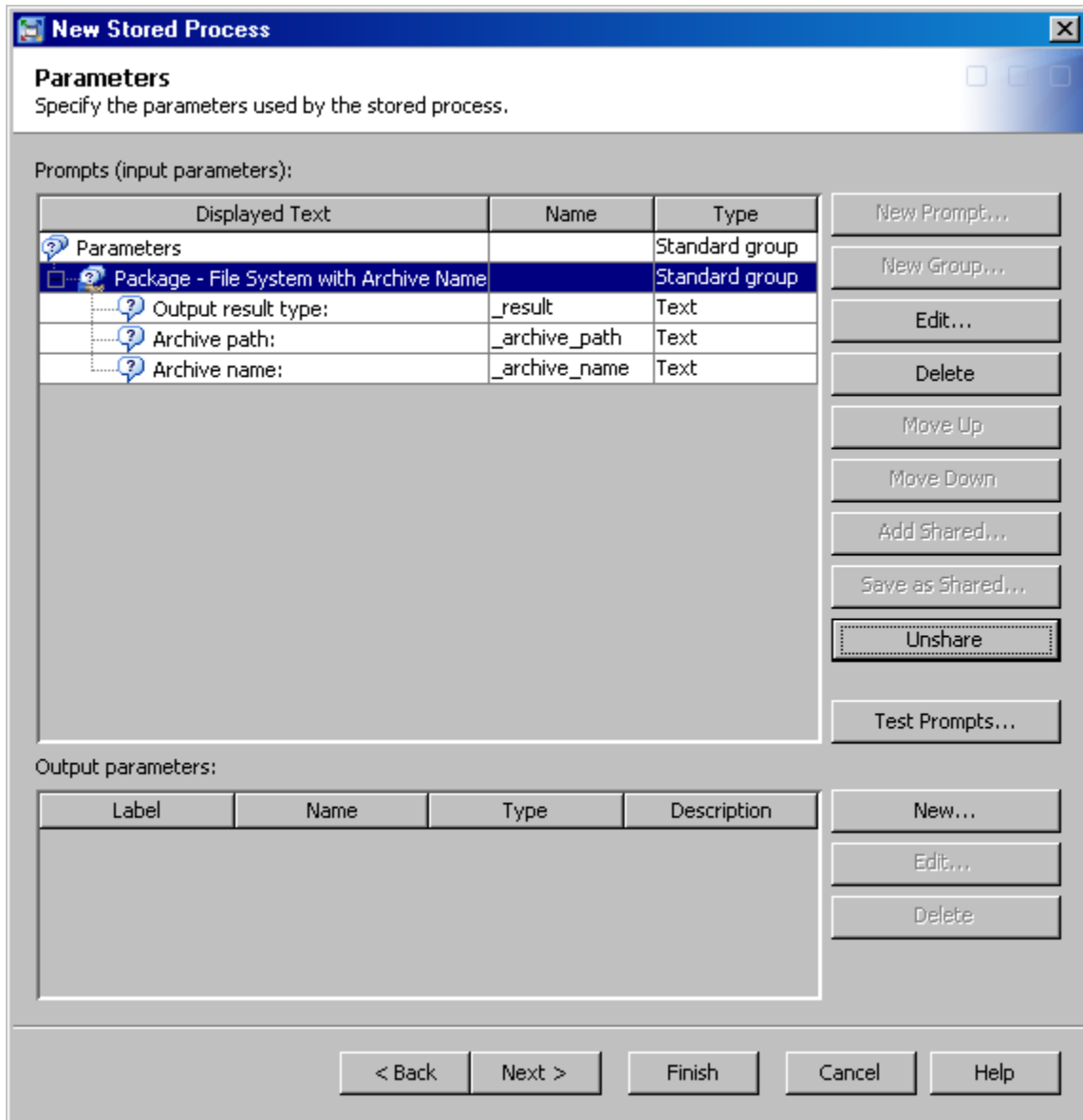


Click **Yes** to continue. This operation gives you your own copy of the data from the shared prompt group, so you can modify it.

**5** If you have not already done so, expand the prompt group to display its members. Double-click the first prompt to open it.

In this example, for the _RESULT prompt, the comment in the **Description** field tells you `Do not modify`. Close the Edit Prompt dialog box without modifying the prompt.

6  Open the remaining prompts in the group. In this example, _ARCHIVE_PATH is next.
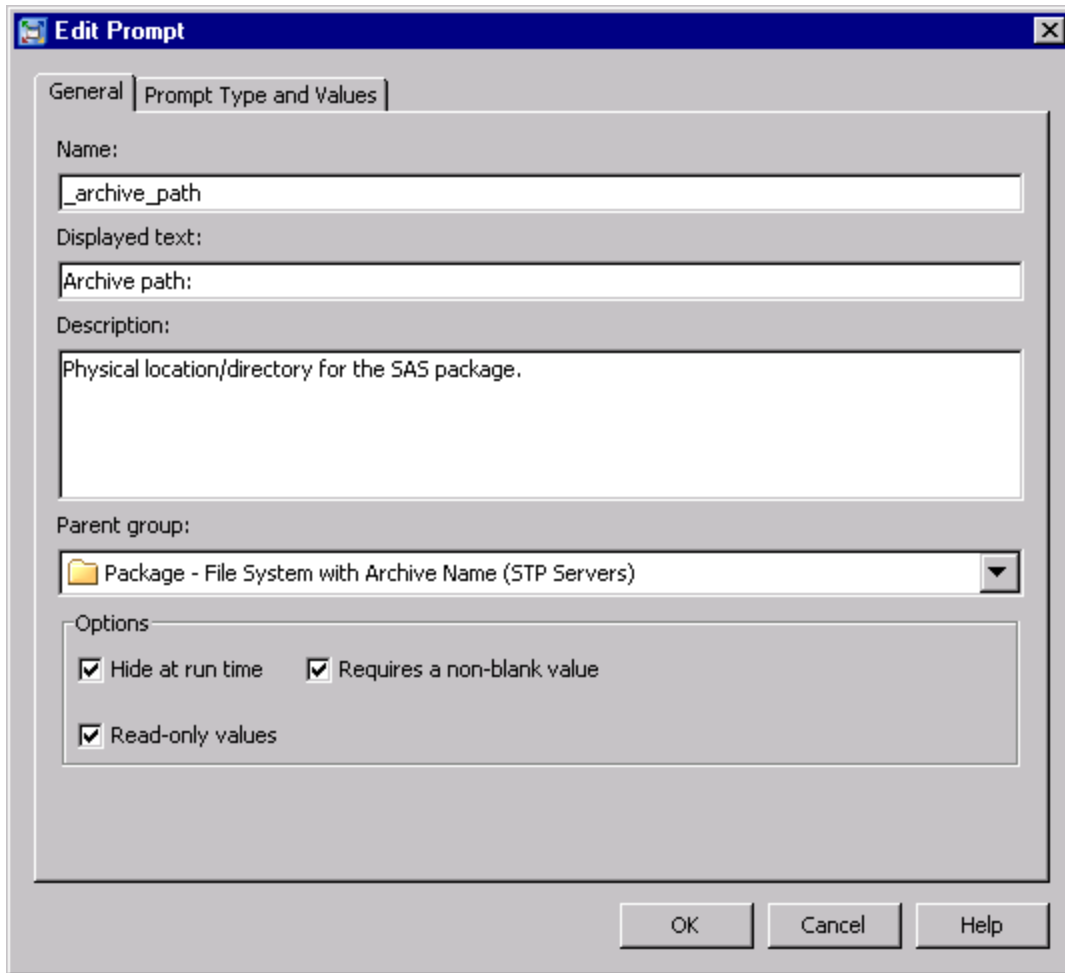
No changes are needed for the **General** tab. The description text tells you what type of value needs to be supplied for this prompt.

Click the **Prompt Type and Values** tab.

Notice that the **Default value** needs to be supplied (as indicated by the text `Supply_Valid_Value`). Set it to a valid value (for example, a physical path on the file system such as `C:\temp`). Click **OK** to accept changes and close the prompt.

7 Repeat this process for the remaining prompts in the group, examining prompt properties, and making necessary changes.

8 Click **Next** in the New Stored Process wizard if you have data sources or data targets to define. Otherwise, click **Finish**.

9 Create the SAS program and save it. Use the source filename and source code repository path that you specified in the New Stored Process wizard.

10 When the stored process is executed, the prompts in the prompt group remain hidden, and the user does not see them. The default values that you specified when you edited them are supplied to the server. If you want to show the prompts at run time, then you need to make the prompt group visible. Make any or all of the prompts visible by deselecting the **Hide from user** check box for each prompt and prompt group.

The SAS code in the previous step writes a SAS package file under the `C:\temp` folder. This package can be opened with WinZip. This content is also returned back to the calling client as HTML.

# Creating Transient Package Results

If transient package results are desired, select `Package` as the result capability in the New Stored Process wizard. None of the special shared prompt groups needs to be added to the stored process. The SAS code returns the package content to the caller when the stored process is executed, but it is not to be written or published to any of the permanent destinations.

# Using Prompts

Input parameters are defined in SAS Management Console as prompts. You can add prompts or prompt groups when you are using the New Stored Process wizard to register a new stored process or when you are viewing properties for a currently registered stored process. The following features are available with prompts:

dynamic prompts
> Dynamic prompts allow the lookup of possible prompt values from a data source such as a SAS data set or information map.

dependencies between prompts
> When you create a set of prompts, you sometimes want the prompts to be interrelated. You might want the values of one prompt to depend on the value that is selected for another prompt. In that case, you would want to set up dependencies between the prompts.
>
> For example, you have a prompt whose values are the names of the divisions in your organization. You also have a prompt whose values are the names of the departments in those divisions. If you want the end user to see only the departments for the selected division, then you set the department prompt to be dependent on the division prompt. After you select a value for the division prompt, the department prompt is then populated with only the names of the departments from that division.

shared prompts and prompt groups
> A shared prompt is a prompt that is stored in a shared location and that can be accessed by multiple users, applications, and software features. Prompt groups can also be shared. Sharing prompts is helpful when that prompt is complex or when you might need to reuse that prompt (perhaps in other applications or contexts). The following examples are good candidates for sharing:
>
> - dynamic prompts with complex configurations
> - sets of cascaded prompts
> - groups of prompts that are often reused (like chart options)

selection groups
> Use a selection group when you want the user to choose from several prompt groups. Selection groups contain selection-dependent groups. Each selection-dependent group is displayed as one of the selections for its parent selection group. The contents (subgroups and prompts) of a selection-dependent group

are displayed to the end user only after the user selects that group for the parent selection group. For example:

- A user is given a choice of `Laptop` or `Desktop` for a computer type prompt.

- If the user selects `Laptop` as the value of the computer type prompt, then the user receives prompts for `Battery Type`, `Hard Drive Size`, and `Processor Speed`.

- If the user selects `Desktop` as the value of the computer type prompt, then the user receives prompts for `Hard Drive Size`, `Processor Speed`, and `Type of Keyboard`.

When you run a stored process that contains prompts, one or more macro variables is generated for each prompt. The values that are specified for the prompts at run time are assigned to the generated macro variables. When a prompt generates more than one macro variable, suffixes such as _REL, _MIN, and _MAX are appended to the prompt name to create unique names for these macro variables. Because macro variables are limited to 32 characters in length, you must use caution when specifying a prompt name. If a suffix of _REL (4 characters long) is added to the prompt name to generate a macro variable, then you should not specify a prompt name that is more than 28 characters long. For more information about how macro variables are generated and the suffixes that are used, see the prompt Help in SAS Management Console.

For more information about input parameters in stored processes, see "Using Input Parameters" on page 8. For more information about how to specify values for prompt, and macro variables that are generated by prompts, see Appendix 3, "Formatting Prompt Values and Generating Macro Variables from Prompts," on page 209. For more information about prompt types and defining prompts, see the prompt Help in SAS Management Console.

# Making Stored Processes Compatible with 9.2 and Upgrading Stored Processes

Starting with 9.3, you can use new stored process features, or you can choose to make your stored processes compatible with 9.2. If your client does not support new stored process features, then you might need to make your stored processes compatible with 9.2. Stored processes that are compatible with 9.2 can also be upgraded to use the new stored process features. The following table shows the differences between stored processes that are compatible with 9.2 and stored processes that use newer features:

| Stored Process That Are Compatible with 9.2 | Stored Processes That Use Newer Features |
| --- | --- |
| Are associated with a specific logical server, which can be a | Are associated with an application server context, and can be run by either a stored process server or a workspace server. You can |

| Stored Process That Are Compatible with 9.2 | Stored Processes That Use Newer Features |
| --- | --- |
| stored process server or a workspace server. | choose whether to restrict the server type or let the client application make the server selection. |
| Store source code on the application server. | Can store source code either on the application server, or in metadata. |
| Allow execution on the specified application server only. | Can allow execution on other application servers, or on the specified application server only. |
| Require the *ProcessBody; comment if they are running on a workspace server. | Do not require the *ProcessBody; comment, regardless of which server is used. |
| Must use the stored process server to produce streaming output. | Can use either the stored process server or the workspace server to produce streaming output. |
| Data sources and targets can be generic streams or XML streams. | Data sources and targets can be generic streams, XML streams, or data tables. |

To make stored processes compatible with 9.2, select one or more stored processes in SAS Management Console. It is helpful to open the Stored Process Properties dialog box for each stored process to make sure that none of the newer features are being used. (See the product Help.) Right-click and select **Make Compatible**. If the stored process runs on a workspace server, make sure that the *ProcessBody; comment is included in the source code.

To upgrade stored processes to use newer features, select one or more stored processes that are compatible with 9.2. Right-click and select **Upgrade**. Open the Stored Process Properties dialog box for each stored process to use the newer features.

# 5

# Debugging Stored Processes

# Examining the SAS Log

The client interfaces that are provided to stored processes usually include a mechanism for retrieving the SAS log from a stored process. For example, passing the input parameter _DEBUG=LOG to the SAS Stored Process Web Application causes the SAS log to be returned with the stored process output. The SAS log is directly accessible from the Stored Process Java API. Assuming that your installation is configured correctly, most run-time stored process errors appear in the SAS log.

If you are unable to access the SAS log from the client interface, you might be able to access the SAS log from the server log files. The server administrator controls the level of server logging that is used and the location of the server log files. Server log options vary depending on the server type.

Stored process and workspace servers enable you to capture the SAS log for each stored process in the server log. To enable logging for the server, perform the following steps:

1  In the `.../Lev1/SASApp/StoredProcessServer/` directory, rename the `logconfig.xml` file as `logconfig_orig.xml`.

     . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
     **Note:**  For a workspace server, this file is located in the `.../Lev1/SASApp/WorkspaceServer/` directory.
     . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2  Make a copy of the `logconfig.trace.xml` file (which is located in the same directory), and name the copy `logconfig.xml`.

3  Restart the Object Spawner.

> **Note:** If you enable logging for the workspace server, then all users who run workspace server requests must have Write access to the location where the log files are written (because the workspace server runs under the client user's account).

For more information about SAS logging, see *SAS Logging: Configuration and Programming Reference*.

# Using SAS Options

Several SAS options can help you debug problems in your stored processes. If you can return the SAS log to your web browser, then activating some of these options can make that log more useful. If you are debugging a stored process that contains macro code, you should supply one or more of these options at the beginning of your stored process: MPRINT, SYMBOLGEN, MLOGIC, or MERROR.

If, for security reasons, you have disabled the display of submitted source code in your stored process by using the NOSOURCE option when you are debugging, you should enable this feature by supplying the SOURCE option. You can then see your submitted SAS code in the log that is returned to your web browser. After you are finished debugging, you can revert to using NOSOURCE if your security model requires it.

# 6

# Composing Stored Process Reports

## Overview of Stored Process Reports

A stored process report consists of stored process output that is cached. The output can be viewed without re-executing the stored process. A stored process report is a view of a stored process. Stored process reports are targeted at stored processes that might involve substantial processing, but do not require real-time updates. A stored process report generated by a stored process can change on a daily, weekly, monthly, or quarterly basis and these stored process reports can now be easily generated without needless repetition of the stored process execution.

SAS Management Console is the design-time environment for stored process reports. Stored process report definitions can be created and modified in the SAS Management Console Folder view. When you create a stored process report, you must specify a stored process and the prompt values that are used to generate the stored process report. All prompt values must be fixed when the stored process report is created. These values are used each time new stored process report output needs to be generated. You can export, import, copy, and paste the stored process report definition, but not the associated stored process report packages.

The output of a stored process report is a result package. Result packages are saved in the SAS Content Server and managed by the stored process report. All cached result packages are deleted when a stored process report is deleted. A stored process report client accesses the stored process report by retrieving the result package through the object API. If a package exists and has not expired, the package is returned to the caller with no additional processing. If no package exists or the package has expired, the stored process is executed to create an up-to-date result package (if the user has permission to execute the stored process and to save data on the SAS Content Server). After the stored process has been executed,

the new result package is returned to the client and any expired packages are deleted.

The SAS Stored Process Web Application provides a run-time or end-user environment for stored process reports. Users can navigate to a stored process report, specify it by path, or find it by searching and then display the contents of the stored process report. When a request is made for a stored process report, the latest output is returned or new stored process report output is generated. The stored process report is displayed by the SAS Package Viewer. Controls are provided to force a stored process report refresh (if this is allowed by the permission settings) or to view prior generations of the stored process report (if there are any available). The refresh method forces the creation of a new package regardless of whether the existing package has expired. You can also purge all cached output, or purge a specified package generation.

# Creating and Managing Stored Process Reports

The New Stored Process Report wizard in SAS Management Console can be used to create new stored process reports, or you can use the Stored Process Report Properties dialog box in SAS Management Console to modify existing stored process reports. You can specify and manage the following information for stored process reports:

**Folder**
specifies the metadata location of the stored process report. The folders are defined in metadata and do not correspond to any physical location. The folder hierarchies that are used for stored process reports can also hold other objects such as SAS reports, information maps, and administrative metadata. You can create and modify folders using SAS Management Console.

**Name**
specifies the stored process report name, which acts as both a display label and as part of the URI for the stored process report.

**Description**
specifies an optional text description of the stored process report.

**Keywords**
specifies an optional list of keywords to associate with the stored process report. Keywords are arbitrary text strings that are typically used for searching or to indicate specific capabilities.

**Responsibilities**
specifies one or more users who are responsible for the stored process report. This information is optional.

**Stored process**
specifies the stored process that is to be used to run the stored process report.

**Prompt values**
enables you to provide values for the prompts for the selected stored process.

**Maximum retained generations**

specifies the number of generations of output to save for the stored process report. By default, stored process reports keep only a single, current result package. Multiple generations can be enabled, which allows old result packages to be kept subject to the multiple generation policy. Multiple generation policy is set by a generation count limit and a generation age limit. For example, a stored process report can be set to keep packages that are no more than one month old, but with a limit of no more than six packages. Multiple generation support is optional for stored process report clients. A client might choose to show only the most recent generation or might allow the user to choose previous generations of the result package. Each package has the following attributes available to the client:

- Generation (an integer from 1 to *n*, incremented for each new package)

- Timestamp (creation time)

- Package URL User (user that generated the package)

- Execution time for the stored process

An API method is provided to delete specific packages.

**Expiration policy**

specifies how often and when output generations expire for the stored process report. New output is not automatically generated. This setting determines when a result package expires. If a client accesses a stored process report before it expires, then the cached stored process results are displayed in the client. If a client accesses a stored process report after it has expired, then the stored process is re-executed and a new output generation is created for the stored process report. Sample expiration policies might be:

- `Never`

- `Every day at 23:30`

- `Every weekday at 00:00`

- `Every Tuesday and Thursday at 12:00`

- `Last day of every month at 00:00`

- `Every April 15th at 23:59`

**Authorization**

specifies access controls for the stored process report.

You cannot specify authorization information from the New Stored Process Report wizard. To specify authorization information, you must open the Stored Process Report Properties dialog box for an existing stored process report.

# 7

# Building a Web Application with SAS Stored Processes

# Overview

## Overview of Stored Process Web Applications

Stored processes are frequently used in web-based applications. While almost any stored process can be executed through a web interface, the typical web application design might require special techniques. This chapter documents special issues that you might encounter when building a web application.

Web applications are typically implemented by streaming output stored processes. Streaming output stored processes deliver their output through the _WEBOUT fileref. You can write directly to the _WEBOUT fileref by using PUT statements, or you can use the Output Delivery System (ODS) to generate output. The example code throughout this chapter demonstrates both approaches. The workspace server is not an appropriate host for many web applications.

Web applications can be implemented using the SAS Stored Process Web Application, the Stored Process Service application programming interface (API), or a combination of both. The SAS Stored Process Web Application is a Java middle-tier application that executes stored processes on behalf of a web client. Only SAS and HTML programming skills are required; no Java programming is required. Most of the examples in the remainder of this chapter assume the use of the SAS Stored Process Web Application. The Stored Process Service API enables the Java developer to embed stored processes within a Java web application.

# How the SAS Stored Process Web Application Works

The SAS Stored Process Web Application is a Java web application that can execute stored processes and return results to a web browser. The SAS Stored Process Web Application is similar to the SAS/IntrNet Application Broker, and it has the same general syntax and debugging options. The SAS Stored Process Web Application is included with the SAS Web Infrastructure Platform, which is a component of SAS Integration Technologies.

Request processing for the SAS Stored Process Web Application is similar to SAS/IntrNet Application Dispatcher request processing. The following steps describe how the SAS Stored Process Web Application processes a request:

1   Users enter information in an HTML form by using their web browser and then submitting it. The information is passed to the web server, which invokes the first component, the SAS Stored Process Web Application.

2   The SAS Stored Process Web Application accepts data from the web server and contacts the SAS Metadata Server for user authentication and retrieval of stored process information.

3   The stored process data is then sent by the SAS Stored Process Web Application to a stored process server through the object spawner.

4   The stored process server invokes a SAS program that processes the information.

5   The results of the SAS program are sent back through the web application and web server to the web browser of the user.

The following diagram illustrates this process:
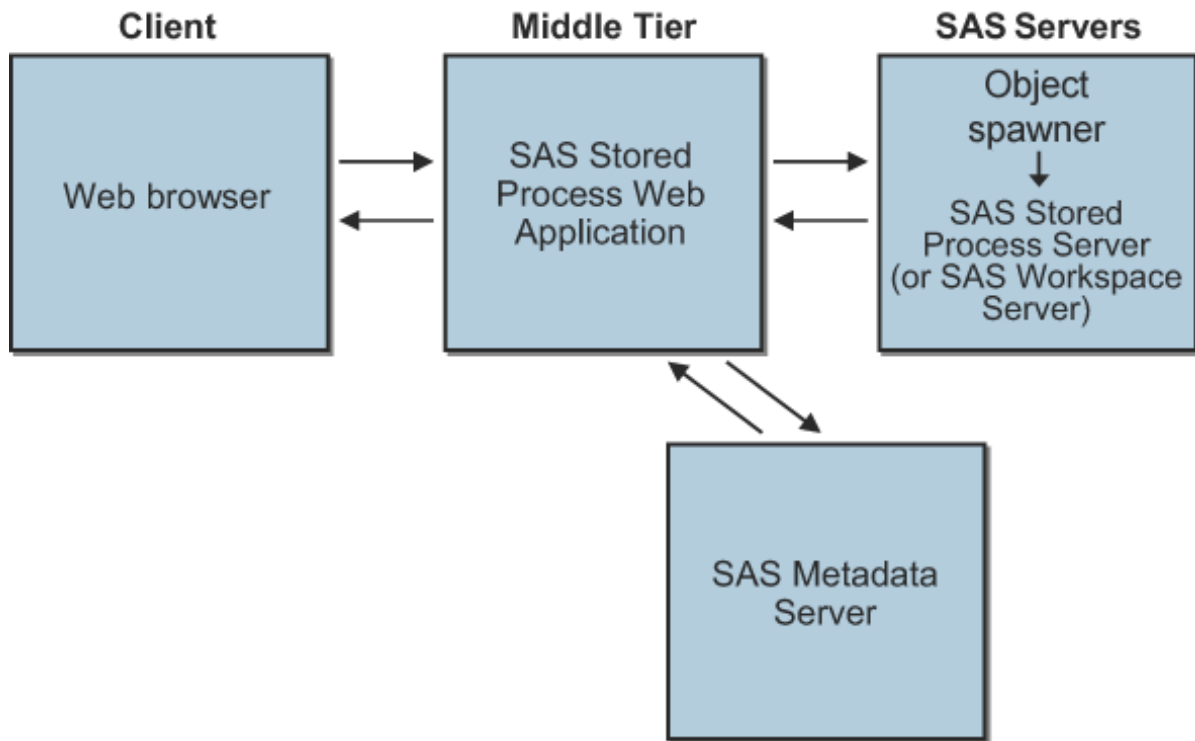
## SAS Stored Process Web Application Samples

The SAS Stored Process Web Application comes installed with a sample Welcome page and a set of sample stored processes. These samples illustrate various features that are available with stored processes. The Welcome page searches for the stored processes and displays them in a table. The Welcome page and sample custom input forms have been localized.

You can find the SAS source for the samples is in the SAS installation directory, which might look like the following path: `/SAS/SASFoundation/9.4/inttech/sample`

The sample custom input forms are located in the SAS Stored Process Web Application installation directory, which might look like the following path: `<SASHOME>\Config\Lev1\Web\WebAppServer\SASServer1_1\sas_webapps\sas.storedprocess.war\input\Samples`

The samples are registered in metadata at the following path: `/Products/SAS Intelligence Platform/Samples/`

# Configuring the SAS Stored Process Web Application

## Configuration Files

The SAS Stored Process Web Application can be customized for your site through various configuration files and servlet initialization parameters. The following table describes the external files that are read by the SAS Stored Process Web Application.

*Table 7.1*  *Configuration Files*

| File | Description |
| --- | --- |
| application_config.xml | Contains user information for the SAS Metadata Repository and is delivered in the sas.storedprocess.war file. |
| banner.jsp | Is used to generate the top banner in the SAS Stored Process Web Application pages. This file is located in the /`SASStoredProcess/jsp` directory and can be altered or replaced if you want to customize the banner. |
| Params.config | Contains stored process input parameters that are set before any client parameters are processed. The parameters are defined in the form `name=value` on a separate line with a '#' character in column one to indicate a comment. Continuation lines can be specified with a '\' character at the end of a line. For more information about properties that can be substituted into input parameters in the Params.config file, see "Web Application Properties" on page 88. Parameters defined in the Params.config file cannot be overridden. |
| Resources.properties | Contains name/value pairs for locale-defined output strings. This file is delivered in the |

| File | Description |
|---|---|
| | sas.storedprocess.webapp.jar file and is usually not altered. |
| search.jsp | Is used to search the SAS Stored Process Web Application for stored processes and stored process reports. |
| web.xml | Contains servlet mappings and initialization parameters. This file is the web application configuration file and is delivered in the sas.storedprocess.war file. |
| Welcome.jsp | Specifies an optional page that is displayed when the SAS Stored Process Web Application is invoked with no parameters. |

The SAS Stored Process Web Application session contains values that might be useful to a JSP (including custom input forms) that is installed in the SAS Stored Process Web Application directory tree. These values are obtained using the following method:

```
session.getAttribute("parameter_name")
```

For *parameter-name*, specify one of the following parameters:

| | |
|---|---|
| Banner_Locale | returns the current user locale. |
| Banner_LogoffURL | returns current URL that is used for logging out. |
| Banner_Theme | returns the current theme. |
| Banner_TimeoutURL | returns the current URL that is used for time-outs. |
| Banner_Title | returns the current banner title. |
| sas.framework.user.name | returns the login user name (for example, `SAS Demo User`). |
| sas.framework.userid | returns the login user ID (for example, `sasdemo`). |
| saspfs_sessionid | returns the security session ID. |
| SASStoredProcessURI | returns the servlet URI (for example, `/SASStoredProcess/do`). |
| sp_counter | returns the number of times the HTTP session has been accessed. |
| sp_sessionContext | returns the local SessionContextInterface. Use sp_sessionContext.getUserContext() to get the current UserContextInterface. |

# Custom Responses

You can also customize responses for the SAS Stored Process Web Application by creating JSP files that are described in the following table. You must store the JSP files in the `/SASStoredProcess/jsp/response/` directory so that the web application can forward to the corresponding file.

*Table 7.2   Custom Responses*

| File | Description |
| --- | --- |
| Background.jsp | Specifies a page that is displayed when a stored process has been submitted for background processing. |
| InvalidSession.jsp | Specifies a page that is displayed when an invalid or expired session ID is sent. This file is used only with _REPLAY sessions. |
| NotFound.jsp | Specifies a page that is displayed when the stored process cannot be located. |

# Initialization Parameters

The following table describes the initialization parameters that are available to the SAS Stored Process Web Application. Initialization parameters are values that are set when the SAS Stored Process Web Application is started. These parameters control various web application processing options. Initialization parameters are defined in the SAS Stored Process Web Application configuration metadata. Values can be added or changed using the Configuration Manager in SAS Management Console. Expand the **Configuration Manager** group on the Plug-ins tab in SAS Management Console, and then expand the **SAS Application Infrastructure** node. Right-click the **Stored Process Web App 9.4** node and select **Properties**. In the Properties dialog box, click the Advanced tab and then add or edit the initialization parameters.

**Note:** The web application server must be restarted for parameter changes to take effect.

*Table 7.3* *Initialization Parameters*

| Initialization Parameter | Description |
| --- | --- |
| ActionMask | Specifies the _ACTION values that users can set. The default is to allow all keywords. Valid names can be specified as a comma-separated list. |
| AllowBasicAuthentication | Enables the SAS Stored Process Web Application to bypass the Logon Manager and to handle user verification through the<br><br>`http://yourserver.com:8080/SASStoredProcess/do1`<br><br>URL if the AllowBasicAuthentication parameter is set to true. |
| AllowEncodedPassword | Allows encoded passwords to be passed in via the _password parameter if the AllowEncodedPassword parameter is set to `true`. |
| AllowGuest | Enables a guest user to run stored processes without logging in if this parameter is set to `true`.<br><br>**Note:** App.AllowGuest must also be set to `true`, or GuestUsername and GuestPassword must be set. |
| App.AllowGuest | Enables an anonymous web user to run stored processes without logging in if this parameter is set to `true`. This parameter also causes an additional **Guest** option to be added to the sign-in page.<br><br>**Note:** AllowGuest must also be set to `true`. |
| App.PublicIdAllowed | Allows public IDs in the SAS Stored Process Web Application if this parameter is set to `true`. |
| BannerRows | Specifies the number of rows sent in the tag for the banner frame. (For _ACTION=INDEX, three HTML frames are created with the top frame being the banner frame.) By default, the size of the banner frame is dynamically adjusted to the size of the displayed banner. |
| Debug | Specifies default _DEBUG values. |
| DebugMask | Specifies the _DEBUG values that users can set. The default is to allow all |

| Initialization Parameter | Description |
| --- | --- |
| | keywords. Valid names can be specified as a comma-separated list. |
| GuestUsername | Specifies the user name to use when accessing the SAS Stored Process Web Application as /guest. |
| GuestPassword | Specifies the password to use when accessing the SAS Stored Process Web Application as /guest. |
| InternalURL | Specifies a URL for the SAS Stored Process Web Application to use to retrieve resource files. This is useful when the external host or port has been denied access to the SAS Stored Process Web Application. The value should be in the form `http://xxx.yyy.com:8080`. Starting with SAS 9.4M2, this initialization parameter is no longer needed because the SAS Stored Process Web Application accesses resource files directly rather than through a URL. |
| ParamsFile | Specifies the file that contains the preset input parameters. This value is a fully expanded file specification. The default preset filename is Params.config in the SAS Stored Process Web Application root context directory. |
| RTL.SupportedLocales | Specifies a comma-delimited list of locales that should be displayed right to left. Use this initialization parameter to enable the SAS Stored Process Web Application to display languages from right to left. For SAS 9.4, this is supported only by the SAS Stored Process Web Application prompt page. |
| SessionTimeout | Specifies the number of minutes that elapse before a servlet session expires. The default session time-out varies by application server (typically 30-60 minutes). After the session expires, the user is required to log on again. Any data that was entered on the prompt page needs to be reentered. |
| ShowLogButton | Disables the **Show SAS log** button from being displayed on program errors (if this parameter is set to `false`). |

| Initialization Parameter | Description |
| --- | --- |
| UploadDirectory | Specifies a temporary directory for caching files when the file size exceeds 32768 bytes. The default directory is `java.io.tmpdir`. |
| UploadMaxSize | Specifies the maximum file size in bytes that can be uploaded. |
| ValidatePromptValues | Forces constraint checking, static list validation, or dynamic list validation on a stored process, if the ValidatePromptValues parameter is set to `true`. By default, this parameter is set to `false`. |
| WelcomePage | Specifies a page to display if no parameters are entered in the URL. If the value that you specify starts with a slash (/), then the Welcome page is relative to the web application root context (for example, `/jsp/Welcome.jsp`) and the web browser is forwarded to that page. Otherwise, a redirect command is sent to the web browser for the specified page. |
| | The SAS Stored Process Web Application uses the following sequence to determine what is displayed when the user logs in: |
| | 1 Use the value for _WELCOME, if this value has been set. |
| | 2 Use the value of the WelcomePage initialization parameter, if this value has been set. |
| | 3 Check for a Welcome.jsp file in the /jsp directory. |
| | 4 Display the SAS Stored Process Web Application version and build number. |

## Web Application Properties

Various reserved values, or properties, are available to be passed as input parameters to stored processes that are executed by the SAS Stored Process Web Application. To pass a property to every stored process that is executed by the SAS Stored Process Web Application, add a line of the form `name=$reserved_name` to the Params.config file. For example, to add request cookie information as an input parameter, add the following line to Params.config:

```
_HTCOOK=$servlet.cookies
```

The input parameter _HTCOOK is created, and it contains the HTTP header cookie data. The _HTCOOK parameter is added to the input parameters for the stored process.

Fixed data values can also be passed by using the form *name=string*. For example, the following line sets the parameter MYPARM to the fixed string **Hello**:

```
MYPARM=Hello
```

**Note:** Any unresolved values can result in the corresponding parameter being set to a zero-length string.

*Table 7.4*  *Properties for Web Applications*

| Reserved Name | Recommended SAS Variable Name | Description |
| --- | --- | --- |
| servlet.auth.type | _AUTHTYP | Specifies the name of the authentication scheme that is used to protect the SAS Stored Process Web Application (for example, BASIC or SSL, or null if the SAS Stored Process Web Application was not protected). |
| servlet.character.encoding | | Specifies the name of the character encoding that is used in the body of the request. |
| servlet.content.length | | Specifies the length, in bytes, of the request body and is made available by the data source. If the length is not known, the value is –1. |
| servlet.content.type | | Specifies the MIME type of the body of the request. If the type is not known, the value is null. |
| servlet.context.path | | Specifies the portion of the request URL that indicates the context of the request. |
| servlet.cookies | _HTCOOK | Specifies all of the cookie strings that the client sent with this request. |

| Reserved Name | Recommended SAS Variable Name | Description |
| --- | --- | --- |
| servlet.header | | Specifies the HTTP request header as it was received by the SAS Stored Process Web Application. |
| servlet.header.accept | _HTACPT | Specifies the MIME types that are accepted by the stored process client. |
| servlet.header.referer | _HTREFER | Specifies the address of the referring page. |
| servlet.header.user-agent | _HTUA | Specifies the name of the user agent. |
| servlet.header.*<name>* | | Specifies a particular HTTP request header line as it was received by the SAS Stored Process Web Application, where *<name>* is the header keyword name. |
| servlet.info | | Specifies any information about the SAS Stored Process Web Application, such as author, version, and copyright. |
| servlet.jsessionid | | Specifies the Java servlet session ID. |
| servlet.locale | | Specifies the preferred locale in which the client accepts content, based on the Accept-Language header. |
| servlet.method | _REQMETH | Specifies the name of the HTTP method with which this request was made (for example, GET, POST, or PUT). |
| servlet.name | | Specifies the name of this SAS Stored Process Web Application instance. |
| servlet.path | | Specifies the part of the request URL that calls the |

| Reserved Name | Recommended SAS Variable Name | Description |
|---|---|---|
| | | SAS Stored Process Web Application. |
| servlet.path.info | | Specifies any extra path information that is associated with the URL that the client sent when it made this request. |
| servlet.path.translated | | Specifies any extra path information after the SAS Stored Process Web Application name but before the query string, and translates this information to a real path. |
| servlet.protocol | _SRVPROT | Specifies the name and version of the protocol that the request uses in the form protocol/ majorVersion.minorVersion (for example, HTTP/1.1). |
| servlet.query.string | _QRYSTR | Specifies the query string that is contained in the request URL after the path. |
| servlet.remote.addr | _RMTADDR | Specifies the Internet Protocol (IP) address of the client that sent the request. |
| servlet.remote.host | _RMTHOST | Specifies the fully qualified name of the client that sent the request, or specifies the IP address of the client if the name cannot be determined. |
| servlet.remote.user | _RMTUSER | Specifies the login ID of the user that is making this request if the user has been authenticated. If the user has not been authenticated, the value is null. |

| Reserved Name | Recommended SAS Variable Name | Description |
|---|---|---|
| servlet.request.inputencoding | _REQENCODING | Specifies the servlet input encoding. The default encoding is UTF-8. |
| servlet.request.uri | _URL | Specifies the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request. |
| servlet.root | | Specifies the SAS Stored Process Web Application root context directory. |
| servlet.scheme | | Specifies the name of the scheme that was used to make this request (for example, HTTP, HTTPS, or FTP). |
| servlet.secure | | Returns true or false indicating whether this request was made using a secure channel, such as HTTPS. |
| servlet.server.name | _SRVNAME | Specifies the host name of the server that received the request. |
| servlet.server.port | _SRVPORT | Specifies the port number on which this request was received. |
| servlet.server.software | _SRVSOFT | Specifies the web server software. |
| servlet.user.name | _username | Specifies the value for the user name that was obtained from the web browser authentication. The symbol _username is set automatically by the SAS server. |
| servlet.version | _VERSION | Specifies the SAS Stored Process Web Application version and build number. |

Numerous system properties (for example, user.name) can be obtained. Setting
_DEBUG to ENV shows all the available values.

# Specifying Web Application Input

## Overview of Web Application Input

A web application that uses stored processes must have a way of sending input
parameters to the stored processes. Input parameters are typically generated by an
HTML page and passed through the SAS Stored Process Web Application or a
user-written JSP to the stored process. Input parameters can be specified in the
following:

■ fields in an HTML form. The user provides the required information and submits
the request. The web browser sends data from the form (including both user-
entered data and hidden fields) to the server. HTML forms are generally used
where user input is required to control the execution of the stored process.

■ a hypertext link in an anchor tag. The link URL includes parameter values that
are passed to the server when the user selects the link. Hypertext links are
generally used where the input parameters have fixed values (for example, as
drill-down links in a table or image).

■ an inline image or other embedded link in the HTML page. This case also
includes frames within an HTML frameset. In most cases, the web browser
fetches the embedded object when the user loads the HTML page. Fetching the
embedded object can cause input parameters to be passed to a stored process.

■ URLs or forms that are created and submitted by JavaScript or a similar scripting
technology in the web browser.

The HTML page that uses these techniques can be a static HTML page or a
dynamic page that is generated on demand by another stored process or by a Java
Server Page (JSP). In all cases, the input parameters must follow the naming
conventions and other basic rules that are described in "Using Input Parameters" on
page 8. Reserved parameter names should be used only as recommended. For
more information, see "Using Reserved Macro Variables" on page 26. Reserved
parameter names should be used only as recommended.

The SAS Stored Process Web Application is set up to use the
SanitizingRequestFilter to check for invalid requests. If an invalid input string (for
example, `<script>`) is found in input parameters, then a status code 403 is returned
to the web browser. For more information about this filter, see the *SAS Intelligence
Platform: Web Application Administration Guide*.

All of the previously mentioned techniques for specifying input parameters rely on
URLs or HTML forms. The following sections discuss how parameters are passed in
both cases. These sections assume the use of the SAS Stored Process Web
Application. JSPs generally use similar conventions, but the details are determined
by the author of the JSP.

# Specifying Input Parameters in a URL

You can specify input parameters as a sequence of name/value pairs in a URL by using the query string syntax. For example, the following URL specifies two name/value pairs.

```
http://yourserver/SASStoredProcess/do?
 _program=/WebApps/Sales/Weekly+Report&region=West
```

The URL specifies your server, an absolute path to your SAS Stored Process Web Application, and the query string (following the question mark character). Each name in the query string is separated from the following value by an equal sign (=). Multiple name/value pairs are separated by ampersand characters (&). In this example, `_program=/WebApps/Sales/Weekly+Report` is the reserved input parameter that specifies the stored process that is to be executed. The second name/value pair (`region=West`) is another input parameter to be passed to the stored process.

There are special rules for the formatting of name/value pairs in a URL. Special characters (such as most punctuation characters, including spaces) in a value must be URL-encoded. Spaces can be encoded as a plus sign (+) or %20. Other characters are encoded using the %*nn* convention, where *nn* is the hexadecimal representation of the character in the ASCII character set. In the previous example, the value `/WebApps/Sales/Weekly+Report` actually identifies the stored process named "Weekly Report". The space in the name is encoded as a plus sign (+). If your parameter values contain special characters, then it is important that they are URL-encoded. Use the URLENCODE DATA step function when creating URLs in a stored process.

URLs are typically used in an HTML tag attribute, and this might require extra encoding to be properly interpreted. The ampersand characters that are used in the URL query string can cause the web browser to interpret them as HTML markup. The parameter `&region=West` is interpreted as `&reg;ion=West` in some web browsers. Use HTML encoding to avoid this problem. The following example shows the correct HTML code:

```
 <A HREF="http://yourserver/SASStoredProcess/do?
_program=/WebApps/Sales/Weekly+Report&amp;region=West">
```

The HTMLENCODE DATA step function can be used to encode the URL in a stored process. If we assume that the variable `myurl` contains a URL with various input parameters, then the following code creates an anchor tag in the variable `atag` that is properly encoded:

```
 atag = '<A HREF="' || htmlencode(myurl,
 'lt gt amp quot') || '">';
```

Note that some web browsers and web servers might impose a limit on the total length of a URL. URLs with many parameter values that exceed this limit can be truncated without warning, which results in incomplete or inconsistent input data for your stored process. URL length limits are not well documented and might require experimentation with your particular configuration.

For information about specifying multiple values for an input parameter, see "Input Parameters with Multiple Values" on page 12.

# Specifying Name/Value Pairs in an HTML Form

HTML forms provide the most versatile mechanism for sending input parameters to a stored process. A form definition begins with the <FORM> tag and ends with the </FORM> tag. Between these two tags, other HTML tags define the various components of the form, including labels, input fields, selection lists, push buttons, and more. Here are some issues that are related to stored process input parameters in HTML forms:

- The ACTION attribute of the <FORM> tag generally points to the SAS Stored Process Web Application or a JSP that executes the stored process. The METHOD attribute of the <FORM> tag can be set to GET or POST.

- The GET method causes the web browser to construct a URL from all of the field values in the form. The URL is exactly like the URLs that were discussed in the previous section. The GET method enables the user to bookmark a specific stored process execution, including all input parameters, but the total length of all parameters might be limited. Web servers typically log all requested URLs, and this method causes all input parameters to be included in the web server log, which can be a possible security issue.

- The POST method uses a special post protocol for sending the parameters to the server. The POST method allows an unlimited number of input parameters and usually hides them from the web server log, but this method does not allow the execution to be bookmarked in a web browser.

Hidden fields are name/value pairs in a form that do not appear as buttons, selection lists, or other visible fields on the HTML page. Hidden fields are frequently used to hold fixed input parameters that do not require user input. For example, the following code specifies the stored process to be executed by this form.

```
<INPUT TYPE="hidden"
 NAME="_program" VALUE="/WebApps/Sales/Weekly Report">
```

The space in the stored process name is not encoded as in the previous URL section. Values in hidden fields and other field types should not be URL-encoded, but might still need to be HTML-encoded if they contain HTML syntax characters such as a less than sign (<), a greater than sign (>), an ampersand (&), or quotation marks (").

# Specifying Custom Input Forms

The SAS Stored Process Web Application looks for a custom input form if you add the parameter _ACTION=FORM to the web application URL. You can use the _FORM variable with _ACTION=FORM to specify the location of a custom input form JSP file for a stored process. If the value starts with a slash (/), then the JSP file is assumed to be located relative to the SAS Stored Process Web Application root. Otherwise, it is assumed to be a complete URL and a redirect is performed to that value.

The stored process samples have _FORM defined in the source code. The _FORM parameter is defined in the stored process metadata.

For example, the Shoe Sales by Region sample has _FORM defined as follows:

```
_FORM=/input/samples/stpods1/stpods1.jsp
```

The Shoe Sales by Region sample stored process can be accessed with the following code:

```
http://yourserver/SASStoredProcess/do?
 _program=/Samples/Stored+Processes/
 Sample:+Shoe+Sales+by+Region&_action=form
```

Your web browser is forwarded as shown here:

```
http://yourserver/SASStoredProcess
 /input/samples/stpods1/stpods1.jsp?
 _program=/Samples/Stored+Processes/
 Sample:+Shoe+Sales+by+Region
```

If _FORM is not specified, then custom input forms are stored as JSPs under the `input` folder in the `SASStoredProcess` directory.

......................................................................................................................

**Note:** If a custom input form with zero length is found, then the form is skipped and the stored process executes immediately.

......................................................................................................................

In order to create the input form path and name for the case when _FORM is not specified, all names in the stored process path (both folders and the stored process itself in the _PROGRAM parameter) are converted to an equivalent file system path for a JSP file. The following special characters in a folder or stored process name are converted to underscore characters: ' " ;: * ? < >\ | tabs and blank spaces.

For example:

```
 /Samples/John's Test Area/Test: Hello World (English) V1.0
```

would be converted to:

```
<webapp-home>/input/Samples/John_s_Test_Area/
 Test__Hello_World_(English)_V1.0.jsp
```

For more information about the SAS Stored Process Web Application and custom input forms, including a sample form, see "Custom Input Form" on page 115.

Custom input forms are provided with most of the sample stored processes that are included in the SAS Web Infrastructure Platform. Custom input form JSP files can be deployed from the SAS Stored Process Web Application configuration area. Consider a stored process with the following name and location: **/Reports/East Region/Sales Summary 2005** This stored process has a custom input form with the filename Sales_Summary_2005.jsp. It is maintained and deployed from the following location: **<SASHOME>\Config\Lev1\Web\WebAppServer \SASServer1_1\sas_webapps\sas.storedprocess.war\input\Reports \East_Region**

Custom input forms can be deployed as part of the sas.storedprocess.war file. The sas.storedprocess.war file is built and deployed by the SAS Web Infrastructure Platform redeployment process.

......................................................................................................................

**Note:** The SAS Stored Process Web Application is delivered in an EAR file and can be run directly from the EAR file or from the exploded directory. For more information about how to explode the EAR file, see the *SAS Intelligence Platform: Web Application Administration Guide*.

......................................................................................................................

As an alternative to creating a separate JSP file for your custom input form, you can use HTML following the CARDS4 statement in a DATA step.See "Passing Data through the DATA Step" on page 133 for more information.

## Specifying Prompt Pages

Prompt pages provide a parameter input page for stored processes that do not have custom input forms. The prompt page is accessed by adding the parameter _ACTION=PROPERTIES to the web application URL. Parameters must be defined in the stored process metadata in order for them to be visible in a prompt page. For more information about the SAS Stored Process Web Application and prompt pages, see "Using the SAS Stored Process Web Application Pages" on page 110.

If you are unsure whether a stored process has a custom input form, you can specify _ACTION=FORM,PROPERTIES,EXECUTE on the web application URL. This is the default action for a stored process accessed from the SAS Information Delivery Portal. This action causes the web application to do the following:

- display the custom input form if it exists

- display the prompt page if the input form does not exist and the stored process has prompts defined

- execute the stored process if there is no custom input form and there are no prompts defined

**Note:** Starting with SAS 9.4M8, SAS Information Delivery Portal and SAS BI Portlets are not available from SAS. For more information, see Unconfiguring and Uninstalling Retired Products in *SAS Guide to Software Updates and Product Changes*.

## Uploading Files

## Overview of Uploading Files

You can use the SAS Stored Process Web Application to upload one or more files to your SAS Stored Process Server or SAS Workspace Server. The upload process is initiated by a custom input form that contains an INPUT tag with the attribute TYPE set to `file`:

```
<input type="file"
name="myfile">
```

This tag enables you to specify the file that you want to upload. For more information, see "Specifying Custom Input Forms" on page 95. After the form data is submitted, the file that you chose and any other name/value pairs that are contained

in the custom input form are sent to the server. Your stored process can then use both the name/value pairs and the file that was uploaded.

# Reserved Macro Variables

The reserved SAS macro variables that are associated with uploading files all start with `_WEBIN_`.

_WEBIN_CONTENT_LENGTH
  specifies the length, in bytes, of the file that was uploaded.

_WEBIN_CONTENT_TYPE
  specifies the content type that is associated with the file.

_WEBIN_FILE_COUNT
  specifies the number of files that were uploaded. If no files were uploaded, then the value of this variable is set to zero.

_WEBIN_FILEEXT
  specifies the extension of the file that was uploaded.

_WEBIN_FILENAME
  specifies the original location of the file.

_WEBIN_FILEREF
  specifies the SAS fileref that is automatically assigned to the uploaded file. You can use this fileref to access the file. The uploaded file is stored in a temporary location on the stored process server or workspace server, and is deleted when the request is completed. Be sure to copy the file to a permanent location if you need to access it at a later date.

_WEBIN_NAME
  specifies the value that is specified in the NAME attribute of the INPUT tag.

_WEBIN_SASNAME
  specifies a unique name for the SAS table, view, or catalog that was uploaded. A value is set for this macro variable only if a SAS table, view, or catalog was uploaded. All SAS data types are stored in the Work library. The type of SAS file that was uploaded is stored in the _WEBIN_SASTYPE macro variable. See also _WEBIN_SASNAME_ORI.

_WEBIN_SASNAME_ORI
  specifies the original name of the SAS table, view, or catalog that was uploaded. If a SAS table named `mydata.sas7bdat` was uploaded, then _WEBIN_SASNAME_ORI contains the value `mydata`. A value is set for this macro variable only if a SAS table, view, or catalog that was uploaded. All SAS data types are stored in the Work library. The type of SAS file that was uploaded is stored in the _WEBIN_SASTYPE macro variable. See also _WEBIN_SASNAME.

_WEBIN_SASTYPE
  specifies the type of SAS file that was uploaded: DATA for SAS tables, VIEW for SAS views, and CATALOG for SAS catalogs. A value is set for this macro variable only if a SAS table, view, or catalog was uploaded. The name of the uploaded file is stored in the _WEBIN_SASNAME macro variable.

_WEBIN_STREAM
  specifies the name of the data source that was used to upload the file.

_WEBIN_STREAM_COUNT
> specifies the number of files that were uploaded. If no files were uploaded, then the value of this variable is set to zero.

If you are uploading more than one file, then unique macro variables are created for each file. This applies to all of the previous reserved macro variables except _WEBIN_FILE_COUNT and _WEBIN_STREAM_COUNT.

**Note:** For z/OS, the SAS server must be invoked with the FILESYSTEM=HFS option in order to be able to upload SAS file types.

# Examples of How to Upload Files

## Example 1: Uploading a Single File

The following figure shows a custom input form that can be used to upload a single file to the stored process server:
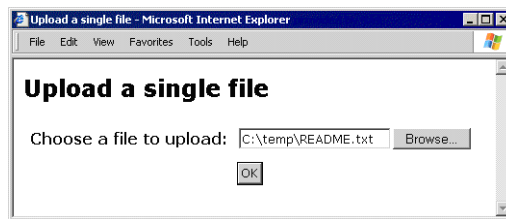


Here is an HTML example for uploading a single file:

```
<form action="StoredProcessWebApplicationURL" method="post"
   enctype="multipart/form-data">
<input type="hidden" name="_program" value="/Path/StoredProcessName">
<table border="0" cellpadding="5">
   <tr>
      <th>Choose a file to upload:</th>
      <td><input type="file" name="myfile"></td>
   </tr>
   <tr>
      <td colspan="2" align="center"><input type="submit" value="OK"></td>
   </tr>
</table>
</form>
```

In the preceding HTML example, you must replace "*StoredProcessWebApplicationURL*" with the path to the SAS Stored Process Web Application. This path is usually `http://YourServer:8080/SASStoredProcess/do`, where *YourServer* corresponds to the domain name of your server. Similarly, you need to specify the path and name of the stored process that you want to execute after the file has been uploaded. You should specify the exact values that are shown for the METHOD and ENCTYPE attributes of the FORM tag.

The INPUT tag in the preceding HTML example is used to create the **Browse** button and text entry field in the preceding figure. The appearance of this control might be different depending on which web browser you use, but the functionality should be the same. Clicking the **Browse** button enables you to navigate to the file that you want to upload. You can choose any file that you have access to. This example uses the file `readme.txt`, which resides in the Windows directory `C:\temp`.

After you select a file and click **OK**, all form data is sent to the SAS Stored Process Web Application, which forwards the data to the server. As a result, the following SAS macro variables are created:

**Table 7.5**   *SAS Macro Variables*

| Variable Name | Value | Description |
| --- | --- | --- |
| _WEBIN_CONTENT_LENGTH | 1465 | Specifies the size of the file that was uploaded in bytes (supplied automatically by the web browser). |
| _WEBIN_CONTENT_TYPE | text/plain | Specifies the content type that corresponds to the file that was uploaded (supplied automatically by the web browser). |
| _WEBIN_FILE_COUNT | 1 | Specifies the number of files that were uploaded. |
| _WEBIN_FILEEXT | txt | Specifies the extension of the file that was uploaded. |
| _WEBIN_FILENAME | C:\temp\README.txt | Specifies the name and original location of the file that was uploaded. |
| _WEBIN_FILEREF | #LN00197 | Specifies the SAS fileref that you can use to access the uploaded file. This fileref is assigned for you by the SAS server. |
| _WEBIN_NAME | myfile | Specifies the value that corresponds to the NAME attribute of the INPUT tag. |

Your stored process has access to the uploaded file through the fileref that is stored in the value of the _WEBIN_FILEREF macro variable. The following code example returns the uploaded file to the client:

```
* Set the Content-type header;
%let RV = %sysfunc(stpsrv_header(Content-type, &_WEBIN_CONTENT_TYPE));

*  Write the file back to the web browser;
```

```
   data _null_;
      length data $1;

      infile &_WEBIN_FILEREF recfm=n;
      file _webout recfm=n;
      input data $char1. @@;
      put data $char1. @@;
   run;
```

The preceding code example shows how to use the _WEBIN_CONTENT_TYPE macro variable to set the content-type header. This code also shows how to use the _WEBIN_FILEREF macro variable to access the uploaded file.

# Example 2: Uploading Multiple Files

The following figure shows a custom input form that can be used to upload multiple files to the stored process server:
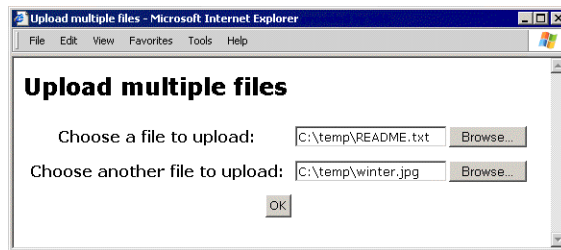


Here is an HTML example for uploading multiple files:

```
<form action="StoredProcessWebApplicationURL" method="post"
   enctype="multipart/form-data">
<input type="hidden" name="_program" value="/Path/StoredProcessName">
<table border="0" cellpadding="5">
   <tr>
      <th>Choose a file to upload:</th>
      <td><input type="file" name="firstfile"></td>
   </tr>
   <tr>
      <th>Choose another file to upload:</th>
      <td><input type="file" name="secondfile"></td>
   </tr>
   <tr>
      <td colspan="2" align="center"><input type="submit" value="OK"></td>
   </tr>
</table>
</form>
```

Example 2 uses the files `readme.txt` and `winter.jpg`, which reside in the Windows directory `c:\temp`. Note that the two input files do not need to be in the same directory.

After you select a file and click **OK**, all form data is sent to the SAS Stored Process Web Application, which forwards the data to the server. As a result, the following SAS macro variables are created:

*Table 7.6* SAS Macro Variables

| Variable Name | Value | Description |
| --- | --- | --- |
| _WEBIN_CONTENT_LENGTH | 1465 | Specifies the size of the first file that was uploaded in bytes (supplied automatically by the web browser). |
| _WEBIN_CONTENT_LENGTH0 | 2 | Specifies the number of files that were uploaded. |
| _WEBIN_CONTENT_LENGTH1 | 1465 | Specifies the size of the first file that was uploaded in bytes (supplied automatically by the web browser). |
| _WEBIN_CONTENT_LENGTH2 | 5367 | Specifies the size of the second file that was uploaded in bytes (supplied automatically by the web browser). |
| _WEBIN_CONTENT_TYPE | text/plain | Specifies the content type that corresponds to the first file that was uploaded (supplied automatically by the web browser). |
| _WEBIN_CONTENT_TYPE0 | 2 | Specifies the number of files that were uploaded. |
| _WEBIN_CONTENT_TYPE1 | text/plain | Specifies the content type that corresponds to the first file that was uploaded (supplied automatically by the web browser). |
| _WEBIN_CONTENT_TYPE2 | image/jpeg | Specifies the content type that corresponds to the second file that was uploaded (supplied automatically by the web browser). |
| _WEBIN_FILE_COUNT | 2 | Specifies the number of files that were uploaded. |
| _WEBIN_FILEEXT | txt | Specifies the extension of the first file that was uploaded. |

| Variable Name | Value | Description |
| --- | --- | --- |
| _WEBIN_FILEEXT0 | 2 | Specifies the number of files that were uploaded. |
| _WEBIN_FILEEXT1 | txt | Specifies the extension of the first file that was uploaded. |
| _WEBIN_FILEEXT2 | jpg | Specifies the extension of the second file that was uploaded. |
| _WEBIN_FILENAME | C:\temp \README.txt | Specifies the name and original location of the first file that was uploaded. |
| _WEBIN_FILENAME0 | 2 | Specifies the number of files that were uploaded. |
| _WEBIN_FILENAME1 | C:\temp \README.txt | Specifies the name and original location of the first file that was uploaded. |
| _WEBIN_FILENAME2 | C:\temp\winter.jpg | Specifies the name and original location of the second file that was uploaded. |
| _WEBIN_FILEREF | #LN00014 | Specifies the SAS fileref that you can use to access the first file that was uploaded. |
| _WEBIN_FILEREF0 | 2 | Specifies the number of files that were uploaded. |
| _WEBIN_FILEREF1 | #LN00014 | Specifies the SAS fileref that you can use to access the first file that was uploaded. |
| _WEBIN_FILEREF2 | #LN00016 | Specifies the SAS fileref that you can use to access the second file that was uploaded. |
| _WEBIN_NAME | firstfile | Specifies the value that corresponds to the NAME attribute of the first INPUT tag. |

| Variable Name | Value | Description |
| --- | --- | --- |
| _WEBIN_NAME0 | 2 | Specifies the number of files that were uploaded. |
| _WEBIN_NAME1 | firstfile | Specifies the value that corresponds to the NAME attribute of the first INPUT tag. |
| _WEBIN_NAME2 | secondfile | Specifies the value that corresponds to the NAME attribute of the second INPUT tag. |

# Examples of How to Use Uploaded Files

## Example 3: Uploading a CSV File to a SAS Table

After you have uploaded a comma-separated values (CSV) file, you can use the IMPORT procedure to import the file to a SAS table. The following sample code shows one way of achieving this:

```
%let CSVFILE=%sysfunc(pathname(&_WEBIN_FILEREF));

proc import datafile="&CSVFILE"
   out=work.mydata
   dbms=csv
   replace;
   getnames=yes;
run;

title 'First 10 records of CSV file after importing to a SAS table.';

%STPBEGIN;
   proc print data=work.mydata(obs=10); run; quit;
%STPEND;
```

Because the IMPORT procedure requires a full path to the CSV file, you must first use the PATHNAME function to get the path to the file. The GETNAMES statement uses the data in the first row of the CSV file for the SAS column names. For more information, see the IMPORT procedure in the *Base SAS Procedures Guide*.

An alternative method is to write a DATA step to import the CSV file. This method requires only Base SAS. The following code is an example of how to do this:

```
data work.mydata;
   infile &_WEBIN_FILEREF dlm=',' dsd;
```

```
   *  Your code to read the CSV file;
run;
```

# Example 4: Uploading an Excel XML Workbook to Multiple SAS Tables

Starting with Excel XP (Excel 2002), a workbook can be saved as an XML file. This XML file can be read into SAS using the SAS XML LIBNAME engine and an XMLMap. Each worksheet in the workbook is imported to a SAS table with the same name. The column headings in the worksheets are used for the column names in the SAS tables. The following code is an example of how to do this. Be sure to include the appropriate directory paths.

```
%let XMLFILE=%sysfunc(pathname(&_WEBIN_FILEREF));

   *  Include the XLXP2SAS macro;
%include 'loadxl.sas';
   *  Import the workbook into SAS tables;
%XLXP2SAS(excelfile=&XMLFILE,
   mapfile=excelxp.map);
```

The %INCLUDE statement makes the XLXP2SAS macro available to SAS. The %XLXP2SAS macro imports the data from all the worksheets into separate SAS tables with the help of an XMLMap. For more information, see the paper "Creating AND Importing Multi-Sheet Excel Workbooks the Easy Way with SAS" at http://support.sas.com/rnd/papers. Links are available for you to download both the macro and the XMLMap.

# Example 5: Uploading a SAS Table or View

When a SAS data type (table, view, or catalog) has been uploaded, additional reserved macro variables are created. For example, the following macro variables are created if the file `C:\temp\djia.sas7bdat` has been uploaded:

*Table 7.7*  *SAS Macro Variables*

| Variable Name | Value | Description |
| --- | --- | --- |
| _WEBIN_SASNAME | _B3FF5FCAF39482 D93793AEEF05BB 15F | Specifies a unique name for the uploaded SAS table, which is stored in the Work library. |
| _WEBIN_SASNAME_ORI | djia | Specifies the original name of the uploaded SAS table. |
| _WEBIN_SASTYPE | DATA | Specifies the type of SAS file that was uploaded: DATA for a |

| Variable Name | Value | Description |
|---|---|---|
| | | SAS table; VIEW for a SAS view. |

To print the SAS table or view that has been uploaded, use the following code:

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

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

# Example 6: Uploading a SAS Catalog

You can use the following sample code to list the contents of a SAS catalog that has been uploaded:

```
  %STPBEGIN;
     proc catalog c=&_WEBIN_SASNAME;
        contents;
     run; quit;
  %STPEND;
```

# Example 7: Uploading a SAS Table, View, or Catalog and Saving a Permanent Copy

You can use the following sample code to make a permanent copy of a SAS table, view, or catalog that has been uploaded and to retain the name of the original uploaded file:

```
  proc datasets library=YourLibrary;
     copy in=work out=YourLibrary memtype=&_WEBIN_SASTYPE;
        select &_WEBIN_SASNAME;
     run;
        change &_WEBIN_SASNAME=&_WEBIN_SASNAME_ORI;
  run;
  quit;
```

In the preceding example of SAS code, you must replace *YourLibrary* with the name of the SAS library in which you want to store the SAS table, view, or catalog.

## Example 8: Uploading an Excel Workbook to a SAS Table

You can use the IMPORT procedure to import an Excel workbook file that has been uploaded to a SAS table. The following sample code shows one way of achieving this:

```
%let XLSFILE=%sysfunc(pathname(&_WEBIN_FILEREF));

proc import datafile="&XLSFILE"
   out=work.mydata
   dbms=excel
   replace ;
   getnames=yes;
run; quit;

title 'First 10 records of Excel workbook after importing to a SAS table.';

%STPBEGIN;
   proc print data=work.mydata(obs=10); run; quit;
%STPEND;
```

Because the IMPORT procedure requires a full path to the Excel workbook, you must first use the PATHNAME function to get the path to the file. The GETNAMES statement uses the data in the first row of the workbook for the SAS column names. For more information, see the IMPORT procedure in the *Base SAS Procedures Guide*.

# Authentication in the SAS Stored Process Web Application

## Logon Manager and Basic Authentication

Starting with SAS 9.2, the default way for a user to log on to the SAS Stored Process Web Application is to use the Logon Manager. This is the standard mechanism used by SAS web products. The user enters credentials in a logon dialog box. After verifying the user credentials, the Logon Manager forwards to the URL that was entered for the SAS Stored Process Web Application.

To log on using the same Basic authentication that was used in previous releases of the SAS Stored Process Web Application, use the following URL:

```
http://yourserver.com:8080/SASStoredProcess/do1
```

This URL bypasses the Logon Manager and enables the SAS Stored Process Web Application to handle the user verification. The SAS Stored Process Web Application sends an HTTP status 401 to force the web browser to display a logon dialog box. To enable this capability, the AllowBasicAuthentication configuration parameter must be set to true. Use the Configuration Manager in SAS Management Console to set this parameter. Expand the **Configuration Manager** group on the Plug-ins tab in SAS Management Console, and then expand the **SAS Application Infrastructure** node. Right-click the **Stored Process Web App 9.4** node and select **Properties**. In the Properties dialog box, click the Advanced tab. Set the AllowBasicAuthentication configuration parameter to `true`.

**Note:** The Login Manager uses a fully expanded host name by default. If you use a shortened host name, then you are prompted for a user name and password for every page of the web application.

# Anonymous Access

Starting with SAS 9.2, users can run stored processes without having to log on. A guest user name can be defined to run stored processes under a fixed account. The guest user name and password are specified in the SAS Stored Process Web Application initialization parameters.

The default guest account is the anonymous web account, usually named webanon, that was defined during the system installation. If this account was not created, or if you want to specify a different account, then the initialization parameters GuestUsername and GuestPassword are used to define a guest account. The encoded value of the GuestPassword parameter can be used for the GuestPassword property value, which can be obtained as follows:

```
PROC PWENCODE
in="mypassword";
run;
```

To enable guest access, you must perform the following steps to configure the advanced properties of the SAS Stored Process Web Application:

1   In SAS Management Console, click the **Plug-ins** tab. Under **Application Management**, expand the **Configuration Manager** group.

2   Expand the **SAS Application Infrastructure** node.

3   Right-click the **Stored Process Web App 9.4** node and select **Properties**.

4   In the Properties dialog box, click the **Advanced** tab.

5   Double-click the property value for the **AllowGuest** property, and change the value to `true` in order to grant guest access to the Anonymous Web User.

6   Starting with SAS 9.4M3, you must use one or both of the following options:

■   To define a guest account, you can add the **GuestUsername** and **GuestPassword** initialization parameters. To add the **GuestUsername** and **GuestPassword** parameters, click **Add** and enter the property name and desired value for each.

■   To enable users to authenticate as the anonymous web user without entering any credentials, you can set the value of the **App.AllowGuest** property to `true`. This property also causes a **Guest** option to display on the SAS Logon Manager sign-in screen.

After you modify the advanced properties for the SAS Stored Process Web Application in the Configuration Manager, you must restart the web application server. A URL similar to the following can then be used to access the SAS Stored Process Web Application by using the guest account:

`http://yourserver.com:8080/SASStoredProcess/guest`

If the guest account is defined as an internal account, then any requests that use a workspace server will fail, including prompts that use dynamically generated lists and prompts that have dependencies.

# Other Authentication Options

The values _username and _password can be given as input parameters in the URL in order to bypass any login dialog box. The password value can be encoded as shown previously if the initialization parameter AllowEncodedPassword is set to `true`.

If a user name is defined on the host server but is not defined in metadata, then the user is considered a member of the Public group. By default, the Public group does not have permission to execute stored processes. You can use the Authorization Manager in SAS Management Console to assign ReadMetadata permission to the Public group, which enables these users to execute stored processes. For more information about using the Authorization Manager, see the product help.

To allow public IDs in the SAS Stored Process Web Application, you must perform the following steps to configure the advanced properties of the SAS Stored Process Web Application:

1   In SAS Management Console, click the **Plug-ins** tab. Under **Application Management**, expand the **Configuration Manager** group.

2   Expand the **SAS Application Infrastructure** node.

3   Right-click the **Stored Process Web App 9.4** node and select **Properties**.

4   In the Properties dialog box, click the **Advanced** tab.

5   Double-click the property value for the **App.PublicIdAllowed** property, and change the value to `true`.

To allow single system sign-on, you can use web server trusted authentication with the Logon Manager. To set up your system for trusted authentication see the *SAS Intelligence Platform: Security Administration Guide*.

To log off, the variable _ACTION=LOGOFF can be sent to SAS Stored Process Web Application. This forces the current session to be immediately deleted, and a logoff screen is displayed.

# Using the SAS Stored Process Web Application Pages

## Welcome Page

To execute the SAS Stored Process Web Application, enter the application's URL in the web browser. Either the default Welcome page or static data is displayed.

Here is an example of a URL for the SAS Stored Process Web Application:

```
http://yourserver.com:8080/SASStoredProcess/do
```

This is the default URL. If the SAS Stored Process Web Application is accessed as http://*yourserver*:8080/SASStoredProcess, then it defaults to http://*yourserver*:8080/ SASStoredProcess/do. In this example, if the Welcome.jsp file is installed, then the Welcome page is displayed. The Welcome page might look like this one:



The Welcome page contains the following links:

**Stored Process Samples**
Click this link to display a page of stored process samples that are installed with the SAS Web Infrastructure Platform.

**List Available Stored Processes**
Click this link to display a page that contains a tree view of folders, stored processes, and stored process reports. You can select a stored process (or stored process report) in the tree view in order to run the stored process. For more information, see "Tree View" on page 111. If there are no parameters or input forms, then the stored process executes immediately and the results are displayed. If there are parameters or input forms, then you are taken to the custom input form or prompt page.

**Search for Stored Processes and Reports**
Click this link to search for a stored process or stored process report. You can search for a string within the name, description, or keywords for a stored process or stored process report. No wildcards are accepted in the search term. You can also select one or more of the following columns to display in the search results: description, keywords, creation date, and modified date. For more information, see "Search Page" on page 118.

In the preceding example, if the Welcome.jsp file has not been installed, then static data is displayed. The static data might look like this:

```
Stored Process Web Application
Version 9.4 (Build 499)
```

Instead of navigating through this interface from the Welcome page, you can also use the _ACTION and _PROGRAM variables in the URL to open different pages. For more information, see "Using Reserved Macro Variables" on page 26.

# Tree View

You can access the tree view of stored processes by appending the _ACTION variable with a value of INDEX (_ACTION=INDEX) to the SAS Stored Process Web Application URL. On the left, the tree view displays the same list of folders, stored processes, and stored process reports that you see when you click **List Available Stored Processes** on the Welcome page. When you click a stored process in the tree view, the default action is to execute that stored process.

**Note:** Any stored process that has the **Hide from user** check box selected in SAS Management Console does not show up in the tree view.

You can use the _PATH variable with _ACTION=INDEX to control what level the tree view begins with. For example, if you specify `_PATH=/Products/SAS Intelligence Platform/Samples` on the SAS Stored Process Web Application URL, then the `Samples` folder is at the top of the tree view.

The keywords `properties`, `form`, and `execute` can be added to the _ACTION variable to control which page is displayed when you select a stored process as shown in the following examples:

_ACTION=INDEX,PROPERTIES
displays the prompt page for the stored process. If there are no prompts, then only the **Run** button is displayed.

_ACTION=INDEX,FORM,EXECUTE
   displays the custom input form (if available). Otherwise, the stored process
   executes.

_ACTION=INDEX,FORM,PROPERTIES,EXECUTE
   displays the custom input form (if available). If there is no custom input form,
   then the prompt page for the stored process is displayed. If there are no
   prompts, then the stored process executes.

# Summary Pages

## Stored Process Summary Page

To display the summary page for a stored process, specify
_ACTION=DATA&_PROGRAM=<*stored-process-path*> on the SAS Stored Process
Web Application URL. (You can also add _ACTION=INDEX,DATA to the SAS
Stored Process Web Application URL and then select a stored process in order to
display this page.)

| Stored Process *Sample: Hello World* | |
| --- | --- |
| Metadata path | /Products/SAS Intelligence Platform/Samples/Sample: Hello World |
| Source code location | C:\Program Files\SASHome\SASFoundation\9.4\inttech\sample |
| Source file | stphello.sas |
| SAS server type | Stored Process Server |
| Result type | Stream |
| Created | August 13, 2012 11:32:24 AM EDT |
| Last modified | August 31, 2012 10:24:16 AM EDT |
| Keywords | Data _NULL_, PUT, _Sample |
| Description | DATA Step-generated output using PUT statements. |

Run

The following items are included on this page:

Metadata path
   specifies the location of the SAS folder that contains the stored process. You can
   use this path as the value for the _PROGRAM variable. For more information,
   see "Using Reserved Macro Variables" on page 26.

Source code location
> specifies the source code repository, where the SAS code is located.

Source file
> specifies the name of the file that contains the SAS code.

SAS server type
> specifies the type of server that is used to run the stored process (either a stored process server or a workspace server).

Result type
> specifies the type of results that the stored process is capable of producing (can be Stream, Package, both, or neither of these).

Created
> specifies the date and time that the stored process metadata was first registered.

Last modified
> specifies the date and time that the stored process metadata was last modified.

Keywords
> specifies any keywords that are associated with the stored process. These keywords are part of the stored process metadata.

Description
> contains a description of the stored process. This description is part of the stored process metadata.

To run the stored process, click **Run** at the bottom of the summary. If there are no parameters or input forms, then the stored process executes immediately and the results are displayed. If there are parameters or input forms, then you are taken to the custom input form or prompt page.

# Stored Process Report Summary Page

To display the summary page for a stored process report, specify _ACTION=DATA&_REPORT=<*stored-process-report-path*> on the SAS Stored Process Web Application URL. (You can also add _ACTION=INDEX,DATA to the SAS Stored Process Web Application URL and then select a stored process report in order to display this page.)

| Stored Process Report *Multiple Output Formats* | |
|---|---|
| Metadata path | /Users2/reports/Multiple Output Formats |
| Number of reports | 1 |
| Maximum retained | 1 |
| Created | August 27, 2010 12:51:43 PM EDT |
| Last modified | August 27, 2010 12:51:43 PM EDT |
| Keywords | |
| Description | |

Run

| ID | Creator | Creation Date | Expiration Date |
|---|---|---|---|
| 2 | sasadm | November 17, 2010 4:24:14 PM EST | |

The following items are included on this page:

Metadata path
specifies the location of the SAS folder that contains the stored process report. You can use this path as the value for the _REPORT variable. For more information, see "Using Reserved Macro Variables" on page 26.

Number of reports
specifies the number of stored process report generations that are currently available.

Maximum retained
specifies the maximum number of stored process report generations that can be saved at any time.

Created
specifies the date and time that the stored process report was first created.

Last modified
specifies the date and time that the stored process report was last modified.

Keywords
specifies any keywords that are associated with the stored process report.

Description
contains a description of the stored process report.

The bottom of the summary page shows all the generations of the stored process report that are available, including who created each one and when, and when each stored process report generation expires. You can click the number of a stored process report generation in order to view that generation. To run a new stored process report, click **Run** at the bottom of the summary.

To run a stored process report from the URL, the metadata location must be specified. This can be done using the parameters `_PROGRAM=/myfolder/myreport&_TYPE=report`. Alternatively, just the _REPORT parameter can be used, for example, `_REPORT=/myfolder/myreport`. This returns the latest stored process report generation or runs a new one if none are found. To return a specific stored process report, the parameter _REPORTID=ID can be added to the URL with the ID of the desired generation of the report. A _REPORTID=0 forces a new stored process report to be run.

**Note:** The stored process log is available only when the stored process report is run to generate new output.

# Custom Input Form

If you want the SAS Stored Process Web Application to display a custom input form for a stored process, then you can use any of the following methods:

■ On the Welcome page, click the **Stored Process Samples** link to display a page of stored process samples that are installed with the SAS Web Infrastructure Platform. Each of these samples has a link and a description. Click any of these links to display the custom input form for that stored process.

■ Use the _PROGRAM variable along with _ACTION=FORM in the URL to display the custom input form for a stored process. For more information, see "Specifying Custom Input Forms" on page 95.

■ Select a stored process from the tree view. If the stored process has a custom input form, then it is displayed.

A custom input form might look like this:

You can use the custom input form to execute the stored process. In this example, clicking **Display SAS Output** generates the following results:
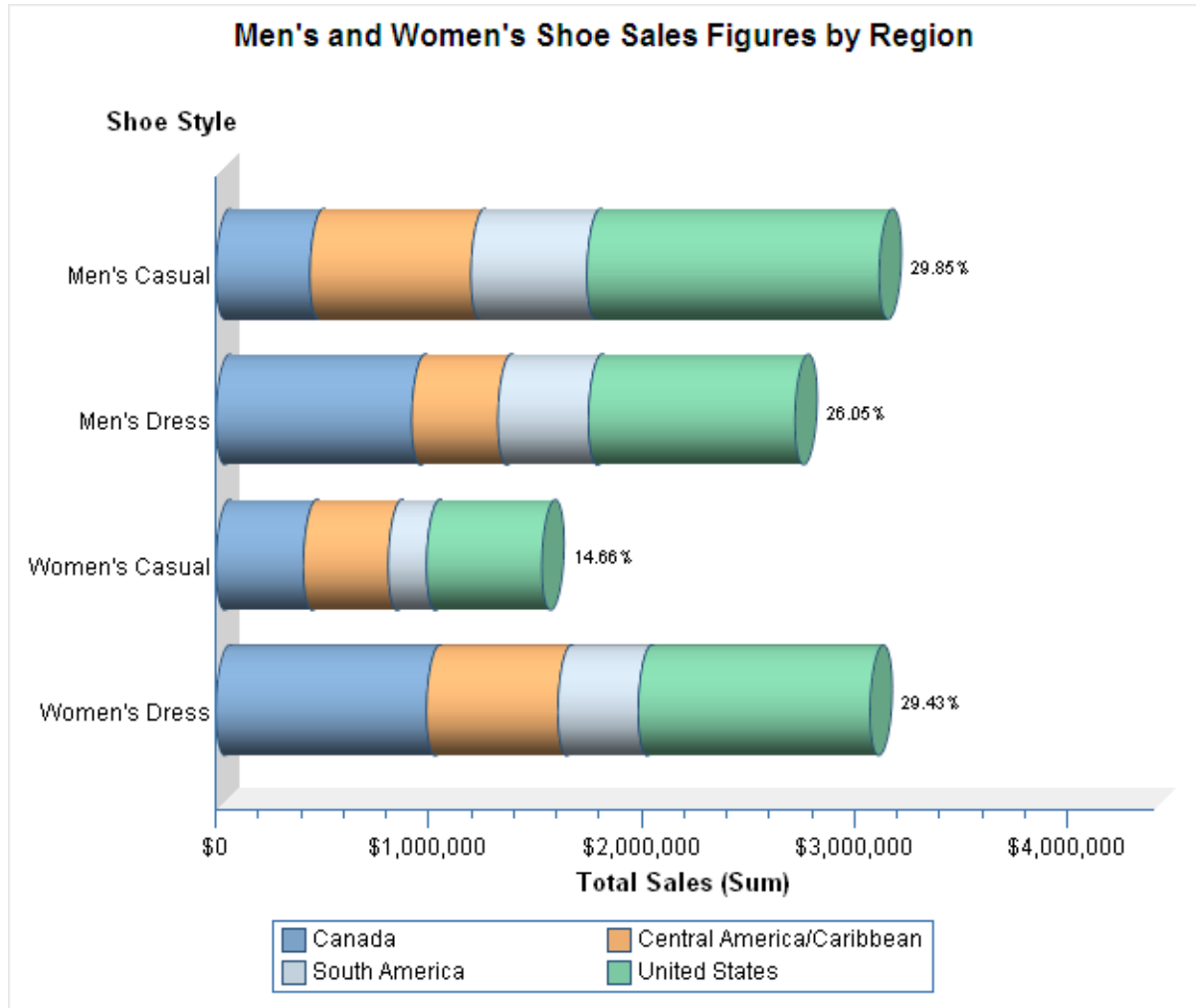


# Prompt Page

In order to display the prompt page for a stored process, you can do one of the following:

- Use the _PROGRAM variable along with _ACTION=PROPERTIES in the URL to display the prompt page for a stored process. For more information, see "Specifying Prompt Pages" on page 97.

- Select a stored process from the tree view. If the stored process has parameters but does not have a custom input form, then the prompt page is displayed.

.......................................................................................................................

**Note:** If a stored process does not have any parameters, then it does not have a prompt page. If you click a stored process that does not have a prompt page or a custom input form, then the stored process is immediately executed.

.......................................................................................................................

If you have defined parameters groups, then the groups are shown as items in the menu on the left side of the prompt page. You can click each group name to display the parameters that are in that group. A prompt page without groups might look like this:



This sample prompt page shows an example of a date range type parameter. Parameters must be defined in the stored process metadata in order for them to be visible in a prompt page. Parameters are called prompts in SAS Management Console. The stored process metadata is where the name, label, type, default value, and any constraints for a prompt are defined. Constraints help define which values are allowed for a prompt, how many values are allowed, and so on.

The prompt type determines how that parameter is displayed in the prompt page. You can have any of the following types of prompts in this page:

- Text
- Text range
- Numeric
- Numeric range
- Date
- Date range
- Time
- Time range
- Timestamp
- Timestamp range
- Color
- Data source

- File or directory

- Data library

For more information about how to create prompts and the constraints that can be specified for each type of prompt, see the Help for prompts in SAS Management Console. For more information about how to specify values for prompt, and macro variables that are generated by prompts, see Appendix 3, "Formatting Prompt Values and Generating Macro Variables from Prompts," on page 209.

# Execution Options

Execution options are delivered as a sample shared prompt group that you can add to a stored process for use with the prompt page. Execution options are prompts that enable you to specify the graphic device, ODS destination, ODS style, and debugging options for a stored process at run time.

To add execution options to a stored process, perform the following steps:

1 Open the Stored Process Properties dialog box for the stored process and click the **Parameters** tab.

2 Select **Add Shared**.

3 Under SAS Folders, navigate to `/Products/SAS Intelligence Platform/ Samples`.

4 Select `Execution Options`.

5 Click **OK**.

The following table contains a list of the execution options and the SAS macro variables that they represent:

*Table 7.8  Execution Options*

| Execution Option | SAS Variable Name (Prompt Name) |
| --- | --- |
| Graphic device | _GOPT_DEVICE |
| Output format | _ODSDEST |
| ODS style | _ODSSTYLE |
| Debug options | _DEBUG |

# Search Page

You can access the search page for stored processes and stored process reports by appending the _ACTION variable with a value of SEARCH (_ACTION=SEARCH) to

the SAS Stored Process Web Application URL. This is the seach.jsp file and is the same page that is displayed if you click **Search for Stored Processes and Reports** on the Welcome page. This form enables you to enter a search term and select what columns are to be displayed in the search results. No wildcards are accepted in the search term. The default behavior is to search the name fields of both stored processes and stored process report for the specified string, and to return only a Stored Processes column (which displays both stored processes and stored process reports, as well as the folders that contain these objects) and a Description column. You can either use the fields in the search page, or combine _ACTION=SEARCH with values for the _MATCH, _FIELD, _COLUMNS, _TYPE, or _PATH variables in order to modify search parameters.



If you search for stored processes with `Sample` in their name and specify that you want all the columns to display, then the search results might look like the following table. You can click a stored process in the search results to execute that stored process. You can click a stored process report to display the last generation of that stored process report. If the stored process report has not been previously run, then it is run and then displayed. You can use the right mouse button on the column headings to sort the display by that column.

| Stored Processes | Description | Keywords | Created | Modified |
|---|---|---|---|---|
| ⊟ 🗁 Stored Processes | | | | |
| ⊟ 🗁 Products | | | | |
| ⊟ 🗁 SAS Intelligence Platform | | | | |
| ⊟ 🗁 Samples | | | | |
| Sample STP Report: Cholesterol by Sex and Age Group | Stored process report that creates box plots using ODS. | PROC SGPANEL, ODS, STPBEGIN, STPEND | 2012-08-13 | 2012-08-13 |
| Sample: Cholesterol by Sex and Age Group | Creates box plots using ODS. | PROC SGPANEL, ODS, STPBEGIN, STPEND, _Sample | 2012-08-13 | 2012-08-13 |
| Sample: European Demographic Data | Dynamically generated map with drilldown capabilities using ODS. | PROC GMAP, ODS, STPBEGIN, STPEND, Java, ActiveX, GIF, _Sample | 2012-08-13 | 2012-08-31 |
| Sample: Frequency Analysis of Municipalities | Uses ODS to generate output. | PROC FREQ, ODS, STPBEGIN, STPEND, PROC PRINT, _Sample | 2012-08-13 | 2012-08-31 |
| Sample: Hello World | DATA Step-generated output using PUT statements. | Data _NULL_, PUT, _Sample | 2012-08-13 | 2012-08-31 |
| Sample: Multiple | Uses ODS to generate PDF, | PROC PRINT, ODS, STPBEGIN, STPEND, | | |

**Note:** Any stored process that has the **Hide from user** check box selected in SAS Management Console does not show up in the search results.

# XML Output

In addition to returning HTML output, the data for the various SAS Stored Process Web Application displays can be obtained in XML format. The keyword `xml` can be added to the _ACTION parameter to request that the corresponding data be returned as XML. The _ACTION values for the tree, data, properties, and search displays can be modified with the XML qualifier. Stand-alone clients such as Flex based applications can use the XML data to build non-HTML displays.

# Using HTTP Headers

## Overview of HTTP Headers in Stored Processes

Stored process streaming output is always accompanied by an HTTP header. The HTTP header consists of one or more header records that identify the content type of the output and can provide other information such as encoding, caching, and expiration directives. A streaming stored process client can use or ignore the HTTP header. The SAS Stored Process Web Application forwards the HTTP client to the web browser (or other HTTP client).

HTTP headers are defined by the HTTP protocol specification (RFC 2616), which can be found at http://www.w3.org. Each header record is a single text line consisting of a name and a value separated by a colon (:). The following example shows records in an HTTP header:

```
Content-type: text/html; encoding=utf-8
Expires: Wed, 03 Nov 2004 00:00:00 GMT
Pragma: nocache
```

You can set any HTTP record for your stored process output by calling the STPSRV_HEADER function. For more information, see "STPSRV_HEADER Function" on page 55. Typically, you must call STPSRV_HEADER before the %STPBEGIN statement. The following DATA step function calls generate the previous example header records:

```
old = stpsrv_header("Content-type",
"text/html; encoding=utf-8");
old = stpsrv_header("Expires",
"Wed, 03 Nov 2004 00:00:00 GMT");
old = stpsrv_header("Pragma", "nocache");
```

You can also call this function directly from SAS macro code outside a DATA step. Note that string parameters are not enclosed in quotation marks, and macro characters such as semicolon (;) must be masked in this case:

```
%let old = %sysfunc(stpsrv_header(Content-type,
text/html%str(;) encoding=utf-8);
%let old = %sysfunc(stpsrv_header(Expires,
```

```
Wed, 03 Nov 2004 00:00:00 GMT));
%let old = %sysfunc(stpsrv_header(Pragma, nocache));
```

Headers must be set before _WEBOUT is opened. There are several ways that
_WEBOUT can be opened. Here are some examples:

- ```
  data _null_;
  file _webout;
  ...;
  run;
  ```

- ```
  %STPBEGIN; *  if the stored process creates streaming output;
  ```

- ```
  ods html body=_webout ... ;
  ```

# Commonly Used Headers

The following are a few commonly used HTTP header records:

- Content-type
- Expires
- Location
- Pragma
- Set-Cookie
- Status-Code

# Content-type

The `Content-type` header record is generated automatically. The value is set based
on the ODS destination that you use in your stored process. The value is
determined by looking up the ODS destination in the file types section of the SAS
registry and, if appropriate, the Windows registry. If you do not use ODS to generate
the output, then `Content-type` defaults to **text/html**. Use the STPSRV_HEADER
function if you want to override the default value. Override the value of `Content-type` when you want to do any of the following:

- specify the encoding of the data. This might be important in web applications
  where the client (typically a web browser) might expect a different encoding than
  the stored process output. Examples:

  ```
  Content-type: text/xml; encoding=utf-8
   Content-type: text/plain; encoding=iso-8859-1
   Content-type: text/html; encoding=windows-1252
  ```

- direct the output to a specific content handler. For example, HTML output can be
  directed to Microsoft Excel (in later versions of Microsoft Office) by setting the
  `Content-type` to **application/vnd.ms-excel**.

- override the default **text/html** value. Overriding this value typically occurs if you
  are using ODS custom tagsets or you are not using ODS at all to generate the
  output.

The following table shows commonly used `Content-type` values.

*Table 7.9   Content Types*

| Content-type | Description |
| --- | --- |
| application/octet-stream | Unformatted binary data. |
| image/gif | GIF (Graphics Interchange Format) images. |
| image/jpeg | JPEG (Joint Photographic Expert Group) format images. |
| image/png | PNG (Portable Network Graphics) format images. |
| text/html | HTML (Hypertext Markup Language). |
| text/plain | Plain unformatted text. |
| text/xml | XML (eXtensible Markup Language). |
| text/x-comma-separated-values | Spreadsheet data. |

`Content-type` values are also known as MIME types. For a list of all official MIME types, see the IANA registry at http://www.iana.org/assignments/media-types/. An unregistered MIME type or subtype can be used; the value should be preceded by `x-`.

# Expires

Web clients frequently cache HTML and other content. Accessing the same URL might return the cached content instead of causing the output to be regenerated by the server. Accessing the cached content is often desirable and reduces server and network loads, but can lead to unexpected or stale data. The `Expires` header record enables you to control how long a web client caches the content.

The `Expires` header record requires that the expiration time be specified in Greenwich Mean Time (GMT) and in a particular format. A SAS picture format can be used to create this value. Use PROC FORMAT to create a custom format as shown in the following example:

```
proc format;
picture httptime (default=29)
other='%a, %0d %b %Y %0H:%0M:%0S GMT'
(datatype=datetime);
run;
```

This format can be created one time and saved in a global format library, or you can create it dynamically as needed in your stored process. The format generates a date in this form:

```
     Sun, 24 AUG 2003 17:13:23 GMT
```

DATA step functions can then be used to set the desired expiration time, adjust to GMT, and format, as shown in the following examples:

```
/* Expire this page in six hours */
data _null_;
exptime = datetime() + '6:00:00't;
old = stpsrv_header('Expires',
put(exptime - gmtoff(), httptime. ));
run;

/* Expire this page at the beginning of next
week (Sunday, 00:00 local time) */
data _null_;
exptime = intnx('dtweek', datetime(), 1);
old = stpsrv_header('Expires',
put(exptime - gmtoff(), httptime. ));
run;
```

Specifying an expiration time in the past causes caching to be disabled for your output. It is recommended that you also use the Pragma header record in this case. For more information, see "Pragma" on page 124. Specify an expiration time far in the future if you want your content to be cached indefinitely.

# Location

The `Location` header record is unlike other header records. It redirects the web client immediately to a different URL. Generally, all other header records and content are ignored when this header record is used. Use this header to redirect the client to another location for special conditions. For example, a stored process might redirect a client to a Help URL if an invalid input or other error condition is detected. For example, the following stored process redirects the web client to a static Help page when an error condition is detected:

```
%macro doSomething;

...

%if error-condition %then %do;
%let old = %sysfunc(stpsrv_header(Status-Code,302));
%let old = %sysfunc(stpsrv_header(Location,
http://myserv.abc.com/myapp/help.html));
%goto end_processing;
%end;

... normal processing ...

%end_processing:
%mend;

%doSomething;
```

The URL that is specified in the `Location` header is not limited to a static URL. It might be a SAS Stored Process Web Application or JSP URL, and it might contain parameters. In the preceding example, the erroneous request, complete with input

parameters, can be redirected to an error handling stored process. The error handling stored process can examine the input parameters and generate specific error messages and context-sensitive Help. This is one method to avoid replicating error handling or Help material across multiple stored processes.

**Note:** The `Status-Code` header must be used to set the HTTP status before the `Location` header can be used.

## Pragma

The `Pragma` header record is used to specify information not formally defined in the HTTP specification. The most commonly used value is **nocache**. This value disables web client caching of content for most web browsers. Some web browsers require that other headers be set in order to prevent caching. For example:

```
old = stpsrv_header('Expires','Thu, 18 Nov 1999 12:23:34 GMT');
old = stpsrv_header('Cache-Control','no-cache,no-store');
old = stpsrv_header('Pragma','no-cache');
```

## Set-Cookie

The `Set-Cookie` header record sends a cookie to the web client to maintain client-side state. Here is the format:

```
Set-Cookie: name=value; name2=
value2; ...; expires=date;
path=path; domain=domain_name; secure
```

where EXPIRES, PATH, DOMAIN, and SECURE are all optional. The date must be specified in the HTTP GMT format that is described in "Expires" on page 122.

For example:

```
old = stpsrv_header("Set-Cookie",
"CUSTOMER=WILE_E_COYOTE; path=/SASStoredProcess/do; " ||
"expires=Wed, 06 Nov 2002 23:12:40 GMT");
```

The next time your application is run, any matching cookies are returned in the _HTCOOK environment variable, assuming that this variable has been enabled in your SAS Stored Process Web Application environment. You must parse the cookie string to retrieve the information that you saved in the cookie. Use the `scan` DATA step function to split the name/value pairs on the semicolon (;) delimiters. Then split the name/value pairs on the equal sign (=) delimiter.

Most web browsers support cookies, but some users disable them due to privacy concerns, site policies, or other issues. If you use cookies, explain to your users why you need them and if they must be enabled in order to use your application. Some web clients might not support cookies at all.

## Status-Code

The `Status-Code` header record is used by web applications to set the HTTP status for every page that is returned to the web browser. For information about status code definitions, see http://www.w3.org.

# Embedding Graphics

## Embedding Graphics in Web Pages

Web pages frequently contain embedded graphic images. For static images, an <IMG> tag is enough to embed the image, as shown in the following example:

```
<IMG SRC="mykids.gif">
```

Dynamically generated images, such as charts that vary over time or due to input parameters, are more complicated. Stored processes can generate graphics in addition to HTML output. The following stored process creates a bar chart followed by a tabular report:

```
/* Sales by Region and Product */

%stpbegin;

title "Sales by Region and Product";
legend1 label=none frame;

proc gchart data=sashelp.shoes;
   hbar3d region / sumvar=sales
      sum space=.6
      subgroup=product
      shape=cylinder
      patternid=subgroup
      legend=legend1;
   label product='Shoe Style';
   run;

proc report data=sashelp.shoes;
   column region product sales;
   define region / group;
   define product / group;
   define sales / analysis sum;
   break after region / ol summarize suppress skip;
   run;

%stpend;
```

Depending on input parameters, this stored process might produce the following output:

*Figure 7.1*   *Web Page with Embedded Graphic*
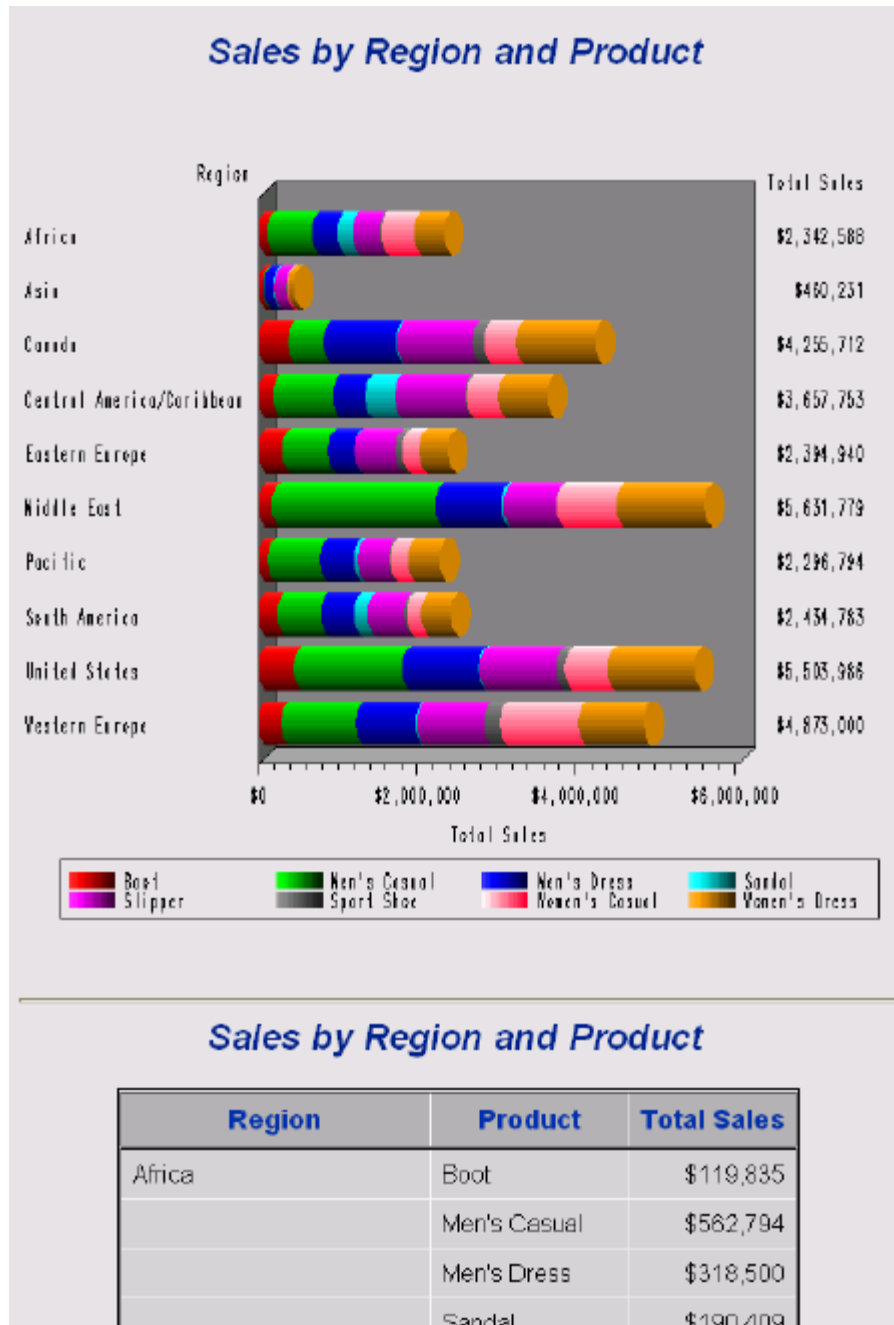


No special code was added to handle the image. ODS and the stored process framework takes care of the details of delivering both the HTML and the image to the web browser. This code handles different image types through the _GOPT_DEVICE input parameter that is supported by the %STPBEGIN macro. For more information, see "Using the %STPBEGIN and %STPEND Macros" on page 18. The image is delivered to the web browser in different ways depending on the graphics device. JAVA and ACTIVEX images are generated by embedding an <OBJECT> tag in the generated HTML that contains the attributes and parameters necessary to invoke the viewer and to display the graphic. There is no <IMG> tag in

this case. Other commonly used drivers (GIF, JPEG, PNG, ACTXIMG, and JAVAIMG) do use the <IMG> tag. The following code is an HTML fragment that is generated by the previous stored process using the GIF image driver:

```
<IMG SRC="/SASStoredProcess/do?_sessionid=
    7CF645EB-6E23-4853-8042-BBEA7F866B55
    &_program=replay&entry=
    STPWORK.TCAT0001.GCHART.GIF">
```

The image URL in the <IMG> tag is actually a reference to the SAS Stored Process Web Application that uses the special stored process named REPLAY. The REPLAY stored process takes two parameters, _SESSIONID and ENTRY. _SESSIONID is new, unique value each time the original stored process is executed. ENTRY is the name of a temporary SAS catalog entry that contains the generated image. Image replay uses a special, lightweight version of the stored process sessions feature to hold image files temporarily until the web browser retrieves them. For more information, see .

You can use the REPLAY stored process to replay entries other than embedded images, such as CSS style sheets, JavaScript include files, PDF files, and HTML or XML files to be displayed in a pop-up window, frame or <IFRAME>. The special macro variable _TMPCAT contains the name of the temporary catalog that is used for REPLAY entries. The variable _REPLAY contains the complete URL that is used to reference the REPLAY stored process (except the actual entry name). The _TMPCAT catalog remains on the server for only a limited time. If the catalog is not accessed within a time-out period (typically 15 minutes), then the catalog and its contents are deleted.

# Generating Direct Graphic Output

In some cases, you might want to generate image output directly from a stored process with no HTML container. This might be useful if you want to include a dynamically generated graphic in static HTML pages or HTML generated by an unrelated stored process, as shown in the following example:

```
/* Sales by Product - pie chart stored process
 *
 * XPIXELS and YPIXELS input parameters are required */

/* assume we want a new image generated
 * each time the image is viewed -
 * disable browser caching of this image */
%let old=%sysfunc(stpsrv_header(Pragma, nocache));

/* need test here in case XPIXELS
 * or YPIXELS are not defined */

/* set up graph display options */
goptions gsfname=_webout gsfmode=replace
    dev=png xpixels=&XPIXELS
    ypixels=&YPIXELS ftext=swiss;
pattern1 color=yellow;
pattern2 color=cyan;
pattern3 color=green;
pattern4 color=black;
```

```
pattern5 color=red;
pattern6 color=blue;

 /* create a simple pie chart */
 proc gchart data=sashelp.shoes;
    pie3d product/sumvar=sales;
    run;
    quit;
```

This stored process expects XPIXELS and YPIXELS to be passed as input parameters. A typical IMG tag to invoke this stored process might be similar to the following example:

```
<IMG SRC="/SASStoredProcess/do?_program=
    /WebApps/Utilities/Sales+by+Product&XPIXELS=
        400&YPIXELS=300">
```

which results in the following output:

*Figure 7.2* *Graphic Output*
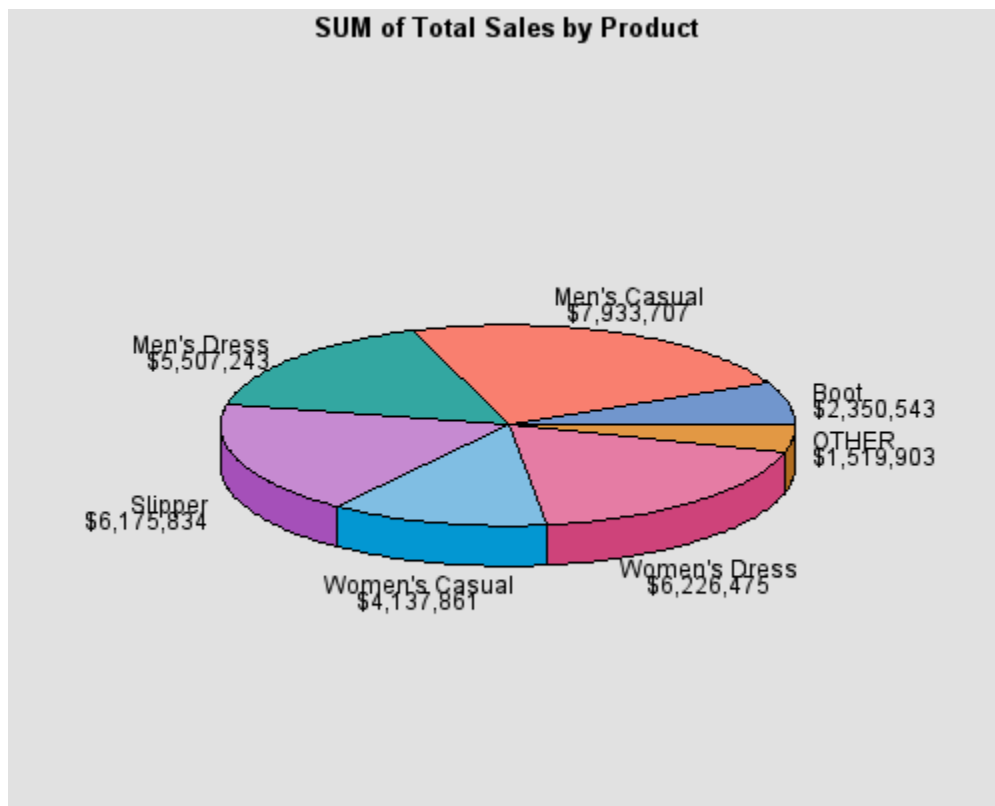


**Note:** Some web browser versions have a defect that causes the web browser to ignore the NOCACHE and Expires directives in the HTTP header. This defect causes the web browser to reuse the image that was previously generated from its cache even if the HTTP header directed that no caching was to occur. This might happen when embedding an image in an HTML page or when directly entering an image URL in the web browser. The old image might be updated by manually performing a web browser REFRESH or RELOAD, but it is difficult to work around this problem without manual intervention.

# Chaining Stored Processes

## Why Chain Stored Processes?

Only the simplest stored process web applications contain a single web page. With the addition of a second and subsequent pages, you face the problem of passing information from one page to another. Typically, an application contains more than a single stored process. This means that you must find a way to connect the stored processes that compose your application and make sure that all of the data that is collected along the way is available in the appropriate places.

It is good programming practice to design applications so that they do not request the same information multiple times. Because HTTP is a stateless environment, each request is separate from all other requests. If a user enters a phone number on the first page of an application and submits the form, that phone number is available as an input parameter to the first stored process. After that stored process completes, the input parameters are lost unless they are explicitly saved. If the second or third stored process in the application needs to know the specified phone number, then the application must ask for the phone number again. There are several ways to solve this problem. You can store data values in the following locations:

- on the client in hidden form fields or URL parameters

- on the client in cookies

- on the server by using sessions

**Note:**  When you are registering each stored process in a chain, you can check the **Hide from user** box on the **General** tab in the Stored Process Properties dialog box to hide the second and subsequent stored processes in a chain from the client application. This option hides the stored process from the folder view and search results for clients like the SAS Stored Process Web Application, so the user will not try to execute the second or subsequent stored processes in a chain directly.

## Passing Data through Form Fields or URL Parameters

Storing data on the client in hidden fields or URL parameters is the simplest technique. To do this, you must dynamically generate all of the HTML pages in your application except for the initial page. Because each page functions as a mechanism for transporting data values from the previous stored process to the next stored process, it cannot be static HTML stored in a file.

Usually, the application takes the following steps:

1   The first HTML page is a welcome or login screen for the application. After the user enters any required information, the first stored process is executed by submitting a form or clicking on a link.

2   The first stored process performs any necessary validation on the submitted parameters and any required application initialization.

3   The first stored process writes an HTML page to the _WEBOUT output stream. This HTML page might be an initial report or it might be the next navigation page in the application. Links in this page typically execute another stored process and pass user identity, user preferences, application state, or any other useful information through hidden form fields or URL parameters.

4   Each succeeding link in the application executes another stored process and passes any required information through the same technique.

Each hidden field in the second form can contain one name/value pair that is passed from the first form. You should use unique names for all of the data values in the entire application. In this way, you can pass all of the application data throughout the entire application.

When you dynamically generate the second form, you can write out the name of the second stored process in the hidden field _PROGRAM. Because the first stored process contains the logic to determine the second stored process, this is referred to as chaining stored processes. A stored process can chain to multiple stored processes depending on the link that a user chooses or on data that is entered by the users. The stored process can even chain back to itself.

In the following example, the MyWebApp application starts with a static HTML welcome page:

```
<!-- Welcome page for MyWebApp -->
<HTML>
<HEAD><TITLE>Welcome to MyWebApp
</TITLE></HEAD>
<BODY><H1>Welcome to MyWebApp</H1>
<FORM ACTION="/SASStoredProcess/do">
Please enter your first name:
<INPUT TYPE="text" NAME="FNAME"><BR>
<INPUT TYPE="hidden" NAME="_program"
VALUE="/WebApps/MyWebApp/Ask Color">
<INPUT TYPE="submit" VALUE="Run Program">
</FORM>
</BODY></HTML>
```

This welcome page prompts the user for a first name and passes the value as the FNAME input parameter to the **/WebApps/MyWebApp/Ask Color** stored process, as in the following example:

```
/* Ask Color stored process
 *
 * This stored process prompts for the user's favorite
 * and passes it to the Print Color stored process.
 */
data _null_;
file _webout;
put '<HTML>';
put '<H1>Welcome to MyWebApp</H1>';
```

```
/* Create reference back to the 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="/WebApps/MyWebApp/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) '"><BR>';

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>';
put '<INPUT TYPE="submit" VALUE="Run Program">';
put '</FORM>';
put '</HTML>';
run;
```

This stored process simply creates an HTML form that prompts the user for more information. The reserved macro variable _URL is used to refer back to the SAS Stored Process Web Application. This enables you to move the web application without modifying each stored process. The _PROGRAM variable specifies the stored process that processes the contents of the form when it is submitted. In order to keep the FNAME that was entered in the initial page, place it in the form as a hidden field. Because the value was entered by the user, it must be encoded using the HTMLENCODE function in case it contains any character that might be interpreted as HTML syntax. The form prompts the user for a color choice and chains to a new stored process named Print Color, as in the following example:

```
/* Print Color stored process
*
* This stored process prints the user's
* first name and favorite color.
*/
data _null_;
file _webout;
put '<HTML>';
fname = htmlencode("&FNAME");
put 'Your first name is <b>'
    fname +(-1) '</b>';
put '<BR>';
put "Your favorite color is
    <b>&FCOLOR</b>";
put '<BR>';
put '</HTML>';
```

```
     run;
```

The `Print Color` stored process prints the values of the variables from both the first and second forms, illustrating that the data has been correctly passed throughout the entire application.

A variation of this technique uses URL parameters instead of hidden form fields. The following example code is an alternative implementation of the `Ask Color` stored process:

```
/* Ask Color stored process
 *
 * This stored process prompts for the user's favorite
 * and passes it to the Print Color stored process.
 */
data _null_;
file _webout;
put '<HTML>';
put '<H1>Welcome to MyWebApp</H1>';

/* Build a URL referencing the next stored process.
 * Use URLENCODE to encode any special characters in
 * any parameters. */
length nexturl $500;
nexturl = "&_URL?_program=
/WebApps/MyWebApp/Print Color" ||
'&fname=' || urlencode("&FNAME");

put 'What is your favorite color?';
put '<UL>';
put '<LI><A HREF="' nexturl +(-1)
'&color=red">red</A></LI>';
put '<LI><A HREF="' nexturl +(-1)
'&color=green">green</A></LI>';
put '<LI><A HREF="' nexturl +(-1)
'&color=blue">blue</A></LI>';
put '<LI><A HREF="' nexturl +(-1)
'&color=other">other</A></LI>';
put '</UL>';
put '</HTML>';
run;
```

This stored process generates a separate URL link for each color choice. The end result is the same as the first implementation of `Ask Color`; the `Print Color` stored process is executed with both FNAME and COLOR input parameters.

The technique of passing data by using hidden fields or URL parameters has the following advantages:

- simple to perform

- easy to debug

- state is maintained indefinitely

- allows stored processes to be distributed across multiple servers

The major disadvantages of this technique are the necessity to use dynamically generated HTML for all pages in the application and the security and visibility of the data. The data in hidden fields is readily visible to the client by viewing the HTML source (and is directly visible in the URL when using GET method forms). The data

is easily changed by the user, and falsified or inconsistent data can be submitted to the application. Sensitive data should be validated in each new stored process, even if it is passed from generated hidden fields.

# Passing Data through the DATA Step

Instead of using a JSP file to create a custom input form, you can use HTML following the CARDS4 statement in a DATA step. Follow these instructions to create your custom HTML form:

- Insert your HMTL code after the CARDS4 statement in place of the existing HTML code.

- Modify the %LET STPNAME statement to specify the name of the next stored process that is to be executed in the HTML form that is created.

This sample stored process creates a custom HTML page that is used to execute another stored process. The DATA step reads the input lines and resolves any macro variables that are specified (such as &stpname and &_URL). This method provides an alternative to the approach of creating a custom JSP file that is stored on your web server.

```
%let stpname=/Samples/Stored Processes/Sample: Frequency Analysis of
   Municipalities;

data _null_;
   format infile $char256.;
   input;
   infile = resolve(_infile_);
   file _webout;
   put infile;
cards4;
<HTML>
<BODY>
<H1>Sample: Frequency Analysis of Municipalities</H1>
This sample illustrates the data analysis capabilities of stored
processes.

<FORM ACTION="&_URL">
<INPUT TYPE="HIDDEN" NAME="_program" VALUE="&stpname">
<HR>
Choose a table to display:<BR>
<INPUT TYPE=RADIO NAME="table" value="city*dept" CHECKED>City by
Dept<BR>
<INPUT TYPE=RADIO NAME="table" value="city*week">City by Week<BR>
<INPUT TYPE=RADIO NAME="table" value="dept*week">Dept by Week<BR>
<HR>
<INPUT TYPE="SUBMIT" VALUE="Run Procedure">
<INPUT TYPE="CHECKBOX" NAME="_debug" VALUE="log">Show SAS Log
</FORM>
</BODY>
</HTML>
;;;;
run;
```

## Passing Data through Cookies

HTTP cookies are packets of information that are stored in the client web browser. They are shuttled back and forth with the application requests. In a general sense, they are quite similar to hidden form fields, but they are automatically passed with every request to the application. Cookies have the advantage of being nearly invisible to the user. They contain a built-in expiration mechanism, and they are slightly more secure than hidden fields. They also work seamlessly across multiple stored process servers and web applications and are preserved even if your application uses static HTML pages. For more information about setting and using cookies, see "Set-Cookie" on page 124. You must enable HTTP cookies in your web application configuration. For more information, see "Configuring the SAS Stored Process Web Application " on page 83.

HTTP cookies can be complex to generate and parse. Carefully consider names, paths, and domains to ensure that your cookie values do not conflict with other applications that are installed on the same web server. HTTP cookies can also be disabled by some clients due to privacy concerns.

## Passing Data through Sessions

Sessions provide a simple way to save state on the server. Instead of passing all of the saved information to and from the web client with each request, a single session key is passed and the data is saved on the server. Applications must use all dynamically generated HTML pages, but the hidden fields or URL parameters are much simpler to generate. In addition, sessions provide a method to save much larger amounts of information, including temporary data sets or catalogs. Sessions have the disadvantage of binding a client to a single server process, which can affect the performance and scalability of a web application. Sessions are not recommended for simple applications that pass small amounts of data from one stored process to another. For more information, see "Using Sessions" on page 47.

# Using Sessions in a Sample Web Application

## Overview of the Sample Web Application

The following sample web application demonstrates some of the features of stored process sessions. The sample application is an online library. Users can log on, select one or more items to check out of the library, and request by email that the selected items be delivered. The sample code shows how to create a session and

then create, modify, and view macro variables and data sets in that session. For more information, see .

# Sample Data

This sample requires a LIB_INVENTORY data set in the SAMPDAT library that is used for other SAS Integration Technologies samples. You can create the data set in Windows using the following code. You can also use the code in other operating environments by making the appropriate modifications to the SAMPDAT LIBNAME statement.

```
libname SAMPDAT 'C:\My Demos\Library';
data SAMPDAT.LIB_INVENTORY;
length type $10 desc $80;
input refno 1-5 type 7-16 desc 17-80;
datalines4;
17834 BOOK SAS/GRAPH Software: Reference
32345 BOOK SAS/GRAPH Software: User's Guide
52323 BOOK SAS Procedures Guide
54337 BOOK SAS Host Companion for UNIX Environments
35424 BOOK SAS Host Companion for OS/390 Environment
93313 AUDIO The Zen of SAS
34222 VIDEO Getting Started with SAS
34223 VIDEO Introduction to AppDev Studio
34224 VIDEO Building Web Applications with
SAS Stored Processes
70001 HARDWARE Cellphone - Model 5153
70002 HARDWARE Video Projector - Model 79F15
;;;;
```

# Main Aisle Stored Process

The main aisle page is generated by the Main Aisle stored process. This page acts as a welcome page to new users. A session is created the first time a user executes this stored process.

```
/* Main Aisle of the Online Library */
data _null_;
file _webout;
if libref('SAVE') ne 0 then
rc = stpsrv_session('create');
put '<HTML>';
put '<HEAD><TITLE>Online Library
Main Aisle</TITLE></HEAD>';
put;
put '<BODY vlink="#004488" link="#0066AA"
bgcolor="#E0E0E0">';
put '<H1>Online Library Main Aisle</H1>';
put;
put 'Select one of the following
areas of the library:';
```

```
put '<UL>';
length hrefroot $400;
hrefroot = symget('_THISSESSION') ||
'&_PROGRAM=/WebApps/Library/';
put '<LI><A HREF="' hrefroot +(-1)
'Aisles&type=Book">Book Aisle</A></LI>';
put '<LI><A HREF="' hrefroot +(-1)
'Aisles&type=Video">Video Aisle</A></LI>';
put '<LI><A HREF="' hrefroot +(-1)
'Aisles&type=Audio">Audio Aisle</A></LI>';
put '<LI><A HREF="' hrefroot +(-1)
'Aisles&type=Hardware">Hardware Aisle</A></LI>';
put '<LI><A HREF="' hrefroot +(-1)
'Shopping Cart">View my shopping cart</A></LI>';
put '<LI><A HREF="' hrefroot +(-1)
'Logout">Logout</A></LI>';
put '</UL>';
put '</BODY>';
put '</HTML>';
run;
```

The main aisle page consists of a list of links to specific sections of the Online Library.
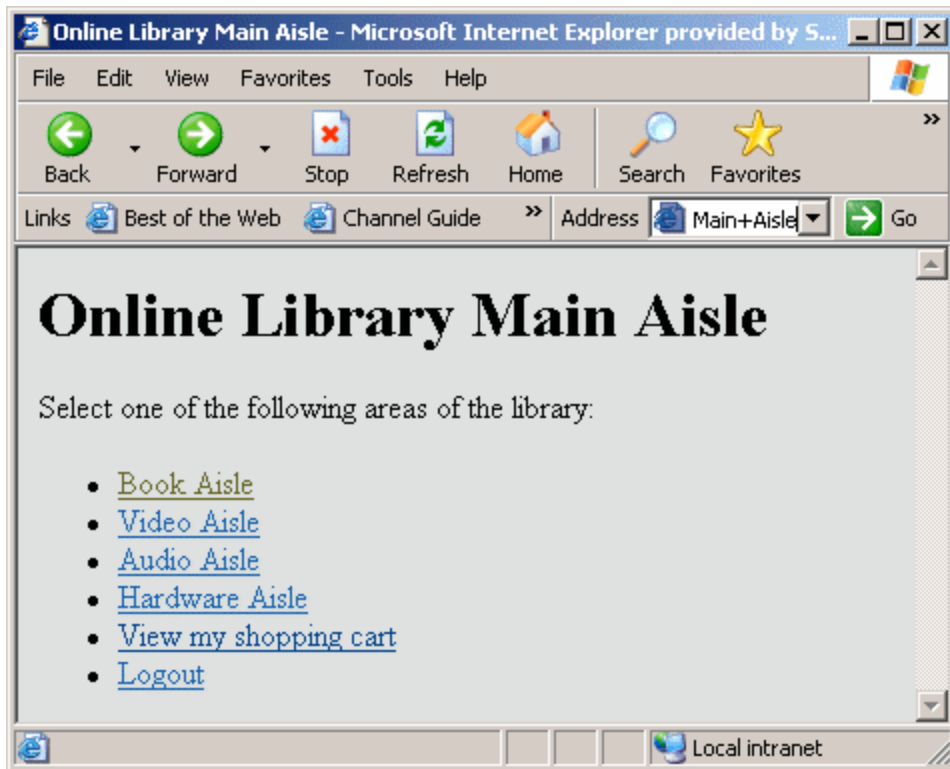


Each link in this page is built using the _THISSESSION macro variable. This variable includes both the _URL value pointing back to the SAS Stored Process Web Application and the _SESSIONID value that identifies the session.

# Aisles Stored Process

The library is divided into aisles for different categories of library items. The pages for each aisle are generated by one shared Aisles stored process. The stored process accepts a TYPE input variable that determines which items to display.

```
/* Aisles - List items in a specified aisle.
The aisle is specified by the TYPE variable. */
libname SAMPDAT 'C:\My Demos\Library';

/* Build a temporary data set that contains the
selected type, and add links for selecting
and adding items to the shopping cart. */
data templist;
if libref('SAVE') ne 0 then
rc = stpsrv_session('create');
set SAMPDAT.LIB_INVENTORY;
where type="%UPCASE(&type)";
length select $200;
select = '<A HREF="' || symget("_THISSESSION") ||
'&_program=/WebApps/Library/Add+Item&REFNO=' ||
trim(left(refno)) || '&TYPE=' || "&TYPE" ||
'">Add to cart</A>';
run;
ods html body=_webout(nobot) rs=none;
title Welcome to the &type Aisle;
proc print data=templist noobs label;
var refno desc select;
label refno='RefNo' desc='Description' select='Select';
run;
ods html close;
data _null_;
file _webout;
put '<P>';
put 'Return to <A HREF="' "&_THISSESSION"
'&_PROGRAM=/WebApps/Library/Main+Aisle'
'">main aisle</A><BR>';
put 'View my <A HREF="' "&_THISSESSION"
'&_PROGRAM=/WebApps/Library/Shopping+Cart'
'">shopping cart</A><BR>';
put '</BODY>';
put '</HTML>';
run;
```

The stored process selects a subset of the LIB_INVENTORY data set by using a WHERE clause, and then uses PROC PRINT to create an HTML table. A temporary data set is created. This data set contains the selected items that users can use to add items. An additional column is generated from the LIB_INVENTORY data set that has an HTML link that users can use to add the item to their shopping cart.

In this stored process, both ODS and a DATA step are used to generate HTML. The ODS HTML statement includes the NOBOT option that indicates that more HTML is appended after the ODS HTML CLOSE statement. The navigation links are then

added using a DATA step. The following display shows the contents of the Book Aisle.
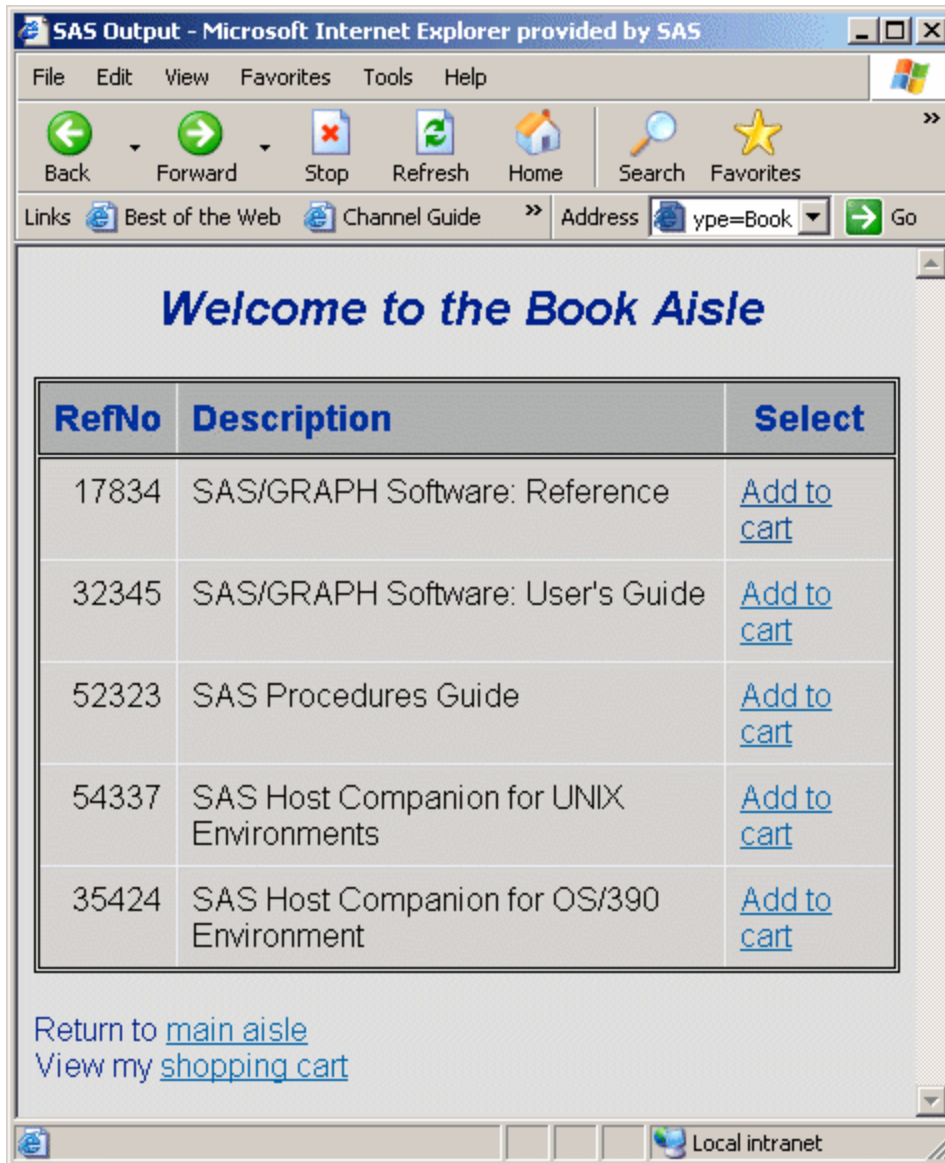


# Add Item Stored Process

The Add Item stored process is run when the user clicks the **Add to cart** link in the aisle item table. The specified item is copied from the LIB_INVENTORY data set to a shopping cart data set in the session library (SAVE.CART). The session and the data set remain accessible to all programs in the same session until the session is deleted or it times out.

```
/* Add Item - Add a selected item to the shopping cart.
This stored process uses REFNO and TYPE input
variables to identify the item. */
libname SAMPDAT 'C:\My Demos\Library';
```

```
/* Perform REFNO and TYPE verification here. */
/* Append the selected item. */
proc append base=SAVE.CART data=SAMPDAT.LIB_INVENTORY;
where refno=&refno;
run;
/* Print the page. */
data _null_;
file _webout;
put '<HTML>';
put '<HEAD><TITLE>Selected Item Added to
Shopping Cart</TITLE></HEAD>';
put '<BODY vlink="#004488" link="#0066AA"
bgcolor="#E0E0E0">';
put "<H1>Item &refno Added</H1>";
put 'Return to <A HREF="' "&_THISSESSION"
'&_PROGRAM=/WebApps/Library/Aisles'
'&TYPE=' "&TYPE" '">' "&TYPE aisle</A><BR>";
put 'Return to <A HREF="' "&_THISSESSION"
'&_PROGRAM=/WebApps/Library/Main+Aisle'
'">main aisle</A><BR>';
put 'View my <A HREF="' "&_THISSESSION"
'&_PROGRAM=/WebApps/Library/Shopping+Cart'
'">shopping cart</A><BR>';
put '</BODY>';
put '</HTML>';
run;
```

The program prints an information page that has navigation links.
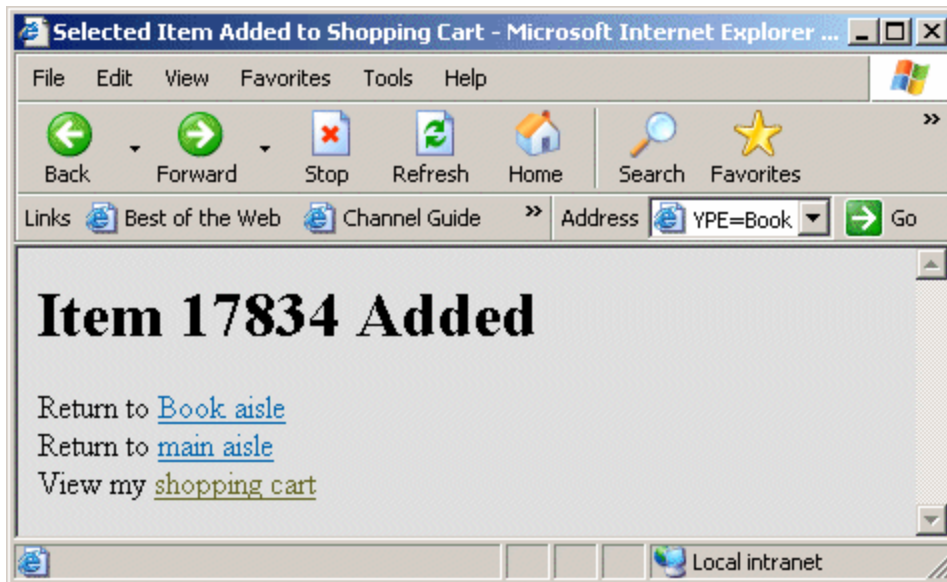


# Shopping Cart Stored Process

The Shopping Cart stored process displays the contents of the shopping cart.

```
/* Shopping Cart - Display contents of the shopping cart
```

```
* (SAVE.CART data set). */
%macro lib_cart;
%let CART=%sysfunc(exist(SAVE.CART));
%if &CART %then %do;
/* This program could use the same technique as the
LIB_AISLE program in order to add a link to each
line of the table that removes items from the
shopping cart. */
/* Print the CART contents. */
ods html body=_webout(nobot) rs=none;
title Your Selected Items;
proc print data=SAVE.CART noobs label;
var refno desc;
label refno='RefNo' desc='Description';
run;
ods html close;
%end;
%else %do;
/* No items in the cart. */
data _null_;
file _webout;
put '<HTML>';
put '<HEAD><TITLE>No items
selected</TITLE></HEAD>';
put '<BODY vlink="#004488" link="#0066AA"
bgcolor="#E0E0E0">';
put '<H1>No Items Selected</H1>';
put;
run;
%end;
/* Print navigation links. */
data _null_;
file _webout;
put '<P>';
if &CART then do;
put '<FORM ACTION="' "&_url" '">';
put '<INPUT TYPE="HIDDEN" NAME="_program"
VALUE="/WebApps/Library/Logout">';
put '<INPUT TYPE="HIDDEN" NAME="_sessionid"
VALUE="' "&_sessionid" '">';
put '<INPUT TYPE="HIDDEN" NAME="CHECKOUT"
VALUE="YES">';
put '<INPUT TYPE="SUBMIT"
VALUE="Request these items">';
put '</FORM><P>';
end;
put 'Return to <A HREF="' "&_THISSESSION"
'&_PROGRAM=/WebApps/Library/Main+Aisle'
'">main aisle</A><BR>';
put '<A HREF="' "&_THISSESSION"
'&_PROGRAM=/WebApps/Library/Logout'
'&CHECKOUT=NO">Logout</A><BR>';
put '</BODY>';
put '</HTML>';
run;
%mend;
```

```
%lib_cart;
```

The contents of the shopping cart are displayed using a PROC PRINT statement. The page also includes a request button and navigation links. The request button is part of an HTML form. In order to connect to the same session, include the _SESSIONID value in addition to the _PROGRAM value. These values are usually specified as hidden fields. This program also has a hidden CHECKOUT field that is initialized to YES in order to indicate that the user is requesting the items in the cart.

The program prints a page that contains the contents of the shopping cart.
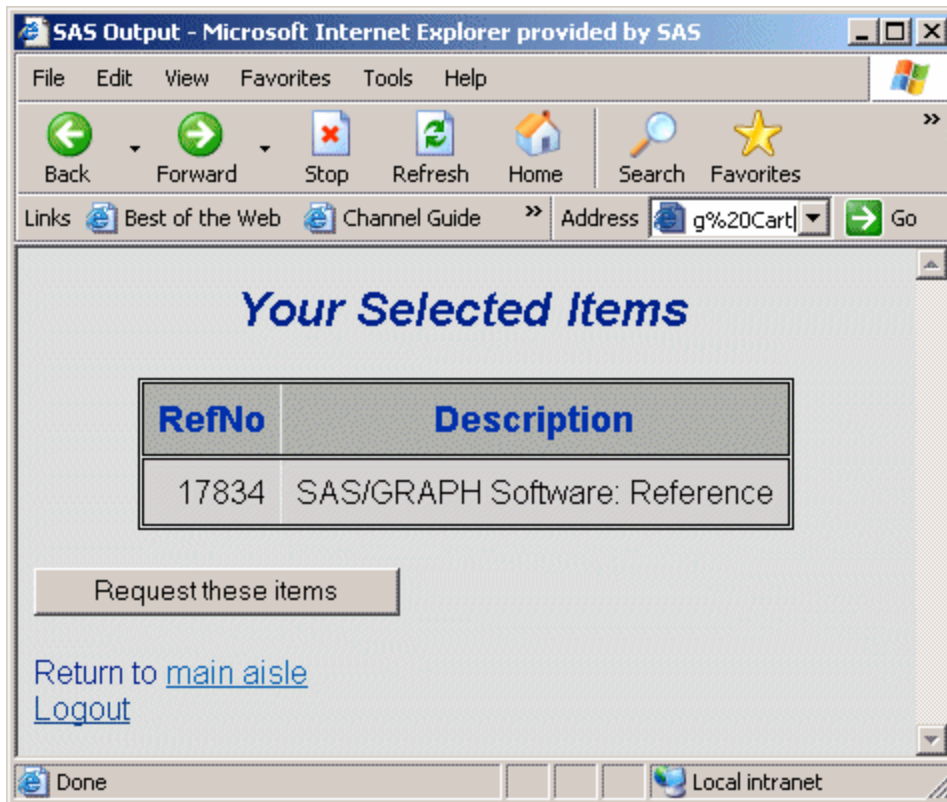


## Logout Stored Process

The Logout stored process checks the user out of the Online Library. If the CHECKOUT input variable is YES, then all of the items in the user's shopping cart are requested through email.

```
/* Logout - logout of Online Library application.
Send email to the library@abc.com account with
requested item if CHECKOUT=YES is specified. */
%macro lib_logout;
%global CHECKOUT;
    /* Define CHECKOUT in case it was not input. */
%if %UPCASE(&CHECKOUT) eq YES %then %do;
/* Checkout - send an email request to the library.
See the documentation for the email access method
on your platform for more information on the
required options. */
```

```
/* ***************** disabled for demo *************
filename RQST EMAIL 'library@abc.com'
SUBJECT='Online Library Request for &_username';
ods listing body=RQST;
title Request for &_username;
proc print data=SAVE.CART label;
var refno type desc;
label refno='RefNo' type='Type'
desc='Description';
run;
ods listing close;
* ********************************************** */
data _null_;
file _webout;
put '<HTML>';
put '<HEAD><TITLE>Library
Checkout</TITLE></HEAD>';
put '<BODY vlink="#004488" link="#0066AA"
bgcolor="#E0E0E0">';
put '<H1>Library Checkout</H1>';
put;
put 'The items in your shopping cart have
been requested.';
put '<P>Requested items will normally
arrive via interoffice';
put 'mail by the following day. Thank you
for using the Online Library.';
put '<P><A HREF="' "&_URL"
'?_PROGRAM=/WebApps/Library/Main+Aisle"
>Click here</A>';
put 'to re-enter the application.';
put '</BODY>';
put '</HTML>';
run;
%end;
%else %do;
/* Logout without requesting anything. */
data _null_;
file _webout;
put '<HTML>';
put '<HEAD><TITLE>Logout</TITLE></HEAD>';
put '<BODY vlink="#004488" link="#0066AA"
bgcolor="#E0E0E0">';
put '<H1>Library Logout</H1>';
put;
put '<P>Thank you for using the Online Library.';
put '<P><A HREF="' "&_URL"
'?_PROGRAM=/WebApps/Library/Main+Aisle"
>Click here</A>';
put 'to re-enter the application.';
put '</BODY>';
put '</HTML>';
run;
%end;
%mend;
%lib_logout;
```

```
/* User is finished - delete the session. */
%let rc=%sysfunc(stpsrv_session(delete));
```

An information page is displayed if the user chooses to request the shopping cart items.
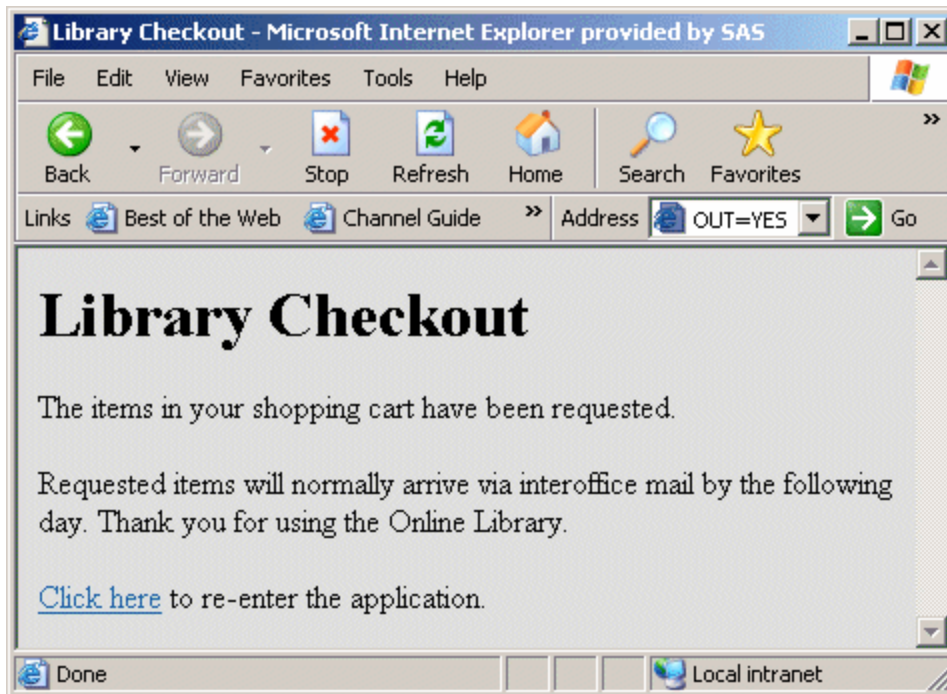


A logoff screen is displayed if the user selects the Logout link.



**Note:** Logging off is not required. All sessions have an associated time-out (the default is 15 minutes). If the session is not accessed for the duration of the time-out, then the session and all temporary data in the session are deleted. In this sample, the SAVE.CART data set is automatically deleted when the session time-out is reached. You can change the session time-out by using the STPSRVSET('session timeout',*seconds*) function inside the program.

# Error Handling

You can write custom JSPs to handle a set of common errors. For more information about the default error JSPs for the SAS Stored Process Web Application, see "Custom Responses" on page 85.

If an error occurs while the stored process is running, then you get an error message with a button that you can click to show the SAS log.
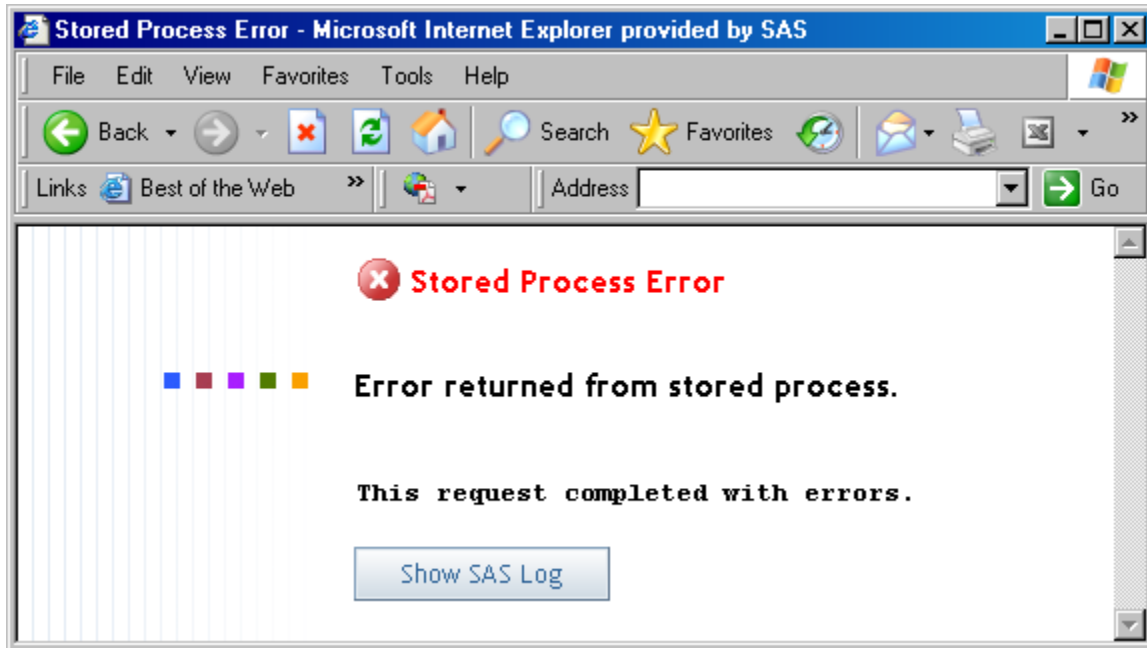


In order to disable the **Show SAS Log** button, you can set the web application initialization parameter ShowLogButton to `false` or set the DebugMask initialization parameter to completely disable debug logging. For more information, see "Debugging in the SAS Stored Process Web Application" on page 145.

If an error is detected before the output stream back to the web browser is opened, then the HTTP header line `X-SAS-STP-ERROR` is returned. This can be used by programs that make URL connections to the SAS Stored Process Web Application.

# Debugging in the SAS Stored Process Web Application

## Testing the SAS Stored Process Web Application

After the SAS Stored Process Web Application has been installed, it can be tested by invoking it directly from a web browser. To execute the SAS Stored Process Web Application, enter the web application URL in the web browser. Either the default Welcome page or static version data is returned.

For example, if you enter the following SAS Stored Process Web Application URL:

```
http://yourserver.com:8080/SASStoredProcess/do
```

then you either get the data from the Welcome.jsp page if it has been installed, or a display similar to the following display:

```
Stored Process Web Application

Version 9.4 (Build 499)
```

If an error is returned, then the installation was not completed successfully or the URL that was entered is incorrect.

The reserved macro variable _DEBUG provides you with several diagnostic options. Using this variable is a convenient way to debug a problem, because you can supply the debug values by using the web browser to modify your HTML or by editing the URL in your web browser's location field. For example, to see the installed environment for the SAS Stored Process Web Application, the URL can be entered with _DEBUG=ENV appended. A table is returned, which contains the system values for the SAS Stored Process Web Application environment.

## List of Valid Debugging Keywords

You can activate the various debugging options by passing the _DEBUG variable to the SAS Stored Process Web Application. Keywords are used to set the debugging options. Multiple keywords can be specified, separated by commas or spaces. Here is an example:

```
_DEBUG=TIME,TRACE
```

Some debugging flags might be locked out at your site for security reasons. Verify with your administrator which flags are locked out at your site. The following chart is a list of valid debugging keywords:

*Table 7.10*   *Debugging Keywords*

| Keyword | Description |
| --- | --- |
| FIELDS | Displays the stored process input parameters. |
| TIME | Returns the processing time for the stored process. |
| DUMP | Displays output in hexadecimal format. |
| LOG | Returns the SAS log file. This log is useful for diagnosing problems in the SAS code. |
| TRACE | Traces execution of the stored process. This option is helpful for diagnosing the SAS Stored Process Web Application communication process. You can also use this option to see the HTTP headers that the server returns. |
| LIST | Lists known stored processes. |
| ENV | Displays the SAS Stored Process Web Application environment parameters. |

# Setting the Default Value of _DEBUG

Web application initialization parameters can be used to set default values for the _DEBUG parameter or to limit the allowable values. Any of the valid keywords that are listed in the preceding table can be specified as a comma-separated list for the `Debug` initialization parameter. These values become the default debugging values for each request. The initialization parameter `DebugMask` can be used to specify a list of _DEBUG values that are valid. For more information about initialization parameters, see .

# Enabling Logging

To enable logging for stored processes that are executed by the SAS Stored Process Web Application, perform the following steps:

**1**   Open the `\Lev1\Web\Common\LogConfig\SASStoredProcess-log4j.xml` file.

**2**   Change the value of priority to "INFO".

```
<category additivity="false"
    name="com.sas.services.storedprocess.webapp">
    <priority value="INFO"/>
    <appender-ref ref="SAS_CONSOLE"/>
    <appender-ref ref="SAS_FILE"/>
</category>
```

# Character Encoding

The SAS Stored Process Web Application is configured to accept parameters that have been UTF-8 encoded. If any URL links are created to access the web application, the parameters in the URL should be encoded. If links are created in the form http://*yourserver*.com:8080/SASStoredProcess/do?_program=/User1/myprogram&x1=ääkkö, then an error is returned. If the following error is returned when you access the web application, verify that all parameters have been encoded:

```
STP: Parameter 'xxxxxx' contains characters that
cannot be processed by the SAS server (wlatin1 encoding).
```

You can determine the encoding that the SAS Stored Process Web Application is using by submitting the following request from your web browser: http://*yourserver*.com:8080/SASStoredProcess/do?_debug=env The servlet.character.encoding value indicates the encoding that the SAS Stored Process Web Application is using to send parameters to the SAS Stored Process Server. The servlet encoding is set for the SAS Stored Process Web Application in the **web.xml** file CharacterEncodingFilter.

Perform the following steps to make sure that all parameter values are properly encoded:

1 Submit the request using the _DEBUG=TRACE,LOG parameter in the URL for the SAS Stored Process Web Application request so that the parameter values that are being passed are displayed, as shown in this example: http://*yourserver*.com:8080/SASStoredProcess/do?_debug=trace,log.

2 Check the list of input parameters for a value that contains a character that is not displayed properly. For example, you might see a small square box instead of a printable character.

3 Modify your input parameters to remove any invalid characters or change the values to UTF-8 encoding. The SAS Stored Process Web Application uses UTF-8 encoding, so values should also be encoded in UTF-8. If the HTML pages are being created by a SAS program, the URLENCODE function can be used to create the links. This function can also be run manually to get the encoded string for any value. For example, the following code returns this string: **%C3%A4%C3%A4kk%C3%B6**.

```
options urlencoding="UTF8";

data _null_;
    x1= urlencode ('ääkkö');
    put x1;
run;
```

Place the UTF-8 encoded parameters in the web application URL as follows: http://*yourserver*.com:8080/SASStoredProcess/do?_program=/User1/myprogram&x1=%C3%A4%C3%A4kk%C3%B6.

In addition, you can specify the UTF-8 encoding in a FILE_WEBOUT statement, as shown in this example:

```
data _null_;
   file _webout encoding='utf-8';
   x1= urlencode ('ääkkö');
   put x1;
run;
```

# Chapter 8

# STP Procedure

# Overview: STP Procedure

## What Does the STP Procedure Do?

The STP procedure enables stored process execution from a SAS program. PROC STP can be executed in an interactive, batch, or server SAS session and can also be executed by another stored process. PROC STP can run the stored process locally or remotely, based on the SERVER option and the stored process metadata.

If the stored process executes remotely (which is the default), then PROC STP executes stored processes on the CONNECT or Grid server that is specified for the SAS Application Server in metadata. If the stored process executes locally, then it

runs within an independent execution environment with its own WORK library and macro symbol table. Data, files, and macro values can be passed between the SAS environment and the stored process environment by using optional PROC STP statements.

**Note:**

- PROC STP must have access to the stored process metadata in order to locate the stored process source code. PROC STP also checks input parameters against the input parameters that are defined in the metadata. In order to specify the metadata server and connection profile to use with PROC STP, use the following system options: METASERVER, METAPORT, METAUSER, and METAPASS. Define these options before running PROC STP.

- PROC STP cannot display prompts at run time. When you execute stored processes using PROC STP, you can use the INPUTPARAM statement to provide prompt values.

# Concepts: STP Procedure

## Requirements for Specifying User Formats with PROC STP

Starting with SAS 9.4, you can specify user formats with PROC STP. In order to use the FMTSEARCH system option to search format catalogs, you must meet the following requirements:

- Any format catalog to be accessed in the stored process must have an INPUTDATA specification, for example:

```
INPUTDATA fmtcat=fmtlib.formats / MT=CATALOG;
```

The '*input-data-file*' argument (in this case, `fmtcat`) does not need to be referenced in the stored process. You can define multiple format catalogs to pass into the stored process environment. There is no restriction on the library in which the formats reside.

- The special input parameter '`_FMTSEARCH`' must be specified. This parameter specifies which format catalogs need to be appended to the FMTSEARCH option in the stored process environment. The value of this parameter should be a blank-delimited list of the INPUTDATA arguments that are specified for the desired catalogs.

```
INPUTPARAM  _FMTSEARCH='fmtcat< fmtcat2  …>';
```

For multiple catalogs, the order in the string is the order the catalogs will appear in the stored process FMTSEARCH.

**Note:** The current FORMATSEARCH system option value is not automatically passed into stored process.

The following example shows how PROC STP can use user formats:

```
filename _webout '…output file…';

proc format library=work.myfmts;
value $gender 'M'='Guy'
               'F'='Gal';
run;
options fmtsearch=(work.myfmts);

* Build CLASS data set with user format here;
libname datalib '…your library…';

data datalib.class;
     format sex $gender.;
     set sashelp.class;
run;


proc stp program='/Users/johndoe/procstp/format';
  inputdata mydata = datalib.class;
  inputdata myfmt = work.myfmts / mt=catalog;
  outputfile _webout;
  inputparam _fmtsearch='myfmt';
run;
```

# Automatic Macro Variables Generated by PROC STP

PROC STP typically sets its exit code to either 0 or 1012 (error). However, if the grammar parser generates a warning, then the exit code might be 4. The SYSCC macro variable is updated with this value if it is greater than the current setting. This variable indicates the maximum exit code that was returned in the SAS stream, unless it has been reset by the user.

The exit code for the procedure does not directly correspond to the exit code (that is, SYSCC) from the stored process environment. The stored process is not executing in the same SAS stream environment as the procedure. If the stored process returns anything greater than a warning (4), then the exit code for the procedure is an error (1012). If the stored process returns 4 or less, then the exit code for the procedure is 0. These numbers reflect whether the stored process ran with an error (1012) or without an error (0). The actual stored process exit code is output to the SAS log at the end of the procedure.

Starting with SAS 9.4M2, the procedure provides the following two macro variables to the user when exiting to communicate status from the stored process. You can incorporate these macro variables in your code if you want the SAS stream to behave differently based on what happened in the stored process.

_STPCC

specifies the maximum exit code from the stored process environment, taken from the SYSCC macro value in that environment. The SYSCC value can be set by any SAS component. See *SAS Macro Language: Reference* for more information about the SYSCC automatic macro variable.

For remote execution, the SYSCC value has to be retrieved before the remote SAS session terminates. If the remote session terminates unexpectedly before the procedure can retrieve the SYSCC value, then _STPCC is set to 1012 (error).

_STPRC

specifies the current return code from the stored process environment. This number is chosen by the component that is setting it.

Note: For remote execution, _STPRC cannot be set reliably and is always 0.

The _STPCC and _STPRC macro variables can also be set by the ABORT and %ABORT statements. These two statements have identical options and generate identical exit codes. The specific values that they generate depend on the options that are specified in the statement.

_STPCC

The following table shows the _STPCC values that are generated by specific ABORT or %ABORT statements. These values are always applicable for locally executed stored processes, but are limited for remotely executed stored processes. These statements are used to stop the current SAS step, and sometimes to immediately terminate the SAS session. Thus, if the remote session is terminated, the _STPCC can only be set to 1012 (error).

| Generated Value | Statement |
| --- | --- |
| 20,000 | (%)ABORT ; |
| 5 | (%)ABORT CANCEL ; |
| 6 | (%)ABORT CANCEL FILE ; |
| 20,001 | (%)ABORT RETURN ; |
| 20,002 | (%)ABORT ABEND ; |

_STPRC

This value can be specified in an optional *n* argument for the ABORT or %ABORT statement. If the *n* argument is not specified, then _STPRC is set to 0.

Note: For remote execution, _STPRC cannot be set reliably and is always 0.

See *SAS Macro Language: Reference* for more information about the %ABORT statement. See *SAS DATA Step Statements: Reference* for more information about the ABORT statement.

# Syntax: STP Procedure

**PROC STP** PROGRAM=*'metadata-path-of-stored-process'* <ODSOUT=STORE | REPLAY>
<SERVER=LOCAL | "*server-name*">;

    **INPUTDATA** *input-data-source(s)* </ <REMOTE> <MEMTYPE=DATA | VIEW | CATALOG>>;

    **INPUTFILE** *input-file(s)* </ REMOTE >;

    **INPUTPARAM** *parameter-name-1<="parameter-value-1">*

        *<parameter-name-2<="parameter-value-2"> …>*;

    **LIST**< GROUP=*level* | (*level1 level2* ...)>;

    **LOG** FILE=*local-fileref* | *local-file-path*;

    **OUTPUTDATA** *output-data-target(s)* </ <REMOTE> <MEMTYPE=DATA | CATALOG>>;

    **OUTPUTFILE** *output-file(s)* </ REMOTE >;

    **OUTPUTPARAM** *parameter-name-1<="local-variable-name-1">*
    *<parameter-name-2<="local-variable-name-2">…>*;;

| Statement | Task |
|---|---|
| PROC STP | Executes a stored process |
| INPUTDATA | Defines input data files for the execution of the stored process |
| INPUTFILE | Defines input files for the execution of the stored process |
| INPUTPARAM | Defines input parameters for the execution of the stored process |
| LIST | Lists details about the stored process that is specified by the PROGRAM= option |
| LOG | Controls the location of the stored process log |
| OUTPUTDATA | Defines output data files that are produced by the execution of the stored process |
| OUTPUTFILE | Defines output files that are produced by the execution of the stored process |
| OUTPUTPARAM | Defines output parameters that are produced by the execution of the stored process |

# PROC STP Statement

Executes a stored process.

## Syntax

**PROC STP** PROGRAM=*'metadata-path-of-stored-process'* <ODSOUT=STORE | REPLAY>
<SERVER=LOCAL | *server-name*>;

## Required Argument

**PROGRAM=*'metadata-path-of-stored-process'***
    specifies the metadata location of the stored process to be executed.

## Optional Arguments

**ODSOUT=STORE | REPLAY**
    specifies a destination for ODS output from the stored process. If you specify STORE, then the ODS output is stored so that it can be used by subsequent SAS statements. If you specify REPLAY, then the ODS output is displayed in the Output window of the SAS windowing environment.

    **Note:** This option is supported only when the local machine and the remote server machine have compatible architectures. The machines must be using the same type of operating system and have the same data type attributes.

**SERVER=LOCAL | "*server-name*"**
    specifies whether the stored process executes locally or on a specified remote application server. This option is useful if you need the stored process to execute on a server other than the default server. Remember that stored processes are associated with an application server context, which defines the environment in which the stored process executes. Application server contexts typically contain multiple server definitions. PROC STP executes stored processes on a Grid or CONNECT server that is defined for the specified application server context.

    If the SERVER option is not specified, then by default the stored process is executed remotely on a Grid or CONNECT server that is defined for the SAS Application Server that was specified in the stored process execution metadata. If that application server is not available, then the stored process executes locally.

    If SERVER=LOCAL is specified, then the stored process is forced to run locally. Stored processes that are compatible with SAS 9.2 are always run locally.

    If SERVER="*server-name*" is specified, then for "*server-name*" you must specify the name of a SAS Application Server that has a Grid or CONNECT server

defined. The server that is specified here takes precedence over server settings in metadata.

The execution log displays information about which server was used to execute the stored process.

# Examples

## Example 1

The following example executes the Hello World stored process and generates an HTML file that is saved in a local directory:

```
proc stp program='/Products/SAS Intelligence Platform/Samples/
      Sample: Hello World';
   outputfile _webout="./hello.html";
   run;
```

## Example 2

The following example executes the Multiple Output Formats stored process and displays the output:

```
proc stp program='/Products/SAS Intelligence Platform/Samples/
      Sample: Multiple Output Formats' odsout=replay;
   run;
```

## Example 3

The following example executes the Multiple Output Formats stored process and saves the output:

```
proc stp program='/Products/SAS Intelligence Platform/Samples/
      Sample: Multiple Output Formats' odsout=store;
   run;
```

## Example 4

The following example executes a stored process that produces a graph and table, and then replays the output:

```
proc stp program='/User Folders/johndoe/odstest' odsout=store;
run;

ods html file="./output.html";
```

```
proc document name=&_ODSDOC (read);
   replay / levels=all;
run;
quit;
ods html close;
```

The following code shows the odstest.sas stored process:

```
%stpbegin;
   proc gchart data=sashelp.class;
      vbar age / discrete;
   run; quit;


   proc print data=sashelp.class;
   run; quit;
%stpend;
```

# Example 5

The following example executes two stored processes and replays both documents:

```
proc stp program='/User Folders/johndoe/procprnt' odsout=store;
run;
%let printdoc=&_ODSDOC;

proc stp program='/User Folders/johndoe/gchart' odsout=store;
run;
%let graphdoc=&_ODSDOC;

ods html file="./output.html";
proc document name=&graphdoc (read);
   replay / levels=all;
run;quit;

proc document name=&printdoc (read);
   replay / levels=all;
run;quit;
ods html close;
```

The following code shows the gchart.sas stored process:

```
%stpbegin;
   proc gchart data=sashelp.class;
      vbar age / discrete;
   run; quit;
%stpend;
```

The following code shows the procprint.sas stored process:

```
%stpbegin;
proc print data=sashelp.class;
   run; quit;
%stpend;
```

# INPUTDATA Statement

Defines input data files for the execution of the stored process.

## Syntax

**INPUTDATA** *input-data-source(s)* </ <REMOTE> <MEMTYPE=DATA | VIEW | CATALOG>>;

## Required Argument

***input-data-source(s)***
    can have the following form:

    `stored-process-data-file<=member-name | 'data-set-path'>`

| | |
|---|---|
| *stored-process-data-file* | specifies the name of an input data file. This name corresponds to an input parameter (macro variable) that is visible to the stored process program. |
| *member-name* | specifies a one- or two-level name of a SAS data set, view, or catalog that the stored process can access directly. If the stored process outputs data to an existing catalog, the catalog must also be specified as an input data source to avoid overwriting an entire catalog with what might be only a few entries created by the stored process. |
| '*data-set-path*' | provides alternate syntax for specifying a data set. |

## Optional Arguments

**REMOTE**
    specifies that the input data source exists on the remote server. This option is not supported for local execution.

**MEMTYPE=DATA | VIEW | CATALOG**
    specifies the member type of the input data source. The value that you specify for this option must match the type of data set that you are accessing. If you do not specify this option, then either a data set or a data view is used as the input data source. Valid values are:

| | |
|---|---|
| DATA | specifies that the input data source is a data set. |
| VIEW | specifies that the input data source is a data view. |
| CATALOG | specifies that the input data source is a catalog. If the stored process accesses a catalog, then the MEMTYPE=CATALOG option is required. |

**Alias** MT

---

# Details

The INPUTDATA statement passes a data file that is known within the PROC STP environment to the stored process environment. Multiple INPUTDATA statements can be specified in a single procedure call. PROC STP prepends the INPUTDATA label with _SOURCE_, in order to be compliant with how data sources are handled in SAS Management Console. If a data set is specified as input for a stored process and subsequently updated by the stored process, then you must also specify that data set as an output data set, using the OUTPUTDATA statement. This ensures that any updates that the stored process makes to the data set are saved in the data set.

---

# Examples

---

# Example 1

The following SAS code supports the print stored process, which is used in the following three examples. Notice that in all three of these examples, PROC STP modifies data1 to be _SOURCE_data1. Within this stored process, PROC PRINT uses the _SOURCE_data1 reference that is passed in by PROC STP:

```
%stpbegin;
proc print data=&_SOURCE_data1
run;
%stpend;
```

The following example passes the data set via a data set path in the local file system:

```
proc stp program='/User Folders/johndoe/print' odsout=replay;
   inputdata data1='C:\temp\employees.sas7bdat';
run;
```

---

# Example 2

The following example passes the data set via a two-level name:

```
libname dataemps 'C:\temp';

proc stp program='/User Folders/johndoe/print' odsout=replay;
   inputdata data1=dataemps.employees;
run;
```

## Example 3

The following example passes the data set via a one-level name. The data set resides in the WORK library.

```
proc stp program='/User Folders/johndoe/print' odsout=replay;
   inputdata data1=emps;
run;
```

## Example 4

The following example uses a catalog as the input data file.

```
proc stp program='/User Folders/johndoe/catalog_contents' odsout=replay;
   inputdata catalog = sashelp.base / mt=catalog;
run;quit;
```

The following SAS code supports the catalog_contents stored process:

```
%stpbegin;
proc catalog catalog=&_SOURCE_catalog;
   contents;
run; quit;
%stpend;
```

# INPUTFILE Statement

Defines input files for the execution of the stored process.

## Syntax

**INPUTFILE** *input-file(s)* </ REMOTE >;

## Required Argument

***input-file(s)***
  can have the following form:

  *stored-process-file<=fileref | 'file-path' | fileref('file')>*

| | |
|---|---|
| *stored-process-file* | specifies the name of an input file. This name corresponds to a fileref that is visible to the stored process program. The short form of the INPUTFILE statement, INPUTFILE *stored-process-file*;, is equivalent to INPUTFILE *stored-process-file=stored-process-file*;. |
| *fileref* | specifies the name of a fileref. |
| '*file-path*' | specifies the path of a file. |

*fileref*('*file*')    specifies a fileref of an aggregate storage location and the name of a file or member, enclosed in parentheses, that resides in that location. You must have previously associated the fileref with a file in a FILENAME statement, a FILENAME function, or an appropriate operating environment command.

## Optional Argument

**REMOTE**

specifies that the input file exists on the remote server. The *stored-process-file* fileref must already exist in the remote server session. This option is required if the input file does not exist locally. This option is not supported for local execution.

## Details

Multiple INPUTFILE statements can be specified in a single procedure call. If a file is specified as input for a stored process and subsequently updated by the stored process, then you must also specify that file as an output file, using the OUTPUTFILE statement. This ensures that any updates that the stored process makes to the file are saved in the file.

## Examples

## Example 1

The following example executes a stored process with three input files, demonstrating different use cases for the INPUTFILE statement:

```
proc stp program='/Products/SAS Intelligence Platform/Samples/
    Example Stored Process';
  inputfile file1;                    /* makes the data from local file1 fileref
                                         available to the stored process file1
                                         fileref */
  inputfile file2=myfile;             /* makes the data from local myfile fileref
                                         available to the stored process file2
                                         fileref */
  inputfile file3="/tmp/data3.csv";   /* makes the /tmp/data3.csv file available
                                         to stored process file3 fileref */
  ...
  run;
```

# Example 2

The following example passes a fileref by its local name:

```
filename incsv './class.csv';

proc stp program='/User Folders/johndoe/readCSV';
   inputfile incsv;
run;
```

The stored process readCSV contains the following code:

```
data work.mydata;
   infile incsv dsd;
   input name $ sex $ age height weight ;
run;
```

# Example 3

You can also pass an input file to a stored process via another name or by using a file system path. In the following two examples, the stored process accesses the fileref by the name incsv:

```
filename localfref 'c:\temp\class.csv';

proc stp program='/User Folders/johndoe/readCSV';
   inputfile incsv=localfref;
run;
```

# Example 4

```
proc stp program='/User Folders/johndoe/readCSV';
   inputfile incsv='c:\temp\class.csv';
run;
```

# Example 5

The following example passes an input file to a stored process by using aggregate syntax to specify the filename and location.

```
filename localfref 'c:\temp';

proc stp program='/User Folders/johndoe/readCSV';
   inputfile incsv = localfref('class.csv');
run;
```

# INPUTPARAM Statement

Defines input parameters for the execution of the stored process.

## Syntax

**INPUTPARAM** *parameter-name-1<="parameter-value-1">*
   *<parameter-name-2<="parameter-value-2"> …>*;

## Required Argument

***parameter-name<="parameter-value">***
   specifies input parameter names, or specifies input parameters as name/value
   pairs where the values are quoted strings. Each name or name/value pair is
   converted to a SAS macro, which can be used in the stored process. The
   parameter name is the name of the macro variable that contains the parameter
   value. The parameter name must be a valid macro variable name and cannot be
   longer than 32 characters in length. You can specify a parameter value, as
   follows:

   | | |
   |---|---|
   | *"parameter-value"* | specifies the value for the input parameter. The parameter value can be any string up to 32767 characters in length. Starting with SAS 9.4M3, if you specify a null value (for example, `inputparam text1="";`), then any default value that was specified for the parameter in metadata is ignored and the value is null. However, if the metadata specifies that the parameter requires a non-blank value, then you will receive an error. |

   Starting with SAS 9.4M3, you can specify multiple lines of text for the value of
   the input parameter. To do this, specify the value in the following format, rather
   than using "*parameter-value*":

   ```
   ('parameter-value-line-1' 'parameter-value-line-2'...)
   ```

   **Note:** If an input parameter is defined in metadata, then the INPUTPARAM
   values must reconcile with any limitations that have been specified for them in
   metadata.

## Details

PROC STP cannot display prompts at run time. When you execute stored
processes using PROC STP, you can use the INPUTPARAM statement to provide
prompt values.

# Examples

## Example 1

The following example executes a stored process with various input parameters:

```
proc stp program='/Products/SAS Intelligence Platform/Samples/
     Example Stored Process';
   inputparam _odsstyle="Blue" _odsdest="PDF" year="2010";
   ...
   run;
```

## Example 2

The following example shows an INPUTPARAM statement without a specified value. The value of the local variable with the same name is used.

```
%let testval=23;
```

```
proc stp program='/User Folders/johndoe/inparam';
   inputparam  testval;
run;
```

The stored process receives TESTVAL=23. It is the same as if the INPUTPARAM statement was the following:

```
Inputparam testval="&testval";
```

## Example 3

An input parameter can also be a range value (for example, a text range, numeric range, date range, time range, or timestamp range). A range value is used to specify lower and upper boundaries for a valid value. A valid range therefore always has two values. A range can be specified with two separate values in the INPUTPARAM statement, where the lower boundary is specified first.

```
proc stp program='/User Folders/johndoe/inparam';
   inputparam dtrange='February 6, 2014'  dtrange='February 9, 2014';
run;
```

When the stored process executes, the lower and upper values are used to limit the data that is processed. You can also use one parameter with two values that are separated by — (two hyphens) as another way to specify the same range.

```
proc stp program='/User Folders/johndoe/inparam';
   inputparam dtrange='February 6, 2014  -- February 9, 2014';
run;
```

## Example 4

In the following example, the input parameter x is defined in metadata as a numeric value that must be within the range of 5-10.

```
proc stp program='/User Folders/johndoe/inparam';
   inputparam x='4';  /* does not reconcile with metadata */
run;
```

Because the value specified in the INPUTPARAM statement does not reconcile with limitations that have been specified in metadata, the stored process does not run. It returns the following error message:

```
ERROR: STP: An error occurred preprocessing input parameters:
        com.sas.prompts.InvalidPromptValueException: An error occurred for the
        prompt "x" (values: 4).The value must be greater than or equal to 5.
```

## Example 5

In the following example, INPUTPARAM is overloaded with multiple values:

```
proc stp program='/User Folders/johndoe/inparam';
   inputparam x='5' x='6' x='7';
run;
```

The SAS macro variables that are created for the x parameter are as follows:

```
        X_COUNT 3
        X0 3
        X 5
        X1 5
        X2 6
        X3 7
```

where the value for each SAS macro variable is described as follows:

| | |
|---|---|
| X_COUNT | specifies the number of duplicate macros. |
| X0 | specifies the number of duplicate macros. |
| X | specifies the value of the first instance of a duplicate macro. |
| X1 | specifies the value of the first duplicate; this value is always the same as the value for X. |
| X2 | specifies the value of the second duplicate. |
| X3 | specifies the value of the third duplicate. |
| X$n$ | specifies the value of the $n$th duplicate. |

**Note:** This is the same strategy for specifying duplicate macros as used in SAS/IntrNet and the SAS Stored Process Server.

# Example 6

You can use the INPUTPARAM statement to specify a selection group and selection-dependent groups. For example, you might have a REGPROD selection group defined in metadata, with two selection-dependent groups (Territory and Merchandise) that each have a prompt associated with them (region and product, respectively). The following image depicts how the parameters appear in SAS Management Console:
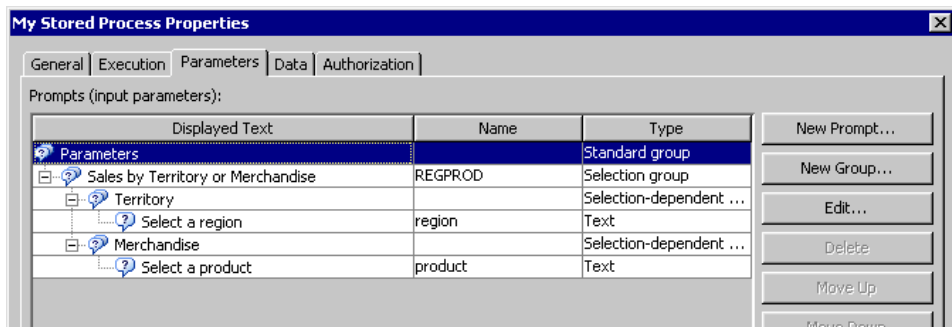


The Selection-Dependent Groups tab in the Edit Group dialog box for the REGPROD selection group shows that the Merchandise selection-dependent group is selected by default.
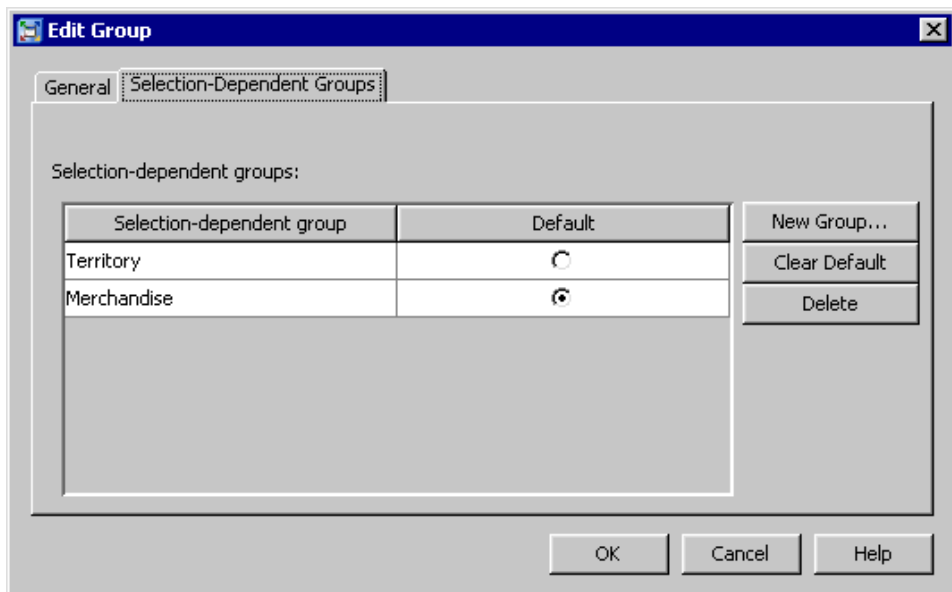


You can use the following INPUTPARAM statements to specify the Territory selection-dependent group (value is `REG`) and the region parameter. After a selection-dependent group (for example, Territory) is specified, you should not specify a prompt from within a different selection-dependent group (for example, product).

```
proc stp program='/User Folders/johndoe/My Stored Process';
   inputparam REGPROD='REG';
   inputparam region='North America';
run;
```

# LIST Statement

Lists details about the stored process that are specified by the PROGRAM= option.

## Syntax

**LIST**< GROUP=*level* | (*level1 level2 ...*)>;

## Optional Argument

**GROUP=*level* | (*level1 level2 ...*)**
    specifies the level of detail that is listed about the stored process that is specified
    by the PROGRAM= option. The stored process is not executed. Specify one or
    more of the following levels:

| | |
|---|---|
| GENERAL | specify this level in order to view a list that contains the following details: stored process name, description, creation date, modification date, keywords, responsible parties. |
| EXECUTION | specify this level in order to view a list that contains the following details: logical server, source in metadata, source code repository, result type. |
| INPUTPARAM | specify this level in order to view a list that contains a list of input parameters for this stored process that are defined in metadata or are defined via an INPUTPARAM statement in PROC STP. Some input parameters, such as _METAPERSON, _METAUSER, and _CLIENT, are automatically defined in metadata without user input. |
| OUTPUTPARAM | specify this level in order to view a list that contains details about output parameters that are defined in metadata. |
| INPUTDATA | specify this level in order to view a list that contains details about input data streams that are defined in metadata. |
| OUTPUTDATA | specify this level in order to view a list that contains details about output data streams that are defined in metadata. |

## Details

The LIST statement displays the metadata attributes of the stored process that is
specified in the PROGRAM= option of the PROC STP statement. These attributes

are displayed in the SAS log. The stored process does not run when the LIST statement is used.

# Examples

# Example 1

The following example uses the GROUP=GENERAL option with the LIST statement:

```
proc stp program='/User Folders/johndoe/myStoredProcess';
   list group=general;
run;
```

The following output is displayed in the SAS log:

```
NOTE: PROC_STP: ====== Metadata Listing for /Users/johndoe/procstp/myStoredProcess
======
NOTE:           Stored Process Name:  myStoredProcess
NOTE:           Description:  Stored Process description to demonstrate the List
statement.
NOTE:           Creation Date:  21Dec2010:14:58:53
NOTE:           Modification Date:  21Dec2010:14:58:53
NOTE:           Keywords:  demo
NOTE:                       list
NOTE:           Responsible Parties:  sasadm
NOTE:                                 sasdemo
```

# Example 2

The following example uses the GROUP=EXECUTION option with the LIST statement:

```
proc stp program='/User Folders/johndoe/myStoredProcess';
   list group=execution;
run;
```

The following output is displayed in the SAS log:

```
NOTE: PROC_STP: ====== Metadata Listing for /Users/johndoe/procstp/myStoredProcess
======
NOTE:           Server Context:  johndoe
NOTE:           Stored Process code may run on any available server
NOTE:           Execution required to be on johndoe application server only
NOTE:           Source Code Repository:  c:\johndoe\progs
NOTE:           Source File:  myStoredProcess.sas
NOTE:           Result Type:  Packages No
NOTE:                         Streaming Yes
```

# Example 3

The following example uses the GROUP=INPUTPARAM option with the LIST statement:

```
proc stp program='/User Folders/johndoe/myStoredProcess';
   list group=inputparam;
   inputparam y=2;
run;
```

The following output is displayed in the SAS log:

```
NOTE: PROC_STP: ====== Metadata Listing for /Users/johndoe/procstp/myStoredProcess
======
NOTE:             Input Parameters:   x = 5
NOTE:                                 Y = 2
NOTE:                                 _result = STREAM
NOTE:                                 _metaperson = sasadm
NOTE:                                 _metauser = sasadm@saspw
NOTE:                                 _client = PROCSTP; TKESTP; JVM 1.6.0_21;
Windows XP (x86)
      5.1
```

# Example 4

The following example uses the GROUP=OUTPUTPARAM option with the LIST statement:

```
proc stp program='/User Folders/johndoe/myStoredProcess';
   list group=outputparam;
run;
```

The following output is displayed in the SAS log:

```
NOTE: PROC_STP: ====== Metadata Listing for /Users/johndoe/procstp/myStoredProcess
======
NOTE:             Output Parameters:  results Integer
NOTE:                                 mean Double
```

# Example 5

The following example uses the GROUP=INPUTDATA option with the LIST statement:

```
proc stp program='/User Folders/johndoe/myStoredProcess';
   list group=inputdata;
run;
```

The following output is displayed in the SAS log:

```
NOTE: PROC_STP: ====== Metadata Listing for /Users/johndoe/procstp/myStoredProcess
======
NOTE:          InputData Sources available:
NOTE:               Source: Generic  Fileref:  istream
NOTE:                  Label: instream
NOTE:                  Expected content type: application/unknown
NOTE:                  Description: Input data stream for stored process
NOTE:                  Allow rewinding stream: Yes
```

# Example 6

The following example uses the GROUP=OUTPUTDATA option with the LIST
statement:

```
proc stp program='/User Folders/johndoe/myStoredProcess';
   list group=outputdata;
run;
```

The following output is displayed in the SAS log:

```
NOTE: PROC_STP: ====== Metadata Listing for /Users/johndoe/procstp/myStoredProcess
======
NOTE:          OutputData Sources available:
NOTE:              Target: Data Table  Table Parameter:    class
NOTE:                  Label: class
NOTE:                  Description: Output data table from stored process
execution
```

# Example 7

The following example shows how to use multiple options with the LIST statement:

```
proc stp program='/User Folders/johndoe/myStoredProcess';
   list group=( general execution inputparam outputdata );
run;
```

# Example 8

When the LIST statement has no arguments, it is functionally equivalent to having
all the group options set. So the following two LIST invocations produce the same
output:

```
proc stp program='/User Folders/johndoe/myStoredProcess';
   list;
run;

proc stp program='/User Folders/johndoe/myStoredProcess';
   list group=( general execution inputparam outputparam inputdata outputdata );
run;
```

# LOG Statement

Controls the location of the stored process log.

## Syntax

**LOG** FILE=*local-fileref | local-file-path | local-fileref('file')*;

## Required Argument

**FILE=*local-fileref | local-file-path | local-fileref('file')***
  specifies the filename and location for the stored process log. Specify either an
  explicit local path, ending with the filename for the log, or specify a local fileref.
  You can also specify a fileref of an aggregate storage location and the name of a
  file or member, enclosed in parentheses, that resides in that location. To use
  aggregate syntax, you must have previously associated the fileref with a file in a
  FILENAME statement, a FILENAME function, or an appropriate operating
  environment command.

## Details

The LOG statement enables the user to direct the PROC STP and stored process
log to an external file. Without the LOG statement, the PROC STP logging
messages go to the standard SAS output location.

## Examples

## Example 1

```
proc stp program='/User Folders/johndoe/myStoredProcess';
   log file = 'c:\johndoe\logs\procstp.txt';
run;
```

## Example 2

```
filename mylogf 'C:\johndoe\logs\filereflog.txt';

proc stp program='/User Folders/johndoe/myStoredProcess';
```

```
        log file = mylogf;
run;
```

---

# Example 3

The following example uses aggregate syntax to specify the filename and location for the stored process log.

```
filename mylogf 'C:\johndoe\logs';

proc stp program='/User Folders/johndoe/myStoredProcess';
   log file = mylogf('filereflog.txt');
run;
```

---

# OUTPUTDATA Statement

Defines output data files that are produced by the execution of the stored process.

---

## Syntax

**OUTPUTDATA** *output-data-target(s)* </ <REMOTE> <MEMTYPE=DATA | CATALOG>>;

## Required Argument

***output-data-target(s)***
can have the following form:

*stored-process-data-file<=member-name | 'data-set-path'>*

| | |
|---|---|
| *stored-process-data-file* | specifies the name of an output data file. This name corresponds to an output parameter (macro variable) that is visible to the stored process program. |
| *member-name* | specify a one- or two-level name of a SAS data set or catalog that the stored process can access directly. If the stored process outputs data to an existing catalog, the catalog must also be specified as an input data source to avoid overwriting an entire catalog with what might be only a few entries created by the stored process. |
| *'data-set-path'* | provide alternate syntax for specifying a data set. |

## Optional Arguments

**REMOTE**

creates or updates the output data target on the remote server. If specifying a *member-name* with a libref, that libref must already exist in the remote server session. This option is not supported for local execution.

**MEMTYPE=DATA | CATALOG**

specifies the member type of the output data target. The value that you specify for this option must match the type of data set that you are accessing. If you do not specify this option, then a data set is used as the output data target. Valid values are:

DATA        specifies that the output data target is a data set.

CATALOG   specifies that the output data target is a catalog. If the stored process accesses a catalog, then the MEMTYPE=CATALOG option is required.

**Alias**   MT

# Details

Multiple OUTPUTDATA statements can be specified in a single procedure call. PROC STP prepends the OUTPUTDATA label with _TARGET_, in order to be compliant with how data targets are handled in SAS Management Console.

# Examples

# Example 1

The following example executes a stored process with two output data files, demonstrating different use cases for the OUTPUTDATA statement:

```
proc stp program='/Products/SAS Intelligence Platform/Samples/
     Example Stored Process';
  outputdata data1=mylib.mydata;          /* output data file data1 is
                                             created as mylib.mydata */
  outputdata data3="/tmp/mydata.sas7bdat"; /* output data file data3 is
                                             created at the specified
                                             path */
  ...
  run;
```

# Example 2

The following example shows the interaction between PROC STP and the stored process:

```
/* Create a libref in the proc stp environment */
libname mylib 'c:\johndoe\temp';

/* Run a stored process that creates a dataset */
proc stp program='/User Folders/johndoe/createData';
   /* Pass in a reference, "outdata" to the dataset */
   outputdata  outdata=mylib.employees;


run;

/* After the stored process creates the dataset,
   print it in our local environment */
proc print data=mylib.employees;
   Title 'Our Employees';
run;quit;
```

The following code is the createData stored process that creates the data set. Notice that in this example, PROC STP has modified outdata to be _TARGET_outdata. Within this stored process, the code uses the _TARGET_outdata data set reference that was passed in by PROC STP:

```
data &_TARGET_outdata;
   length First_Name $8 Last_Name $9 Job_Title $8;
   input First_Name $ Last_Name $
         Job_Title $ Salary;
   datalines;
Joshua Alexander Auditor 41423
Gabriel Ryan Trainee 26018
Eileen Joy Manager 55236
Richard Collier CEO 75000
;
run;
```

# Example 3

The following example uses the PROC STP WORK library with the OUTPUTDATA statement:

```
/* Run a stored process that creates a dataset */
proc stp program='/User Folders/johndoe/createData';
   /* Pass in a reference, "outdata" to the dataset */
   outputdata  outdata=work.employees;


run;

/* After the stored process creates the dataset,
   print it in our local environment */
proc print data=work.employees;
   Title 'Our Employees';
run;quit;
```

# OUTPUTFILE Statement

Defines output files that are produced by the execution of the stored process.

## Syntax

**OUTPUTFILE** *output-file(s)* </ REMOTE >;

## Required Argument

***output-file(s)***
    can have the following form:

    *stored-process-file<=fileref | 'file-path' | fileref('file')>*

| | |
|---|---|
| *stored-process-file* | specifies the name of an output file. This name corresponds to a fileref that is visible to the stored process program. The short form of the OUTPUTFILE statement, OUTPUTFILE *stored-process-file*;, is equivalent to OUTPUTFILE *stored-process-file=stored-process-file*;. |
| *fileref* | specifies the name of a fileref. |
| '*file-path*' | specifies the path of a file. |
| *fileref*('*file*') | specifies a fileref of an aggregate storage location and the name of a file or member, enclosed in parentheses, that resides in that location. You must have previously associated the fileref with a file in a FILENAME statement, a FILENAME function, or an appropriate operating environment command. |

## Optional Argument

**REMOTE**
    creates or updates the output file on the remote server. The *stored-process-file* fileref must already exist in the remote server session. This option is not supported for local execution.

## Details

Multiple OUTPUTFILE statements can be specified in a single procedure call.

# Examples

## Example 1

The following example executes a stored process with three output files, demonstrating different use cases for the OUTPUTFILE statement:

```
proc stp program='/Products/SAS Intelligence Platform/Samples/
    Example Stored Process';
  ...
  outputfile file1;                     /* makes the data from stored process file1
                                              fileref available to local file1 fileref */
  outputfile file2=myfile;              /* makes the data from stored process file2
                                              fileref available to local myfile fileref */
  outputfile file3="/tmp/data3.csv";  /* makes the data from stored process file3
                                              fileref available to the specified local
                                              file */
  ...
  run;
```

## Example 2

The following example writes stored process output to a file by using a local file path:

```
proc stp program='/User Folders/johndoe/hello';
   outputfile _webout='c:\johndoe\data\hello.html';
run;
```

The stored process writes HTML code to the _WEBOUT fileref that is passed in by PROC STP:

```
data _null_;
   file _webout;
   put '<HTML>';
   put '<HEAD><TITLE>Hello World!</TITLE></HEAD>';
   put '<BODY>';
   put '<H1>Hello World!</H1>';
   put '</BODY>';
   put '</HTML>';
run;
```

## Example 3

The following example uses a local fileref with the same name as the reference in the stored process:

```
filename _webout 'c:\johndoe\data\hello.html';

proc stp program='/User Folders/johndoe/hello';
```

```
   outputfile  _webout;
run;
```

# Example 4

The following example uses a local fileref with a different name from the reference in the stored process:

```
filename myfile 'c:\johndoe\data\hello.html';

proc stp program='/User Folders/johndoe/hello';
   outputfile  _webout=myfile;
run;
```

# Example 5

The following example uses aggregate syntax to specify the filename and location for the output file.

```
filename localfref 'c:\johndoe\data';

proc stp program='/User Folders/johndoe/hello';
   outputfile hello = localfref('hello.html');
run;
```

# OUTPUTPARAM Statement

Defines output parameters that are produced by the execution of the stored process.

## Syntax

**OUTPUTPARAM** *parameter-name-1<="local-variable-name-1">*
*<parameter-name-2<="local-variable-name-2"> …>;;*

## Required Argument

**parameter-name*<="local-variable-name">***
specifies output parameter names, or specifies output parameters as name/ value pairs where the values are local macro variables. Each name or name/ value pair is converted to a SAS macro variable, which can be used in the stored process. The parameter name is the name of the macro variable that contains the parameter value. The parameter name must be a valid macro variable name and cannot be longer than 32 characters in length. You can specify the name of a local macro variable, as follows:

| *local-variable-name* | specifies the name of a local macro variable that contains the output parameter value. The macro variable is set to the output parameter value at the completion of stored process execution. If *local-variable-name* is omitted, the parameter name is used as the local variable name. The local variable is not set if the stored process fails to set an output parameter value. |
| --- | --- |

## Details

The OUTPUTPARAM statement identifies macro variables within the stored process environment to retrieve and use in the PROC STP SAS session. You can also create local macro variables within the PROC STP SAS session and assign the stored process output parameter to it.

## Examples

## Example 1

The following example retrieves the resultValue macro variable from the stored process environment and uses it in the PROC STP session. The stored process, outputparam, contains the following code:

```
…
%let tempVal = 5;
%let tempVal2 = 10;

%global resultValue;
%let resultValue = %eval(&tempVal2 + &tempVal);
…
```

The following PROC STP code is used in the SAS session:

```
proc stp program='/User Folders/johndoe/outparam';
   outputparam resultValue;
run;

/* After PROC STP runs, resultValue is set in the stored process
   and is available in the local SAS environment */
%put NOTE: The Stored Process set resultValue=&resultValue;
```

The following message appears in the SAS log:

```
NOTE: The Stored Process set resultValue=15
```

## Example 2

The following example retrieves the resultValue macro variable from the stored process environment and assigns it to a local variable in the SAS session:

```
proc stp program='/User Folders/johndoe/outparam';
   outputparam resultValue=myLocalResult;
run;

%put NOTE: The SAS Session local macro myLocalResult=&myLocalResult;
```

The following message appears in the SAS log:

```
NOTE: The SAS Session local macro myLocalResult=15
```

# Example: Generating an ODS Document File STP Procedure

## Example 1: Generating an ODS Document File

| | |
|---|---|
| Features: | PROC STP statement options<br>    PROGRAM=<br>    ODSOUT=<br>PROC DOCUMENT statement options<br>    NAME=<br>REPLAY statement<br>System options<br>    METASERVER<br>    METAPORT<br>    METAUSER<br>    METAPASS<br>ODS _ALL_ CLOSE statement<br>ODS HTML statement<br>GOPTIONS statement |
| ODS destinations: | DOCUMENT, HTML |

# Details

This sample code is intended to be executed in an interactive or batch SAS session. You need to specify metadata connection information so that your SAS session connects to the metadata server that contains the "Cholesterol by Sex and Age Group" stored process sample.

Starting with SAS 9.3, stored process source code can be stored on the SAS Metadata Server. The SAS source code for this stored process is stored in metadata. Storing source code in metadata is recommended for stored processes that are run using PROC STP. The source code must be stored in a location that is accessible to the server session.

# Program

```
ods _all_ close;

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

proc stp program='/Products/SAS Intelligence Platform/Samples/
    Sample: Cholesterol by Sex and Age Group'
odsout=store;
run;

goptions device=png;

ods html path='your-output-directory' file='test.htm' style=HTMLBlue;
  proc document name=&_ODSDOC (read);
    replay / levels=all;
  run;


ods html close;
```

# Program Description

**Close all open ODS destinations.** The ODS _ALL_ CLOSE statements closes all open destinations.

```
ods _all_ close;
```

**Provide connection information to the metadata server where the stored process is registered.** The METASERVER= option specifies the host name or address of the metadata server, the METAPORT= option specifies the TCP port for the metadata server, the METAUSER= option specifies the user ID for connecting to the metadata server, and the METAPASS= option specifies the password for the metadata server.

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

**Execute the stored process and generate a destination- and device-independent ODS Document file.** The two-level name of the ODS Document file is stored in the _ODSDOC reserved macro variable, and this document is replayed using the DOCUMENT procedure. The stored process source code is stored in metadata.

```
proc stp program='/Products/SAS Intelligence Platform/Samples/
   Sample: Cholesterol by Sex and Age Group'
odsout=store;
run;
```

**Specify the graphics device for the output.** The GOPTIONS DEVICE=PNG statement specifies PNG as the graphics device.

```
goptions device=png;
```

**Use PROC DOCUMENT to create an HTML file containing the output of the stored process.** The ODS HTML statement creates your file in your directory and applies the HTMLBlue style. The DOCUMENT procedure replays the stored process output. For more information about PROC DOCUMENT, see "DOCUMENT Procedure" in *SAS Output Delivery System: Procedures Guide*.

```
ods html path='your-output-directory' file='test.htm' style=HTMLBlue;
  proc document name=&_ODSDOC (read);
    replay / levels=all;
  run;
```

**Close the HTML destination and then view the HTML file using a web browser.**

```
ods html close;
```

# Appendix 1

# Stored Process Software Requirements

# General Requirements

To manage and execute SAS Stored Processes for any client environment, you must have the following components installed:

- SAS software

- SAS Management Console

For general information about installing each of these components, see "Components" on page 182.

# Client-Specific Requirements

Stored processes can be accessed from many different client environments. Software requirements vary depending on the client environment.

To use SAS Stored Processes in a web application environment, the following components are recommended:

- Java Runtime Environment (JRE) or Java Development Kit (JDK)

- servlet container

■ SAS Web Infrastructure Platform

To use SAS Stored Processes in a Java application, the following components are required:

■ Java Development Kit (JDK)

■ servlet container (for servlets or JSPs only)

To access SAS Stored Processes from Microsoft Office, the following component is required:

■ SAS Add-In for Microsoft Office

To access SAS Stored Processes from a web services client, install the following component:

■ SAS BI Web Services

To author SAS Stored Processes in a task-oriented user interface, install the following component:

■ SAS Enterprise Guide

You can install all of the components on a single system or install them across multiple systems. A development system might have all of the components on a single desktop system, while a production system might have SAS installed on one or more systems, a servlet container installed on another system, and client software installed on multiple client desktop systems. For specific requirements about host platforms, see the product documentation for the individual components.

For general information about installing each of these components, see "Components" on page 182.

# Components

SAS software
    Install SAS 9.4 on your designated SAS server. You must install Base SAS and SAS Integration Technologies in order to run stored processes. SAS/GRAPH software is required to run some of the sample stored processes. Install any other products that are used by your stored processes. You must also configure a SAS Metadata Server in order to create and use stored processes. You must configure one or more stored process servers or workspace servers in order to execute stored processes.

SAS Management Console
    Install SAS Management Console on any system with network access to your SAS server.

Java Runtime Environment (JRE)
    The Java interface to SAS Stored Processes requires the Java 2 Runtime Environment (JRE), Standard Edition. For information about the specific version required for your operating environment, see the installation instructions for SAS Management Console. Some development environments and servlet containers include a copy of the appropriate version of the Java 2 JRE. If you need a copy, you can download it from the Third-Party software CD in the SAS Installation Kit. If you are developing Java applications or creating Java Server Pages (JSPs),

then you also need the Java 2 Software Development Kit (SDK), which includes a copy of the Java 2 JRE.

Java Development Kit (JDK)

Java developers or servlet containers that execute Java Server Pages (JSPs) require the Java 2 Software Development Kit (SDK), Standard Edition. For information about the specific version required for your operating environment, see the installation instructions for SAS Management Console. Some development environments and servlet containers include a copy of the appropriate version of the Java 2 SDK. If you need a copy, you can download it from the Third-Party software CD in the SAS Installation Kit.

Servlet Container

A servlet container is a Java server that can act as a middle-tier access point to SAS Stored Processes. A servlet container can be used to host the SAS Web Infrastructure Platform or user-written servlets or Java Server Pages. For specific servlet container requirements for the SAS Web Infrastructure Platform, see the product documentation. Servlet containers used for user-written servlets or JSPs must include a JRE version that is compatible with the SAS 9.4 requirements for the operating environment. SAS Web Application Server is the supported servlet container for SAS 9.4

SAS Web Infrastructure Platform

The SAS Web Infrastructure Platform is installed on a servlet container and includes the SAS Stored Process Web Application. This web application enables you to execute stored processes from a web browser or other web client.

SAS Add-In for Microsoft Office

SAS Add-In for Microsoft Office must be installed on a client Windows system in order to execute stored processes from Microsoft Office on that system. The SAS Integration Technologies client for Windows must also be installed on the same system.

SAS BI Web Services for Java

SAS BI Web Services requires that several other components be installed, including the SAS Web Infrastructure Platform. For more information about required components, see the installation instructions for SAS BI Web Services.

SAS Enterprise Guide

SAS Enterprise Guide is a Microsoft Windows client application that can be installed on any system that has network access to your SAS server.

# Appendix 2

# Converting SAS/IntrNet Programs to SAS Stored Processes

# Overview

To fully use the capabilities of the SAS®9 Enterprise Intelligence Platform, you can convert existing SAS/IntrNet applications into SAS Stored Processes. Many features are implemented in the SAS Stored Process Server and the SAS Stored Process Web Application to minimize the code changes that are required during a conversion. Existing SAS/IntrNet Application Dispatcher programs can usually be converted to streaming stored processes with minimal or no modifications. This

appendix explains how to perform such a conversion and discusses some of the differences between Application Dispatcher programs and stored processes.

SAS/IntrNet Application Dispatcher programs execute very much like a stored process web application. Although Application Dispatcher is only one component of SAS/IntrNet, applications that use Application Dispatcher are the most likely candidates for conversion to stored processes, because these applications execute SAS programs with features that are very similar to stored processes. This appendix focuses only on the conversion of Application Dispatcher programs to stored processes.

SAS/IntrNet continues to be supported, but the stored process framework is a new architecture designed specifically for the SAS®9 platform.

You should convert SAS/IntrNet applications to use stored processes if you want to use the following features:

■ SAS programs in other clients, such as SAS Enterprise Guide, SAS Web Report Studio, or Microsoft Office applications

■ the common security model that is provided by metadata

■ the centralized administration that is provided by metadata integration and SAS Management Console

**Note:** This appendix focuses on converting SAS/IntrNet programs to stored processes that run in the SAS Stored Process Web Application. If you want to run these programs in other stored process clients (such as SAS Web Report Studio or the SAS Add-In for Microsoft Office), there might be additional configuration issues. Each client application has its own requirements for stored process behavior.

# Compatibility Features

The following features describe similarities between Application Dispatcher and stored processes:

■ The SAS Stored Process Web Application (a component of the SAS Web Infrastructure Platform) provides the middle-tier equivalent of the Application Broker. The SAS Stored Process Web Application requires a servlet container such as SAS Web Application Server. For other requirements, see the SAS Web Infrastructure Platform installation instructions.

■ The SAS Stored Process Server (a component of SAS Integration Technologies) provides the equivalent of the SAS/IntrNet Application Server. The typical stored process server configuration (a load-balanced cluster) is very similar in functionality to a SAS/IntrNet pool service. New servers are started as demand increases to provide a highly scalable system.

■ Streaming output from a stored process is written to the _WEBOUT fileref. The underlying access method has changed, but the functionality is very similar to the functionality of SAS/IntrNet. ODS, HTML Formatting Tools, DATA step code, or SCL programs can continue to write output to _WEBOUT.

■ The Application Server functions (APPSRVSET, APPSSRVGETC, APPSRVGETN, APPSRV_HEADER, APPSRV_SESSION, and

APPSRV_UNSAFE) are supported in stored processes except as noted in the "Conversion Considerations" section. In many cases, equivalent STPSRV functions are recommended for new programs.

■ The _REPLAY mechanism is supported by the stored process server. The value of the _REPLAY URL has changed, but this does not affect most programs.

■ The SAS/IntrNet sessions feature has been implemented by the stored process server. The same SAVE library, session macro variables, and session lifetime management functions are available in stored processes.

# Conversion Considerations

## Overview of Conversion Considerations

There are a number of differences in the stored process server environment that might affect existing SAS/IntrNet programs. Use the items in the following sections as a review checklist for your existing programs.

## HTTP Headers

■ In SAS Stored Processes, HTTP headers cannot be written directly to _WEBOUT by using a DATA step PUT statement or SCL FWRITE function. You must use the STPSRV_HEADER (or APPSRV_HEADER) function to set header values. Automatic header generation cannot be disabled with `appsrvset("automatic headers", 0)`.

■ SAS/IntrNet programs require that HTML Formatting Tools use the RUNMODE=S option, which writes an HTML header directly to _WEBOUT. For stored process programs, you should change the option to RUNMODE=B, or an extra header line appears in the output.

## Macro Variables

■ Unsafe processing is different for stored processes; there is no UNSAFE option. Unsafe characters (characters that cause unwanted macro language processing) are quoted instead of removed from the input parameters, so you can safely use the &VAR syntax without worrying about unsafe characters. The following examples work without using the APPSRV_UNSAFE function:

```
%if &MYVAR eq %nrstr(A&P)
%then do something...;
```

Here is another example:

```
data
_null_;
file _webout;
put "MYVAR=&MYVAR";
run;
```

APPSRV_UNSAFE works in the stored process server and still returns the complete, unquoted input value. This change might cause subtle behavioral differences if your program relies on the SAS/IntrNet unsafe behavior. For stored processes, use the STPSRV_UNQUOTE2 function instead.

■ The _REPLAY macro variable does not have the same syntax in stored processes as it did in Application Dispatcher. References to `&_REPLAY` are not recommended for SAS/IntrNet programs, but they can be used in stored processes. The DATA step function `symget('_REPLAY')` does not return a usable URL in a stored process and should be replaced with `"&_REPLAY"`. For example:

```
url =
symget('_REPLAY')...;
```

should be changed to

```
url =
%str(&_REPLAY)...;
```

However, if you were already using `%str(&_REPLAY)` in SAS/IntrNet, then no change is necessary.

■ The _SERVICE, _SERVER, and _PORT macro variables do not exist for stored processes. You must review any code that uses these macro variables. Usually, they are used to create drill-down URLs or forms. In many cases, this code does not require any change; input values for these variables are ignored.

■ In stored processes, _PROGRAM refers to a stored process path and name in the metadata repository folder structure, and not a three-level or four-level program name. Any programs that create drill-down links or forms with _PROGRAM must generally be modified to use the stored process path.

## Code Differences

■ The stored process server cannot directly execute SOURCE, MACRO, or SCL catalog entries. You must use a wrapper .sas source file to execute the catalog entry.

■ Instead of using an ALLOCATE LIBRARY or ALLOCATE FILE statement to assign a library, as you can do in SAS/IntrNet, you must assign a library in one or more of the following ways:

  □ using the Data Library Manager plug-in for SAS Management Console

  □ using the server start-up command or the SAS config file

  □ using a SAS autoexec file

For more information about how to assign libraries, see the *SAS Intelligence Platform: Data Administration Guide*.

■ There is no REQUEST TIMEOUT functionality in stored processes; `appsrvset('request timeout')` is not supported.

- The Application Server functions APPSRV_AUTHCLS, APPSRV_AUTHDS, and APPSRV_AUTHLIB are not supported in stored processes. There are no STPSRV functions that are equivalent to these Application Server functions.

- Stored processes do not support the SESSION INVSESS automatic user exit program. Similar functionality can be implemented in the SAS Stored Process Web Application through a custom web interface.

- AUTH=HOST functionality is not supported by the stored process server. In Application Dispatcher, this functionality provides the ability for the program to run under the operating system permissions of the client user.

- If you are writing to _WEBOUT by using PUT statements while ODS has _WEBOUT open, when you execute the code the PUT statement data might be out of sequence with the data that is generated by ODS. This problem occurs with both SAS/IntrNet applications and stored processes. It tends to be more of an issue if you are upgrading from SAS 8 to SAS®9. This problem occurs because both your code and ODS are opening the same fileref at the same time. For example, the following code might not always work as expected:

```
ods listing close;
ods html body=_webout path=&_tmpcat
(url=&_replay) Style=Banker;
... other code ...
data _null_;
file _webout;
put '<p align="center"> </p>' ;
put '<p align="center"><b>Test.
If you see this in order, it worked.</b></p>';
run;
... other code ...
ods html close;
```

This code might work in some SAS/IntrNet programs, but it can cause problems with the order of data even in SAS/IntrNet. This code is more likely to fail in a stored process. This problem can be fixed by inserting PUT statements before you open ODS, closing ODS while you write directly to the fileref, or using the `ODS HTML TEXT="string"` option to write data. The following code is an example of how you can both close ODS while you write directly to the fileref, and insert your PUT statements before you open ODS:

```
ods html
body=_webout
(no_bottom_matter)...;
... other code ...
ods html close;

data _null_;
file _webout;
put '<p align="center"> </p>' ;
put '<p align="center"><b>Test.
If you see this in order, it worked.</b></p>';
run;

ods html body=_webout (no_top_matter)...;
... other code ...
ods html close;
```

The following code is an example of how you can use the `ODS HTML TEXT="string"` option to write data:

```
ods
listing
close;
ods html body=_webout path=&_tmpcat
(url=&_replay) Style=Banker;
... other code ...
ods html text='<p align="center"> </p>' ;
ods html text='<p align="center"><b>Test.
If you see this in order, it worked.</b></p>';
... other code ...
ods html close;
```

# Overview of Conversion Steps

To convert existing SAS/IntrNet programs to stored processes, perform the following steps:

1  Install and configure the SAS Web Infrastructure Platform, a component of SAS Integration Technologies that includes the SAS Stored Process Web Application, which is used to emulate the Application Broker.

2  Copy the program to a valid source code repository for a stored process server.

   **Note:** Starting with SAS 9.3, you can store source code in the stored process metadata rather than on the application server. To do this, when you register the stored process metadata, click **Edit Source Code** in the New Stored Process wizard or the **Execution** tab of the Stored Process Properties dialog box. Copy and paste the code into the buffer provided.

3  Modify the program as required to address the items discussed in the Conversion Considerations. For more information, see "Conversion Considerations" on page 187.

4  Register the stored process using the New Stored Process wizard in SAS Management Console.

5  Modify the HTML for any custom input forms. Also, convert any HTML pages that link to your stored process to use the SAS Stored Process Web Application URL syntax.

6  Run the stored process.

*Example* **191**

# Example

## Sample Environment

The following software makes up the environment for this example:

- The web server for the SAS/IntrNet portion of the example is Microsoft Internet Information Services (IIS) 6.0.
- SAS Web Application Server is the servlet container that is being used for this example.
- SAS 9.4
- SAS Stored Process Server (as opposed to the SAS Workspace Server)
- Windows middle tier
- Windows SAS server tier

## About the Application Dispatcher Program

### The Program Component

The example in this appendix uses ODS and the TABULATE procedure to display shoe sales data for a selected region. Here is the SAS code:

```
%global regionname;

ods listing close;
ods html body=_webout;

proc tabulate data = sashelp.shoes format = dollar14.;
title "Shoe Sales for &regionname";
   where (product =: 'Men' or product =: 'Women') & region="&regionname";
   table Subsidiary all,
       (Product='Total Product Sales' all)*Sales=' '*Sum=' ';
   class Subsidiary Product;
   var Sales;
   keylabel All='Grand Total'
            Sum=' ';
run;

ods html close;
```

For the sake of illustration, assume that this SAS code is stored in the location `C:\MySASFiles\intrnet\webtab1.sas`.

# The Input Component

The following HTML is the body for the input component, which is the physical HTML file. For the sake of illustration, assume that this HTML is stored in a file named webtab1.html.

```
<H1>Regional Shoe Sales</H1>
<p>Select a region in order to display shoe sales data for that
region by subsidiary and style.  This sample program uses ODS and
the TABULATE procedure.</p>

<HR>

<FORM ACTION="/sasweb/cgi-bin/broker.exe">
<INPUT TYPE="HIDDEN" NAME="_SERVICE" VALUE="default">
<INPUT TYPE="HIDDEN" NAME="_PROGRAM" VALUE="intrnet.webtab1.sas">

<b>Select a region:</b>  <SELECT NAME="regionname">
<OPTION VALUE="Africa">Africa
<OPTION VALUE="Asia">Asia
<OPTION VALUE="Central America/Caribbean">Central America/Caribbean
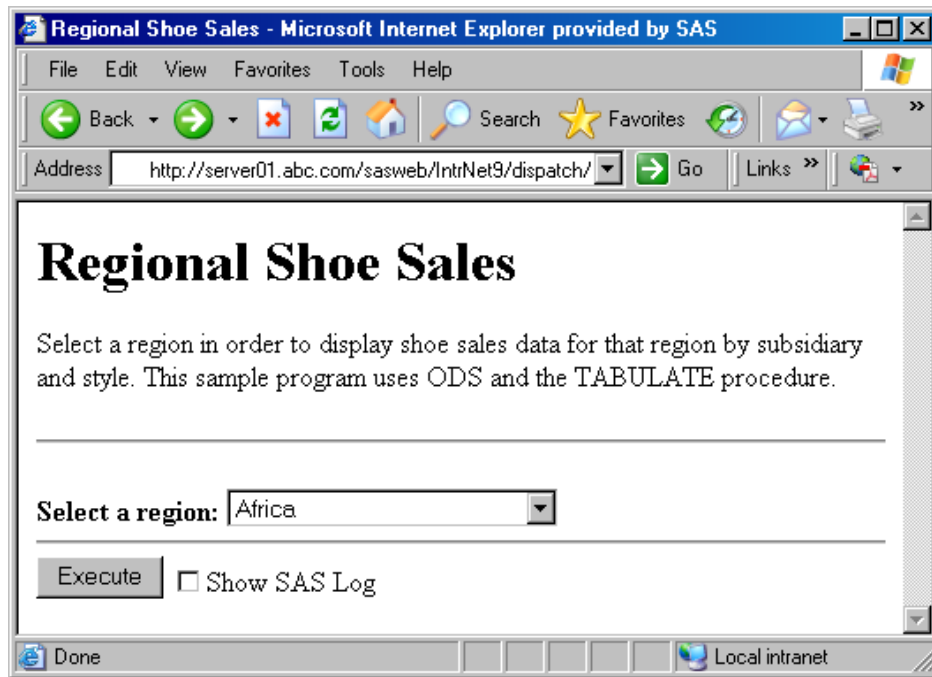<OPTION VALUE="Eastern Europe">Eastern Europe
<OPTION VALUE="Middle East">Middle East
<OPTION VALUE="Pacific">Pacific
<OPTION VALUE="South America">South America
<OPTION VALUE="United States">United States
<OPTION VALUE="Western Europe">Western Europe
</SELECT>

<HR>
<INPUT TYPE="SUBMIT" VALUE="Execute">
<INPUT TYPE="CHECKBOX" NAME="_DEBUG" VALUE="131">Show SAS Log

</FORM>
```

The input component looks like the one shown in the following display.

*Example* **193**

**Figure A4.1**   *Application Dispatcher Input Component*



You can select a region from the list and click **Execute** to display a table of sales data for that region. When you click **Execute**, Application Dispatcher executes the program and sends the results back to the web browser. The results look like the program output shown in the following display.

*Figure A4.2*  *Application Dispatcher Program Output*
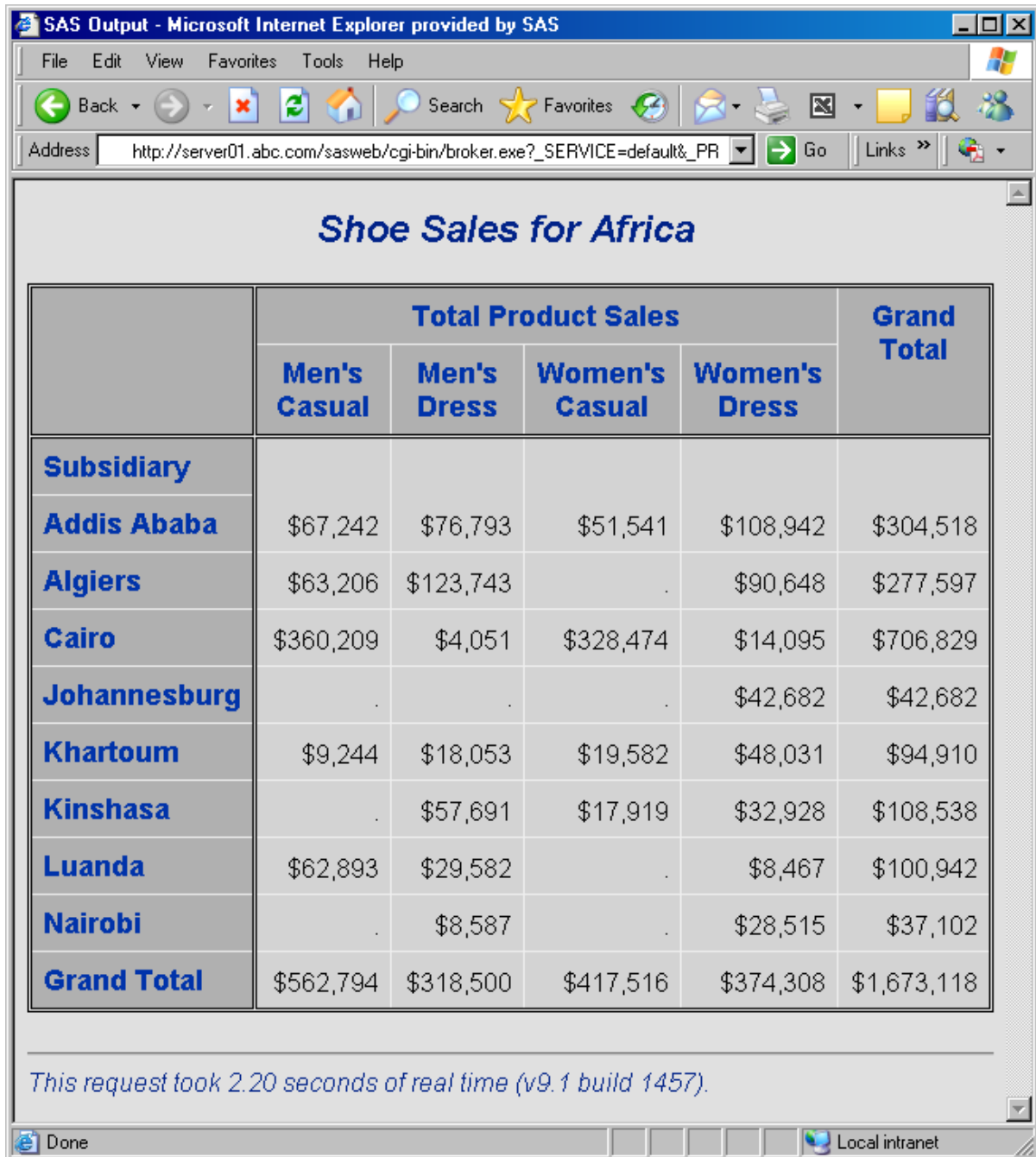


The HTML form created the following URL for the results page, based on the default selections in the input form:

```
http://myserver/sasweb/cgi-bin/broker.exe?
_SERVICE=default&_PROGRAM=intrnet.webtab1.sas&regionname=Africa
```

The URL is typically built for you by the web browser which uses fields in an HTML form. The HTML page uses the following FORM tag to submit the program to the Application Broker:

```
<FORM ACTION="/sasweb/cgi-bin/broker.exe">
```

*Example*   **195**

The following hidden fields are used to create name/value pairs to complete the required syntax:

```
<INPUT TYPE="HIDDEN" NAME="_SERVICE" VALUE="default">
<INPUT TYPE="HIDDEN" NAME="_PROGRAM" VALUE="intrnet.webtab1.sas">
```

Notice that the value for _PROGRAM is set to `intrnet.webtab1.sas`. This first level indicates that the program is stored in a directory identified by the intrnet fileref. The next two levels (`webtab1.sas`) provide the name of the program that is executed.

**Note:** Several samples are shipped with Application Dispatcher, and you can use these samples to practice the conversion to stored processes. (Some of these samples have already been converted to stored process samples, and these samples are installed with SAS Integration Technologies.) You can execute the Application Dispatcher samples from the following URL: http://myserver/sasweb/IntrNet9/dispatch/samples.html

# Converting the Application Dispatcher Program to a Stored Process

## Step 1: Copy the Source Program

To preserve the functionality of the original SAS/IntrNet example, copy the SAS program to a new location before modifying it. Copy the webtab1.sas program from `C:\MySASFiles\intrnet` to a location on the stored process server, such as `C:\MySASFiles\storedprocesses`.

**Note:** Starting with SAS 9.3, you can store source code in the stored process metadata rather than on the application server. To do this, when you register the stored process metadata, click **Edit Source Code** in the New Stored Process wizard or the **Execution** tab of the Stored Process Properties dialog box. Copy and paste the code into the buffer provided.

## Step 2: Modify the Program as Needed

1  Open the new copy of webtab1.sas to check for conversion considerations.

```
%global regionname;

ods listing close;
ods html body=_webout;

* PROC TABULATE code here;
```

```
ods html close;
```

2 Note that the program is already writing to _WEBOUT, so %STPBEGIN and %STPEND are not needed. However, replacing the ODS HTML statement with %STPBEGIN and %STPEND makes it easier to run the stored process from various clients. This replacement also enables you to run the code from a client like the SAS Stored Process Web Application and specify different values for _ODSDEST. For example, you can change the values of _ODSDEST and generate output as PDF, RTF, or PostScript, without making any SAS code changes.

For this example, delete the two ODS statements at the beginning of the code, and replace them with the following line of code:

```
%stpbegin;
```

Replace the ODS statement at the end of the code with the following statement:

```
%stpend;
```

Check the list of conversion considerations. No further code changes are necessary for this program to run as a stored process. The stored process now consists of the following code:

```
%global regionname;

%stpbegin;

* PROC TABULATE code here;

%stpend;
```

3 Save the changes and close webtab1.sas.

# Step 3: Register the Stored Process in SAS Management Console

**Note:** Before you can register a stored process, a server must be defined for the stored process to run on. Converted SAS/IntrNet programs generally should be registered on a stored process server. If a stored process server is not already defined, then you can use the Server Manager in SAS Management Console to define a server. For more information about how to define a server, see the Help for the Server Manager.

To register a new stored process, complete the following steps:

1 From the Folder view in SAS Management Console, select the folder in which you would like to create the new stored process. For this example, create a `/Converted Samples` folder.

To create a new folder, navigate to where you want to put the new folder. Select **Actions** ⇨ **New** ⇨ **New Folder**. The New Folder wizard appears.

2 Select **Actions** ⇨ **New** ⇨ **Stored Process**. The New Stored Process wizard appears.

*Example* **197**

**3** In the New Stored Process wizard, complete the following steps:

    **a** Enter the following information on the first page of the wizard:

        ■ **Name:** `Regional Shoe Sales`

        ■ **Description: `Converted from SAS/IntrNet program.`**

*Figure A4.3*   *New Stored Process Wizard - Name Specification*



    **b** Click **Next**.

    **c** On the next page of the wizard, specify the following information:

        ■ **Application server: `SASApp`**

        ■ **Server type: `Default server`**

        ■ **Source code location and execution: `Allow execution on selected application server only`, `Store source code on application server`**

        ■ **Source code repository: `C:\MySASFiles\storedprocesses`** (A source code repository is a location on the application server that contains stored process source code. Click **Manage** if you need to add a new source

code repository to the list. For more information about the source code repository, see the New Stored Process wizard Help.)

■ **Source file: webtab1.sas**

■ **Output: Stream**

*Figure A4.4* *New Stored Process Wizard - Execution Details*



d Click **Next**.

The next page of the wizard is where you add parameters. Parameters are optional unless you plan to execute the stored process in other clients that need the metadata information in order to build a dialog box, or if you want to take advantage of the dynamic prompt page that is built by the SAS Stored Process Web Application. Parameters are also useful if you want to restrict input values or types of input. Do not define any parameters right now.

*Example*  **199**

Click **Next**.

The next page of the wizard is where you add data sources and data targets. Do not define any data sources or targets for this stored process.

e  Click **Finish** to register the new stored process.

**Note:** After you have registered the stored process, use the Stored Process Properties dialog box to control access to the stored process. For more information, see the Help for the Stored Process Properties dialog box.

# Step 4: Create a New JSP Page and Modify the HTML

To preserve the functionality of the original SAS/IntrNet example, copy the HTML file to a new location before modifying it.

**Note:** This example shows you how to use the input component from the Application Dispatcher program as a custom input form for the stored process. You can use the _PROGRAM variable along with _ACTION=FORM in the URL to display the custom input form for the stored process. However, copying the HTML file is optional. You can run stored processes without a custom input form.

1  If you want this web page to be used as the default input form in the SAS Stored Process Web Application, then copy the webtab1.html file to a physical location under the SAS Web Application Server directory. The exploded directory might be something like `<SASHOME>\Config\Lev1\Web\WebAppServer` `\SASServer1_1\sas_webapps\sas.storedprocess.war\input` `\Converted_Samples`. (The physical location corresponds to the metadata location. This location is correct only if the new stored process is registered in the `/Converted Samples` folder in the metadata repository. Otherwise, the path is different.)

**Note:** The SAS Stored Process Web Application is delivered in an EAR file, and can be run directly from the EAR file or from the exploded directory. For more information about how to explode the EAR file, see the *SAS Intelligence Platform: Web Application Administration Guide*.

**Note:** You can also copy the HTML file to a new directory under the IIS Web Server, or to a new directory under the SAS Web Application Server directory with the SAS Stored Process Web Application. However, if you decide to do this, you should be aware that appending _ACTION=FORM to the URL to find the custom input form does not work.

2  Modify the HTML page to call the stored process instead of the SAS/IntrNet program.

a  Open webtab1.html in an HTML editor.

b   Change the value of the ACTION attribute in the FORM tag to
    `http://myserver:8080/SASStoredProcess/do`.

c   Remove the hidden field for _SERVICE.

d   Change the value of the hidden field for _PROGRAM to the metadata
    location, and name of the stored process: `/Converted Samples/Regional`
    `Shoe Sales`.

e   You can leave the _DEBUG check box with a value of `131`. This is equivalent
    to the value `LOG,TIME,FIELDS`.

f   You can change the text that appears on the web page. If you want to use
    images, then you need to move them to a location in the current directory or
    change the tag to point back to the old directory.

g   The body of the file now contains the following HTML:

```
<H1>Regional Shoe Sales</H1>
<p>Select a region in order to display shoe sales data for that
region by subsidiary and style.  This sample program uses ODS and
the TABULATE procedure.</p>


<HR>


<FORM ACTION="http://myserver:8080/SASStoredProcess/do">
<INPUT TYPE="HIDDEN"
   NAME="_PROGRAM" VALUE="/Converted Samples/Regional Shoe Sales">


<b>Select a region:</b>  <SELECT NAME="regionname">
<OPTION VALUE="Africa">Africa
<OPTION VALUE="Asia">Asia
<OPTION VALUE="Central America/Caribbean">Central America/Caribbean
<OPTION VALUE="Eastern Europe">Eastern Europe
<OPTION VALUE="Middle East">Middle East
<OPTION VALUE="Pacific">Pacific
<OPTION VALUE="South America">South America
<OPTION VALUE="United States">United States
<OPTION VALUE="Western Europe">Western Europe
</SELECT>


<HR>
<INPUT TYPE="SUBMIT" VALUE="Execute">
<INPUT TYPE="CHECKBOX" NAME="_DEBUG" VALUE="131">Show SAS Log


</FORM>
```

h   Save the file as Regional Shoe Sales.jsp, and close it.

**Note:** If this JSP file is located somewhere other than in the SAS Stored Process
Web Application directory, then you need to specify the complete URL to the stored
process servlet, as follows, in the ACTION attribute in the FORM tag:
`http://myserver:8080/SASStoredProcess/do`. Otherwise, this URL can be a
relative link, as follows: `/SASStoredProcess/do`. If you do place the JSP file under
the same directory as the SAS Stored Process Web Application, then you need to
be careful to preserve the application if you later upgrade or redeploy the SAS
Stored Process Web Application.

*Example* **201**

You should also convert any HTML pages that link to your stored process to use the SASStoredProcess URL syntax. For example, you might use the following URL to link to the Hello World sample program using the Application Broker:

```
http://myserver/cgi-bin/broker?
_service=default&_program=sample.webhello.sas
```

The URL specifies your Application Server, an absolute path to the Application Broker, and the query string (followed by the question mark character). The query string contains the name/value pair data that is input to the application. Each name is separated from the following value by an equal sign (=). Multiple name/value pairs are separated by an ampersand (&). The web page that executes an Application Dispatcher program must pass the _SERVICE and _PROGRAM variables. In this example, the _SERVICE=DEFAULT pair specifies the service that handles this request, and the _PROGRAM=SAMPLE.WEBHELLO.SAS pair specifies the library, name, and type of request program to be executed.

For the SAS Stored Process Web Application, the URL in the preceding example would need to be changed. You might use the following URL if you want to run the program from the SAS Stored Process Web Application:

```
http://myserver:8080/SASStoredProcess/do?
_program=/Samples/Stored+Processes/Sample:+Hello+World
```

The URL specifies your stored process server, an absolute path to the SAS Stored Process Web Application (instead of the Application Broker), and the query string. Notice that `/cgi-bin/broker?` has been replaced with the stored process web application equivalent: `/SASStoredProcess/do?`. The _SERVICE name/value pair is not used with stored processes, and _PROGRAM is the reserved input parameter that specifies the metadata location and the name of the stored process to be executed.

There are special rules for the formatting of name/value pairs in a URL. Special characters (most punctuation characters, including spaces) in a value must be URL-encoded. Spaces can be encoded as a plus sign (+) or %20. Other characters are encoded using the %*nn* convention, where*nn* is the hexadecimal representation of the character in the ASCII character set. In the previous example, the value `/Samples/Stored+Processes/Sample:+Hello+World` actually identifies the stored process named `Sample: Hello World`. The space in the name is encoded as a plus sign (+). If your parameter values contain special characters, then they should be URL-encoded.

# Step 5: Execute the Stored Process Using the New JSP Page

1  You can use _ACTION=FORM in the URL in order to display the custom input form. For example, enter the following URL in a web browser:

```
http://myserver:8080/SASStoredProcess/do?
_program=/Converted+Samples/Regional+Shoe+Sales&_action=form
```

Your web browser is forwarded to the following URL, which displays the modified custom input form:

```
http://myserver:8080/SASStoredProcess/input/Converted_Samples/
Regional_Shoe_Sales.jsp?_program=/Converted Samples/Regional Shoe Sales
```

**Note:** Be sure to start SAS Web Application Server first.

2 Select the default region (Africa) and click **Execute**.

The JSP page executes the stored process by using the following generated URL:

```
http://myserver:8080/SASStoredProcess/do?
_PROGRAM=/Converted Samples/Regional Shoe Sales&regionname=Africa
```

The results look like the results from the Application Dispatcher program as shown in the following display:

*Figure A4.5   Stored Process Results*

*Example*   **203**

# Adding a Parameter to the Stored Process Definition

## Step 1: Modify the Stored Process Metadata Definition

Parameter definitions are not required if you are converting a SAS/IntrNet program to a stored process. If macro variables in the program are used to substitute parameter values in the program, you can define the macro variables as parameters to the stored process. If you define the value as a parameter, it means that other clients can use the metadata to create a dialog box that prompts for the parameter, or you can use the dynamic prompt page that is built by the SAS Stored Process Web Application. If you do not define the parameter, it means that the program must use defaults in the code if you want to execute the stored process in other clients. If you intend to use the stored process in other clients, then you should define parameters in the metadata.

In webtab1.html and webtab1.sas, the REGIONNAME macro variable is substituted into the PROC TABULATE code. Because the HTML form uses a drop-down list, you can count on a valid value always being passed to the program from that web page. If you want to make sure this stored process runs correctly in other clients (or if you want to use the dynamic prompt page that was built by the SAS Stored Process Web Application), then you need to define a parameter that returns a macro variable named REGIONNAME with a valid list of regions.

To add the REGIONNAME parameter, complete the following steps:

1   In SAS Management Console, open the Stored Process Properties dialog box for the Regional Shoe Sales stored process.

2   On the **Parameters** tab, click **New Prompt**.

3   On the **General** tab of the New Prompt dialog box, specify the following information:

   - **Name:** `regionname`

   - **Displayed text:** `Select a region`

   - **Options:** `Requires a non-blank value`

*Figure A4.6* *New Prompt Dialog Box: General Tab*



4   In the **Prompt Type and Values** tab of the New Prompt dialog box, specify the following information:

   - **Prompt type: `Text`**

   - **Method for populating prompt: `User selects values from a static list`**

   - **Number of values: `Single value`**

   - **List of values:**

     ```
     Africa
     Asia
     Central America/Caribbean
     Eastern Europe
     Middle East
     Pacific
     South America
     United States
     Western Europe
     ```

   For the **List of Values** table, click **Add** to add each value. Click the radio button for **Default** next to Africa. For more information about these fields, see the Help for this dialog box.

*Example* **205**

***Figure A4.7*** *New Prompt Dialog Box: Prompt Type and Values Tab*



5 Click **OK** in the New Prompt dialog box, and then click **OK** in the Stored Process Properties dialog box.

# Step 2: Execute the Stored Process Using the Dialog Box

To view the parameter that you added to the stored process metadata definition, execute the stored process using the SAS Stored Process Web Application dialog box instead of the custom input form. The dialog box uses the parameter that you defined in the New Stored Process wizard when you registered the stored process

metadata. To access the dialog box for this stored process, enter the following URL in a web browser:

```
http://myserver:8080/SASStoredProcess/do?
_PROGRAM=/Converted Samples/Regional Shoe Sales&_action=properties
```

*Figure A4.8* *SAS Stored Process Web Application: Dialog Box*
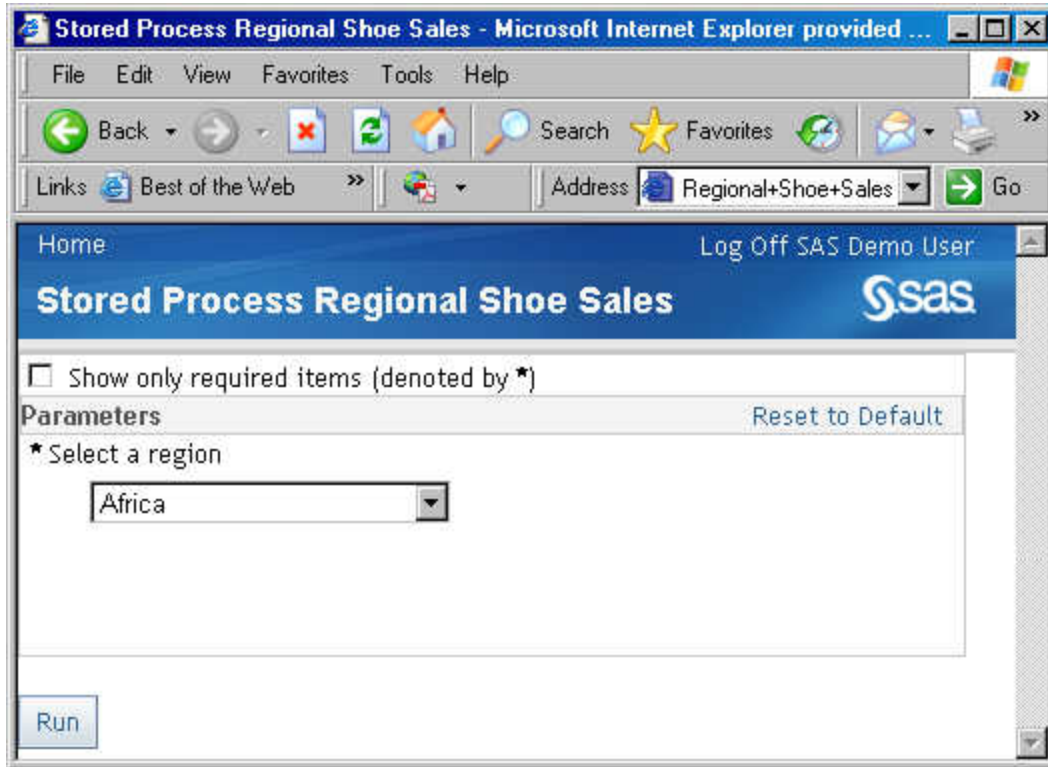


Select the default region (Africa) and click **Execute**. You see the same results (the table of shoe sales for Africa that was shown in "Step 5: Execute the Stored Process Using the New JSP Page" on page 201) displayed in a separate web browser window.

# Executing Catalog Entries

If you are converting SAS/IntrNet programs that use SOURCE or MACRO catalog entries, then you need to use a wrapper .sas source file to execute the catalog entry. As mentioned in the "Conversion Considerations" section, the stored process server cannot directly execute SOURCE, MACRO, or SCL catalog entries.

**Note:** SCL catalog entries cannot be executed using this type of wrapper program.

You can use a wrapper program like the following to execute SOURCE catalog entries:

```
libname mylib 'sas-data-library';  /* this library
```

```
could be pre-assigned */
filename fileref1 catalog 'mylib.catalog.program.source';
%include fileref1;
```

The wrapper program for MACRO catalog entries can be something like the
following wrapper:

```
libname mysas 'SAS-data-library';
/* this library could be pre-assigned */
filename mymacros catalog 'mysas.mycat';
options sasautos=mymacros mautosource;
%macroname;
```

These two sample programs show only the minimum code that is necessary to
execute catalog entries from a SAS program. This might be enough in some cases,
but you might want to use some other SAS/IntrNet features by including macro
variables such as _PGMLIB, _PGMCAT, _PGM, _PGMTYPE, and _APSLIST.

# Formatting Prompt Values and Generating Macro Variables from Prompts

# Entering Prompt Values in the SAS Stored Process Web Application

The following table explains how to format values for the various prompt types in the SAS Stored Process Web Application:

*Table A5.1   Guidelines for Entering Prompt Values (U.S. English Locale)*

| Prompt Type | Guidelines | Examples |
| --- | --- | --- |
| Text | Enter any character value. Blank spaces and nonprintable characters can be used, but the value cannot consist completely of these characters. Trailing blanks are stored as part of the value and are included when the value | ▪ `you are here`<br>▪ `eighty-five`<br>▪ `Bob` |

| Prompt Type | Guidelines | Examples |
|---|---|---|
| | is validated against the minimum and maximum length requirements. | |
| Numeric | Enter a standard numeric value.<br><br>■ If you are working with an integer prompt, then do not use values with decimal places. If you use a value with zeros after the decimal point (for example, `1.00`) for an integer prompt, then the zeros and the decimal point are removed before the value is stored (for example, `1.00` is stored as `1`).<br><br>■ For prompts that allow floating-point values, the unformatted prompt value can contain up to 15 significant digits. Values with more than 15 significant digits of precision are truncated. Note that formatted values can have more than 15 significant digits. | ■ `1.25`<br>■ `6000`<br>■ `2222.444` |
| Date | For dates of type Day, enter values in one of the following formats:<br><br>■ *ddmonth-nameyyyy*<br>■ *mm/dd/yy<yy>*<br>■ *mm.dd.yy<yy>*<br>■ *mm-dd-yy<yy>*<br>■ *month-name/dd/yy<yy>*<br>■ *month-name.dd.yy<yy>*<br>■ *month-name-dd-yy<yy>*<br>■ *month-name dd, yyyy*<br>■ *day-of-week, month-name dd, yy<yy>*<br>■ *yyyy/mm/dd*<br>■ *yyyy.mm.dd*<br>■ *yyyy-mm-dd*<br>■ *yyyy.month-name.dd*<br>■ *yyyy-month-name-dd*<br><br>Here is an explanation of the syntax:<br><br>*day-of-week*<br>specifies either the first three letters of the day of the week or the full name of the day of the week (the full name of the day must be used for values in .NET). This value is not case sensitive. (That is, the lowercase and | ■ `4APR1860`<br>■ `14January1918`<br>■ `12/14/45`<br>■ `02.15.1956`<br>■ `1-1-60`<br>■ `Feb/10/00`<br>■ `March.1.2004`<br>■ `DEC-25-08`<br>■ `SEPTEMBER 20, 2010`<br>■ `FRI, Jan 3, 20`<br>■ `Monday, January 16, 40`<br>■ `2041/5/13`<br>■ `2050.07.25`<br>■ `2100-1-1`<br>■ `2101.December.31`<br>■ `2400-Aug-8` |

| Prompt Type | Guidelines | Examples |
|---|---|---|
| | uppercase versions of the same character are considered to be the same.)<br><br>*dd*<br>  specifies a one-digit or two-digit integer that represents the day of the month.<br><br>*mm*<br>  specifies a one-digit or two-digit integer that represents the month of the year.<br><br>*month-name*<br>  specifies either the first three letters of the month or the full name of the month. This value is not case sensitive. (That is, the lowercase and uppercase versions of the same character are considered to be the same.)<br><br>*yy* or *yyyy*<br>  specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for a four-digit year range from 1600 to 2400. | |
| Date (cont'd.) | For dates of type Week, enter values in one of the following formats:<br><br>■ W*ww yy<yy>*<br><br>■ Week*ww yyyy*<br><br>Here is an explanation of the syntax:<br><br>*ww*<br>  specifies a one-digit or two-digit integer that represents the week of the year. Valid values range from 1 to 52.<br><br>*yy* or *yyyy*<br>  specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for a four-digit year range from 1600 to 2400. | ■ `W1 08`<br><br>■ `W52 1910`<br><br>■ `Week 20 2020`<br><br>■ `Week 5 2048` |
| | For dates of type Month, enter values in one of the following formats:<br><br>■ *mm/yy<yy>*<br><br>■ *mm.yy<yy>*<br><br>■ *mm-yy<yy>* | ■ `12/1828`<br><br>■ `06.65`<br><br>■ `7-76`<br><br>■ `Jul 08`<br><br>■ `JUNE/2010` |

| Prompt Type | Guidelines | Examples |
|---|---|---|
| | ▪ *month-name yy<yy>* | ▪ `SEP.20` |
| | ▪ *month-name/yy<yy>* | ▪ `October-2050` |
| | ▪ *month-name.yy<yy>* | |
| | ▪ *month-name-yy<yy>* | |
| | Here is an explanation of the syntax: | |
| | *mm*<br>     specifies a one-digit or two-digit integer that represents the month of the year. | |
| | *month-name*<br>     specifies either the first three letters of the month or the full name of the month. This value is not case sensitive. (That is, the lowercase and uppercase versions of the same character are considered to be the same.) | |
| | *yy* or *yyyy*<br>     specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for a four-digit year range from 1600 to 2400. | |
| Date (cont'd.) | For dates of type Quarter, enter values in the following format:<br><br>▪ *quarter-name* `quarter` *yyyy*<br><br>Here is an explanation of the syntax:<br><br>*quarter-name*<br>     specifies the quarter of the year. Valid values are `1st`, `2nd`, `3rd`, and `4th`.<br><br>*yyyy*<br>     specifies a four-digit integer that represents the year. Valid values for a four-digit year range from 1600 to 2400. | ▪ `1st quarter 1900`<br><br>▪ `4th quarter 2060` |
| | For dates of type Year, enter values in the following format:<br><br>▪ *yy<yy>*<br><br>Here is an explanation of the syntax:<br><br>*yy* or *yyyy*<br>     specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for | ▪ `1895`<br><br>▪ `86`<br><br>▪ `08`<br><br>▪ `2035` |

| Prompt Type | Guidelines | Examples |
|---|---|---|
| | a four-digit year range from 1600 to 2400. | |
| Time | Enter time values in the following format:<br><br>■ *hh:mm<:ss>* **<AM \| PM>**<br><br>Here is an explanation of the syntax:<br><br>*hh*<br>　specifies a one-digit or two-digit integer that represents the hour of the day. Valid values range from 0 to 24.<br><br>*mm*<br>　specifies a one-digit or two-digit integer that represents the minute of the hour. Valid values range from 0 to 59.<br><br>*ss* (optional)<br>　specifies a one-digit or two-digit integer that represents the second of the minute. Valid values range from 0 to 59. If this value is not specified, then the value defaults to 00 seconds.<br><br>**AM** or **PM** (optional)<br>　specifies either the time period 00:01 – 12:00 noon (AM) or the time period 12:01 – 12:00 midnight (PM). If this value is not specified and you are using the 12-hour system for specifying time, then the value defaults to **AM**. Do not specify **AM** or **PM** if you are using the 24-hour system for specifying time. | ■ `1:1`<br><br>■ `1:01 AM`<br><br>■ `13:1:1`<br><br>■ `01:01:01 PM`<br><br>■ `22:05` |
| Timestamp | Enter timestamp values in one of the following formats:<br><br>■ *mm/dd/yy<yy> hh:mm* AM \| PM<br><br>■ *yyyy-mm-dd*T*hh:mm:ss*<br><br>■ *ddmonth-nameyy<yy> :hh:mm:ss*<br><br>■ *<day-of-week,> month-name dd, yyyy hh:mm:ss* AM \| PM<br><br>Here is an explanation of the syntax:<br><br>*day-of-week* (optional)<br>　specifies either the first three letters of the day of the week or the full name of the day of the week. This value is not case sensitive. (That is, the lowercase and uppercase versions of the same character are considered to be the same.) | ■ `7/3/08 12:40 AM`<br><br>■ `2012-11-23T15:30:32`<br><br>■ `14FEB2020:11:0:0`<br><br>■ `Dec 25, 2020 12:00:00 AM`<br><br>■ `Thursday, November 24, 2050 4:45:45 PM` |

| Prompt Type | Guidelines | Examples |
|---|---|---|
| | *dd*<br>specifies a one-digit or two-digit integer that represents the day of the month. | |
| | *mm*<br>specifies a one-digit or two-digit integer that represents the month of the year. | |
| | *month-name*<br>specifies either the first three letters of the month or the full name of the month. This value is not case sensitive. (That is, the lowercase and uppercase versions of the same character are considered to be the same.) | |
| | *yy* or *yyyy*<br>specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for a four-digit year range from 1600 to 2400. | |
| | *hh*<br>specifies a one-digit or two-digit integer that represents the hour of the day. Valid values range from 0 to 24. | |
| | *mm*<br>specifies a one-digit or two-digit integer that represents the minute of the hour. Valid values range from 0 to 59. | |
| | *ss*<br>specifies a one-digit or two-digit integer that represents the second of the minute. Valid values range from 0 to 59. | |
| Timestamp (cont'd.) | `AM` or `PM` (optional)<br>specifies either the time period 00:01 – 12:00 noon (AM) or the time period 12:01 – 12:00 midnight (PM). If this value is not specified and you are using the 12-hour system for specifying time, then the value defaults to `AM`. Do not specify `AM` or `PM` if you are using the 24-hour system for specifying time. | |
| Color | Enter color values in one of the following formats:<br>■ `CX`*rrggbb*<br>■ `0x`*rrggbb* | Bright red<br>■ `CXFF0000`<br>■ `0xFF0000` |

| Prompt Type | Guidelines | Examples |
| --- | --- | --- |
| | ▪ *#rrggbb*<br><br>Here is an explanation of the syntax:<br><br>*rr*<br>    specifies the red component.<br><br>*gg*<br>    specifies the green component.<br><br>*bb*<br>    specifies the blue component.<br><br>Each component should be specified as a hexadecimal value that ranges from 00 to FF, where lower values are darker and higher values are brighter. | ▪ `#FF0000`<br><br>Black<br><br>▪ `CX000000`<br><br>▪ `0x000000`<br><br>▪ `#000000`<br><br>White<br><br>▪ `CXFFFFFF`<br><br>▪ `0xFFFFFF`<br><br>▪ `#FFFFFF` |
| Data source | Enter the name and location of a data source in the following format:<br><br>▪ */folder-name-1/<.../folder-name-n/ >data-source-name(type)*<br><br>Here is an explanation of the syntax:<br><br>*/folder-name-1/<.../folder-name-n/>*<br>    specifies the location of the data source.<br><br>*data-source-name*<br>    specifies the name of the data source.<br><br>*type*<br>    is the type of data source. The following values are valid unless otherwise noted: `Table`, `InformationMap`, and `Cube`. Use `InformationMap` for specifying either relational or OLAP information maps. | ▪ `/Shared Data/Tables/ OrionStar/ Customers(Table)`<br><br>▪ `/Users/ MarcelDupree/My Folder/My Information Map(InformationMap)`<br><br>▪ `/MyCustomRepository /More Data/ Order_Facts(Table)` |
| File or directory | Enter the name and location of a file or directory in the following format:<br><br>▪ *directory-specification<filename>*<br><br>Here is an explanation of the syntax:<br><br>*directory-specification*<br>    specifies the location of the file or directory in the file system of a SAS server.<br><br>*filename*<br>    specifies the name of the file. This value is required only if the prompt is a file prompt. Depending on the operating environment that the SAS server runs in, you might need to put a forward slash (/) or a backslash (\) between the directory specification and the name of the file. | ▪ `C:\Documents and Settings\All Users \Documents \myfile.txt` |

| Prompt Type | Guidelines | Examples |
| --- | --- | --- |
| Data library | Enter the name and location of a data library in the following format:<br><br>■ */folder-name-1/<.../folder-name-n/ >library-name*`(Library)`<br><br>Here is an explanation of the syntax:<br><br>*/folder-name-1/<.../folder-name-n/>*<br>    specifies the location of the library.<br><br>*library-name*<br>    specifies the name of the library. | ■ `/Data/Libraries/ Customer Data Library(Library)`<br><br>■ `/MyCustomRepository /More Data/ OracleData(Library)` |

# Macro Variables That Are Generated from Prompts

## Macro Variable Generation and Assignment

One or more global macro variables is automatically generated for each prompt when the prompt is executed at run time. The values that are specified for the prompts at run time are assigned to the generated macro variables.

One or more macro variables can be generated from both single-value prompts and multi-value prompts. A prompt can have single or multiple values, depending on what you select for the **Number of values** field on the **Prompt Type and Values** tab in the New Prompt or Edit Prompt dialog box in SAS Management Console. The following list describes the macro variables that can be generated. *PROMPT-NAME* is used to represent the name of the prompt.

■ A base macro variable is generated for every type of prompt except range prompts. The name of the prompt is used as the name of the base macro variable.

.............................................................................

**Note:** Macro variable names can be no more than 32 characters long. Ensure that you consider this restriction when you name the prompt.

.............................................................................

■ For all multi-value prompts, the following macro variables are generated. Suffixes such as _COUNT or a numeric value are appended to the prompt name to create unique names for these macro variables.

*PROMPT-NAME*_COUNT
    The value of this macro variable is the number of values that are specified for the prompt. If no value is specified for a multi-value prompt, then *PROMPT-NAME*_COUNT is set to 0.

*PROMPT-NAME*0

> The value of this macro variable is the same as the value of *PROMPT-NAME*_COUNT. If no value or only one value is specified for a multi-value prompt, then this macro variable is not generated.

*PROMPT-NAMEn*

> If a multi-value prompt has more than one value specified for it, then each value is stored in a macro variable with the name *PROMPT-NAMEn* where *n* is the ordinal number of the value in the list of prompt values. The value of *PROMPT-NAME*1 is the same as the value of the base macro variable. If only one value is specified for a multi-value prompt, then no *PROMPT-NAMEn* macro variables are generated.

....................................................................................................

**Note:** Macro variable names can be no more than 32 characters long. Ensure that you consider this restriction and the length of the suffix when you name the prompt. The length of the suffix is included as part of the 32 characters.

....................................................................................................

■ If any of the following special values are specified for a prompt, then the corresponding base macro variable or *PROMPT-NAMEn* macro variable is assigned a value as follows:

*Table A5.2   Generated Macro Variables for Special Values*

| Special Value | Macro Variable Value |
| --- | --- |
| `(all possible values)` | `_ALL_VALUES_` |
| `(missing values)` for numeric, date, time, and timestamp prompts | `.` |
| `(missing values)` for character prompts | (a single space) |

■ Additional macro variables are generated for certain types of prompts. See the for a list of these macro variables. Suffixes such as _REL, _MIN, and _MAX are appended to the prompt name to create unique names for these macro variables. The following list describes the macro-variables that can be generated. *SUFFIX* is used to represent the various suffixes.

*PROMPT-NAME_SUFFIX*

> This macro variable is generated for both single-value and multi-value prompts.

*PROMPT-NAME_SUFFIXn*

> These macro variables are generated when a multi-value prompt has more than one value specified for it. The *n* is the ordinal number of the value in the list of prompt values. The value of *PROMPT-NAME_SUFFIX*1 is the same as the value of *PROMPT-NAME_SUFFIX*. If only one value is specified for a multi-value prompt, then no *PROMPT-NAME_SUFFIXn* macro variables are generated.

> **Note:** Macro variable names can be no more than 32 characters long. Ensure that you consider this restriction and the length of the suffix when you name the prompt. The length of the suffix is included as part of the 32 characters.

■ If no value is specified for a prompt, then an empty base macro variable is generated. For range prompts, which do not have base macro variables, the *PROMPT-NAME*_MIN and *PROMPT-NAME*_MAX macro variables are empty.

# Example: Single Macro Variable Generation

The following example shows the macro variable that is generated for a single-value text prompt.

*Table A5.3    Generated Macro Variables for a Single-Value Text Prompt*

| Prompt Name | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|
| MYPROMPT | `Hello World!` | MYPROMPT | `Hello World!` |

# Examples: Multiple Macro Variable Generation

The following example shows the macro variables that are generated for a single-value time prompt.

*Table A5.4    Generated Macro Variables for a Single-Value Time Prompt*

| Prompt Name | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|
| MYTIME | `09:59:55 AM` | MYTIME | `9:59:55` |
| | | MYTIME_LABEL | `09:59:55 AM` |

In the preceding example, two macro variables were generated for the single prompt value. If the prompt value had been a relative time value (such as `Yesterday`), then a third macro variable named MYTIME_REL would have been generated.

The following example shows the macro variables that are generated for a multi-value text prompt.

***Table A5.5*** *Generated Macro Variables for a Multi-value Text Prompt*

| Prompt Name | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|
| RESPONSE | Yes | RESPONSE | Yes |
| | No | RESPONSE_COUNT | 4 |
| | Undecided | | |
| | (missing values) | | |
| | | RESPONSE0 | 4 |
| | | RESPONSE1 | Yes |
| | | RESPONSE2 | No |
| | | RESPONSE3 | Undecided |
| | | RESPONSE4 | |

In the preceding example, seven macro variables were generated for the four prompt values. The macro variables RESPONSE and RESPONSE1 both contain the first prompt value. The macro variables RESPONSE_COUNT and RESPONSE0 both contain the number of values that were specified for the prompt. The macro variables RESPONSE2 and RESPONSE3 contain the second and third prompt values, respectively. RESPONSE4 contains a single blank space, which represents the special value `(missing values)`.

The following example shows the macro variables that are generated for a multi-value date prompt.

***Table A5.6*** *Generated Macro Variables for a Multi-value Date Prompt*

| Prompt Name | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|
| MYDATE | Today | MYDATE_COUNT | 3 |
| | Tomorrow | MYDATE0 | 3 |
| | September 04, 2008 | MYDATE | 02Sep2008 |
| | | MYDATE1 | 02Sep2008 |
| | | MYDATE2 | 03Sep2008 |
| | | MYDATE3 | 04Sep2008 |
| | | MYDATE_LABEL | Today (September 02, 2010) |

| Prompt Name | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|
| | | MYDATE_LABEL1 | `Today (September 02, 2010)` |
| | | MYDATE_LABEL2 | `Tomorrow (September 03, 2010)` |
| | | MYDATE_LABEL3 | `September 04, 2008` |
| | | MYDATE_REL | `D0D` |
| | | MYDATE_REL1 | `D0D` |
| | | MYDATE_REL2 | `D1D` |

In the preceding example, 13 macro variables were generated for the three prompt values.

■ The macro variables MYDATE_COUNT and MYDATE0 contain the number of values that were specified for the prompt.

■ The macro variables MYDATE and MYDATE1 contain the specific date that the first relative date (`Today`) resolves to. The macro variable MYDATE2 contains the specific date that the second relative date (`Tomorrow`) resolves to. The macro variable MYDATE3 contains the third prompt value.

■ The macro variables MYDATE_LABEL, MYDATE_LABEL1, and MYDATE_LABEL2 contain the relative dates (and their respective resolved dates) that were specified for the prompt. MYDATE_LABEL3 contains the long form of the specific date that was specified for the prompt.

■ The macro variables MYDATE_REL and MYDATE_REL1 contain the internal representation of `Today`. The macro variable MYDATE_REL2 contains the internal representation of `Tomorrow`.

# Quick Reference

The following table lists, by prompt type, the macro variables that are generated and how their values are determined. Examples of the generated macro variables are also provided.

**Note:** If your application or software feature enables you to create custom types of prompts, then the application or software feature determines which macro variables are generated for those prompts. For information about macro variables for custom types of prompts, see the documentation for your application or software feature.

*Table A5.7*  *Generated Macro Variables by Prompt Type for Prompt MYPROMPT*

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| Text | The *PROMPT-NAME* macro variable contains the value of the prompt. No additional macro variables are generated for single-value prompts. For more information about the macro variables that are generated for multi-value prompts, see "Macro Variable Generation and Assignment" on page 216. | `Hello World!` | MYPROMPT | `Hello World!` |
| Numeric | The *PROMPT-NAME* macro variable contains the value of the prompt. No additional macro variables are generated for single-value prompts. For more information about the macro variables that are generated for multi-value prompts, see "Macro Variable Generation and Assignment" on page 216. | `12` | MYPROMPT | `12` |
| Text range, Numeric range | A base macro variable is not generated for range prompts. | | | |
| | The *PROMPT-NAME*_MIN macro variable contains the lower boundary of the specified prompt range. | From: `23` To: `45` | MYPROMPT_MIN | `23` |
| | The *PROMPT-NAME*_MAX macro variable contains the upper boundary of the specified prompt range. | From: `23` To: `45` | MYPROMPT_MAX | `45` |
| Date (Day) | The *PROMPT-NAME* macro variable contains the value of the prompt in the format *ddmmmyyyy*.[1]  Here is an explanation of the syntax:  *dd*  specifies a two-digit integer that represents the day of the month. | `April 04, 2008` | MYPROMPT | `04Apr2008` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | *mmm* specifies the first three letters of the month. *yyyy* specifies a four-digit integer that represents the year. For more information about the macro variables that are generated for multi-value prompts, see "Macro Variable Generation and Assignment" on page 216. | | | |
| | The *PROMPT-NAME*_LABEL macro variable contains one of the following values: ■ for relative date values, the relative date with the resolved date in parentheses (for example, `Tomorrow (April 04, 2008)`). ■ for specific date values, the date in the format *month-name dd*, *yyyy*. Here is an explanation of the syntax: *month-name* specifies the full name of the month. *dd* specifies a two-digit integer that represents the day of the month. *yyyy* specifies a four-digit integer that represents the year. For more information about the macro variables that are generated for multi-value prompts, see "Macro Variable Generation and Assignment" on page 216. | `April 04, 2008` | MYPROMPT_LABEL | `April 04, 2008` |
| Date (Day) (cont'd.) | The *PROMPT-NAME*_REL macro variable contains the internal representation of the relative date that is specified for the prompt. This macro | `Current day of last year` | MYPROMPT_REL | `D0D-1Y` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | variable is generated only when the prompt value is a relative date.<br><br>For more information about the macro variables that are generated for multi-value prompts, see "Macro Variable Generation and Assignment" on page 216. | | | |
| Date (Week, Month, Quarter, Year) | The *PROMPT-NAME* macro variable contains the first day of the week, month, quarter, or year[2]<br><br>that is specified for the prompt. The format of the macro variable value is *ddmmmyyyy*.[1]<br><br>Here is an explanation of the syntax:<br><br>*dd*<br>    specifies a two-digit integer that represents the day of the month.<br><br>*mmm*<br>    specifies the first three letters of the month.<br><br>*yyyy*<br>    specifies a four-digit integer that represents the year.<br><br>For more information about the macro variables that are generated for multi-value prompts, see "Macro Variable Generation and Assignment" on page 216. | `Week 36 2008` | MYPROMPT | `01Sep2008` |
| | The *PROMPT-NAME*_END macro variable contains the last day of the week, month, quarter, or year[2]<br><br>that is specified for the prompt. See the above base macro variable entry for the format that is used.[1]<br><br>For more information about the macro variables that are generated for multi-value prompts, see "Macro Variable | `Week 36 2008` | MYPROMPT_END | `07Sep2008` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | Generation and Assignment" on page 216. | | | |
| Date (Week, Month, Quarter, Year) (cont'd.) | The *PROMPT-NAME*_LABEL macro variable contains one of the following week, month, quarter, or year[2] values:<br><br>■ for relative date values, the relative date with the resolved date in parentheses (for example, `Current week (Week 36 2008)`).<br><br>■ for specific date values, the date in the following formats:<br><br>□ Week *ww*, *yyyy*<br><br>□ *month-name yyyy*<br><br>□ *quarter-name* quarter *yyyy*<br><br>□ *yyyy*<br><br>Here is an explanation of the syntax:<br><br>*ww*<br>    specifies a two-digit integer that represents the week of the year.<br><br>*month-name*<br>    specifies the full name of the month.<br><br>*quarter-name*<br>    specifies the quarter of the year (`1st`, `2nd`, `3rd`, or `4th`).<br><br>*yyyy*<br>    specifies a four-digit integer that represents the year.<br><br>For more information about the macro variables that are generated for multi-value prompts, see "Macro Variable Generation and Assignment" on page 216. | `Week 36 2008` | MYPROMPT_LABEL | `Week 36 2008` |
| | The *PROMPT-NAME*_REL macro variable contains the | `Current week of last year` | MYPROMPT_REL | `W0W-1Y` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | internal representation of the relative date that is specified for the prompt. This macro variable is generated only when the prompt value is a relative date. For more information about the macro variables that are generated for multi-value prompts, see "Macro Variable Generation and Assignment" on page 216. | | | |
| Date range (Day) | A base macro variable is not generated for range prompts. | | | |
| | The *PROMPT-NAME*_MIN macro variable contains the lower boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.[1] | From: `August 31, 2008` To: `September 06, 2008` | MYPROMPT_MIN | `31Aug2008` |
| | The *PROMPT-NAME*_MAX macro variable contains the upper boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.[1] | From: `August 31, 2008` To: `September 06, 2008` | MYPROMPT_MAX | `06Sep2008` |
| | The *PROMPT-NAME*_MIN_LABEL macro variable contains the lower boundary of the specified prompt range in the format that is specified for the *PROMPT-NAME*_LABEL macro variable entry for date (day) prompts. | From: `August 31, 2008` To: `September 06, 2008` | MYPROMPT_MIN_LAB EL | `August 31, 2008` |
| | The *PROMPT-NAME*_MAX_LABEL macro variable contains the upper boundary of the specified prompt range in the format that is specified for the *PROMPT-NAME*_LABEL macro variable entry for date (day) prompts. | From: `August 31, 2008` To: `September 06, 2008` | MYPROMPT_MAX_LA BEL | `September 06, 2008` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | The *PROMPT-NAME*_MIN_REL macro variable contains the internal representation of the relative date that is specified for the lower boundary. This macro variable is generated only when the prompt value is a relative date. | From: `Today` To: `Current day of next month` | MYPROMPT_MIN_REL | `D0D` |
| | The *PROMPT-NAME*_MAX_REL macro variable contains the internal representation of the relative date that is specified for the upper boundary. This macro variable is generated only when the prompt value is a relative date. | From: `Today` To: `Current day of next month` | MYPROMPT_MAX_REL | `DOD1M` |
| Date range (Week, Month, Quarter, Year) | A base macro variable is not generated for range prompts. | | | |
| | The *PROMPT-NAME*_MIN macro variable contains the first day of the lower boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.[1] | From: `September 2008` To: `June 2009` | MYPROMPT_MIN | `01Sep2008` |
| | The *PROMPT-NAME*_MAX macro variable contains the first day of the upper boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.[1] | From: `September 2008` To: `June 2009` | MYPROMPT_MAX | `01Jun2009` |
| | The *PROMPT-NAME*_MIN_LABEL macro variable contains the lower boundary of the specified prompt range in the formats that are specified for the *PROMPT-NAME*_LABEL macro variable entry for date (week, month, quarter, year) prompts. | From: `September 2008` To: `June 2009` | MYPROMPT_MIN_LABEL | `September 2008` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | The *PROMPT-NAME*_MAX_LABEL macro variable contains the upper boundary of the specified prompt range in the formats that are specified for the *PROMPT-NAME*_LABEL macro variable entry for date (week, month, quarter, year) prompts. | From: `September 2008`<br><br>To: `June 2009` | MYPROMPT_MAX_LABEL | `June 2009` |
| | The *PROMPT-NAME*_MIN_END macro variable contains the last day of the lower boundary of the specified prompt range . See the base macro variable entry for date prompts for the format that is used.[1] | From: `September 2008`<br><br>To: `June 2009` | MYPROMPT_MIN_END | `30Sep2008` |
| | The *PROMPT-NAME*_MAX_END macro variable contains the last day of the upper boundary of the specified prompt range. See the base macro variable entry for date prompts for the format that is used.[1] | From: `September 2008`<br><br>To: `June 2009` | MYPROMPT_MAX_END | `30Jun2009` |
| | The *PROMPT-NAME*_MIN_REL macro variable contains the internal representation of the relative date that is specified for the lower boundary. This macro variable is generated only when the prompt value is a relative date. | From: `Current week`<br><br>To: `Current week of next year` | MYPROMPT_MIN_REL | `W0W` |
| Date range (Week, Month, Quarter, Year) (cont'd.) | The *PROMPT-NAME*_MAX_REL macro variable contains the internal representation of the relative date that is specified for the upper boundary. This macro variable is generated only when the prompt value is a relative date. | From: `Current week`<br><br>To: `Current week of next year` | MYPROMPT_MAX_REL | `W0W1Y` |
| Time | The *PROMPT-NAME* macro variable contains the value of | `02:05:28 PM` | MYPROMPT | `14:05:28` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | the prompt in the format *hh*:*mm*:*ss*.[1]<br><br>Here is an explanation of the syntax:<br><br>*hh*<br>    specifies a one-digit or two-digit integer that represents the hour of a 24-hour day.<br><br>*mm*<br>    specifies a two-digit integer that represents the minute of the hour.<br><br>*ss*<br>    specifies a two-digit integer that represents the second of the minute. | | | |
| | The *PROMPT-NAME*_LABEL macro variable contains one of the following values:<br><br>▪ for relative time values, the relative time with the resolved time in parentheses (for example, `Current time (02:05)28 PM)`).<br><br>▪ for specific time values, the time in the format *hh*:*mm*:*ss* AM \| PM.<br><br>Here is an explanation of the syntax:<br><br>*hh*<br>    specifies a two-digit integer that represents the hour of a 12-hour day.<br><br>*mm*<br>    specifies a two-digit integer that represents the minute of the hour.<br><br>*ss*<br>    specifies a two-digit integer that represents the second of the minute.<br><br>AM or PM<br>    specifies either the time period 00:01–12:00 noon (AM) or the time period 12:01–12:00 midnight (PM). | `02:05:28 PM` | MYPROMPT_LABEL | `02:05:28 PM` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| Time (cont'd.) | The *PROMPT-NAME*_REL macro variable contains the internal representation of the relative time that is specified for the prompt. This macro variable is generated only when the prompt value is a relative time (such as `Next minute`). | `Next minute` | MYPROMPT_REL | `m1m` |
| Timestamp | The *PROMPT-NAME* macro variable contains the value of the prompt in the format *ddmmmyyyy hh*:*mm*:*ss*.[1]<br><br>Here is an explanation of the syntax:<br><br>*dd*<br>　　specifies a two-digit integer that represents the day of the month.<br><br>*mmm*<br>　　specifies the first three letters of the month.<br><br>*yyyy*<br>　　specifies a four-digit integer that represents the year.<br><br>*hh*<br>　　specifies a one-digit or two-digit integer that represents the hour of a 24-hour day.<br><br>*mm*<br>　　specifies a two-digit integer that represents the minute of the hour.<br><br>*ss*<br>　　specifies a two-digit integer that represents the second of the minute. | `September 02, 2008 02:07:03 PM` | MYPROMPT | `02Sep2008 14:07:03` |
| | The *PROMPT-NAME*_LABEL macro variable contains the value of the prompt in the format *month-name dd*, *yyyy hh*:*mm*:*ss* AM \| PM.<br><br>Here is an explanation of the syntax:<br><br>*month-name*<br>　　specifies the full name of the month. | `September 02, 2008 02:07:03 PM` | MYPROMPT_LABEL | `September 02, 2008 02:07:03 PM` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | *dd*<br>    specifies a two-digit integer that represents the day of the month.<br>*yyyy*<br>    specifies a four-digit integer that represents the year. | | | |
| Timestamp (cont'd.) | *hh*<br>    specifies a two-digit integer that represents the hour of a 12-hour day.<br>*mm*<br>    specifies a two-digit integer that represents the minute of the hour.<br>*ss*<br>    specifies a two-digit integer that represents the second of the minute.<br>AM or PM<br>    specifies either the time period 00:01–12:00 noon (AM) or the time period 12:01–12:00 midnight (PM). | | | |
| | The *PROMPT-NAME*_REL macro variable contains the internal representation of the relative timestamp that is specified for the prompt. This macro variable is generated only when the prompt value is a relative timestamp (such as `Current date and time`). | `Current date and time` | MYPROMPT_REL | `T0m` |
| Time range, Timestamp range | A base macro variable is not generated for range prompts. | | | |
| | The *PROMPT-NAME*_MIN macro variable contains the lower boundary of the specified prompt range. See the base macro variable entries for time and timestamp prompts for the formats that are used.[1] | From:<br>`01:13:04 AM`<br><br>To: `02:13:14 PM` | MYPROMPT_MIN | `1:13:04` |
| | The *PROMPT-NAME*_MAX macro variable contains the upper boundary of the specified prompt range. See | From:<br>`01:13:04 AM`<br><br>To: `02:13:14 PM` | MYPROMPT_MAX | `14:13:14` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | the base macro variable entries for time and timestamp prompts for the formats that are used.[1] | | | |
| | The *PROMPT-NAME*_MIN_LABEL macro variable contains the lower boundary of the specified prompt range in the formats that are specified for the *PROMPT-NAME*_LABEL macro variables for time and timestamp prompts. | From: `October 29, 2008 10:12:12 AM` <br> To: `February 14, 2009 12:25:36 PM` | MYPROMPT_MIN_LABEL | `Oct 29, 2008 10:12:12 AM` |
| | The *PROMPT-NAME*_MAX_LABEL macro variable contains the lower boundary of the specified prompt range in the formats that are specified for the *PROMPT-NAME*_LABEL macro variables for time and timestamp prompts. | From: `October 29, 2008 10:12:12 AM` <br> To: `February 14, 2009 12:25:36 PM` | MYPROMPT_MAX_LABEL | `Feb 14, 2009 12:25:36 PM` |
| | The *PROMPT-NAME*_MIN_REL macro variable contains the internal representation of the relative time or timestamp that is specified for the lower boundary. This macro variable is generated only when the lower boundary is a relative time or timestamp (such as `Beginning of next hour`). | From: `Beginning of next hour` <br> To: `End of next hour` | MYPROMPT_MIN_REL | `t1HBH` (for time) <br> `T1HBH` (for timestamp) |
| Time range, Timestamp range (cont'd.) | The *PROMPT-NAME*_MAX_REL macro variable contains the internal representation of the relative time or timestamp that is specified for the upper boundary. This macro variable is generated only when the upper boundary is a relative time or timestamp (such as `End of next hour`). | From: `Beginning of next hour` <br> To: `End of next hour` | MYPROMPT_MAX_REL | `t1HEH` (for time) <br> `T1HEH` (for timestamp) |
| Color | The *PROMPT-NAME* macro variable contains the value of | `CXFF0000` (bright red) | MYPROMPT | `cxff0000` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | the prompt in the format cx*rrggbb*. Here is an explanation of the syntax: *rr* specifies the red component. *gg* specifies the green component. *bb* specifies the blue component. | | | |
| Data source | The *PROMPT-NAME* macro variable contains the value of the prompt in the format /*folder-name-1/<.../folder-name-n/>data-source-name*(*type*). Here is an explanation of the syntax: /*folder-name-1/<.../folder-name-n/>* specifies the location of the data source. *data-source-name* specifies the name of the data source. *type* specifies the type of data source (Table, InformationMap, or Cube). | `/Shared Data/Tables/ MYDATA(Table )` | MYPROMPT | `/Shared Data/Tables/ MYDATA(Table )` |
| | *PROMPT-NAME*_TYPE contains the type of the data source, represented by the following numbers: ▪ 1 represents a table ▪ 2 represents a cube ▪ 4 represents a relational information map ▪ 8 represents an OLAP information map | `/Shared Data/Tables/ MYDATA(Table )` | MYPROMPT_TYPE | `1` |
| File or directory | The *PROMPT-NAME* macro variable contains the value of the prompt in the format *directory-specification<filename>* | `C:\Documents and Settings \All Users` | MYPROMPT | `C:\Documents and Settings \All Users` |

| Prompt Type | Macro Variable | Prompt Value | Macro Variable Name | Macro Variable Value |
|---|---|---|---|---|
| | Here is an explanation of the syntax:<br><br>*directory-specification*<br>    specifies the location of the file or directory in the file system of the SAS Workspace Server that was specified for the prompt.<br><br>*filename*<br>    specifies the name of the file. This value is available only if the prompt is a file prompt. | `\Documents`<br>`\myfile.txt` | | `\Documents`<br>`\myfile.txt` |
| | The *PROMPT-NAME*_SERVER macro variable contains the name of the SAS server that was specified for the prompt. | `C:\Documents`<br>`and Settings`<br>`\All Users`<br>`\Documents`<br>`\myfile.txt` | MYPROMPT_SERVER | `SASApp -`<br>`Logical`<br>`Workspace`<br>`Server` |
| Data library | The *PROMPT-NAME* macro variable contains the libref of the library that is specified for the prompt. | `/Shared`<br>`Data/`<br>`Libraries/`<br>`SASHELP(Libr`<br>`ary)` | MYPROMPT | `SASHELP` |
| | The *PROMPT-NAME*_PATH macro variable contains the name and location of the library. The format of the macro variable value is */folder-name-1/<.../folder-name-n/>library-name*(Library). | `/Shared`<br>`Data/`<br>`Libraries/`<br>`SASHELP(Libr`<br>`ary)` | MYPROMPT_PATH | `/Shared`<br>`Data/`<br>`Libraries/`<br>`SASHELP(Libr`<br>`ary)` |

**1** Relative date, relative time, and relative timestamp values are resolved to specific date, time, and timestamp values, respectively, for this macro variable.

**2** All dates are based on the Gregorian calendar. Each year begins with the month of January, and each week begins on Monday and ends on Sunday, regardless of locale.