SAS® Output Delivery System: Procedures Guide

2021.1.6 - 2023.03*

* This document might apply to additional versions of the software. Open this document in SAS Help Center and click on the version in the banner to see all available versions.

SAS® Documentation
March 3, 2023

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2021. *SAS® Output Delivery System: Procedures Guide*. Cary, NC: SAS Institute Inc.

SAS® Output Delivery System: Procedures Guide

Copyright © 2021, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government’s rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

March 2023

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

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

v_002-P1:odsproc

Contents

PART 1 Introduction 1

Chapter 1 / Introduction 3
Overview of the Output Delivery System 3
Overview of How ODS Works 4

PART 2 Customizing Procedure Output 7

Chapter 2 / Working with General Procedure Output 9
Customized Output for an Output Object 9
Customizing Titles and Footnotes 10

Chapter 3 / Using Selection and Exclusion Lists 11
Overview 11
Dictionary 12

Chapter 4 / Other SAS Statements That Are Used with ODS Procedures 35
Dictionary 35

PART 3 The DOCUMENT Procedure 79

Chapter 5 / ODS DOCUMENT Statement 81
Dictionary 81

Chapter 6 / DOCUMENT Procedure 87
Overview: DOCUMENT Procedure 88
Concepts: DOCUMENT Procedure 89
Syntax: DOCUMENT Procedure 91
Usage: DOCUMENT Procedure 144
Examples: DOCUMENT Procedure 153

PART 4 The ODSLIST Procedure 197

Chapter 7 / ODSLIST Procedure 199
Overview: ODSLIST Procedure 200

Syntax: ODSLIST Procedure . . . 200
Usage: ODSLIST Procedure . . . 220
Examples: ODSLIST Procedure . . . 223

PART 5 The ODSTABLE Procedure 239

Chapter 8 / ODSTABLE Procedure . . . **241**
Overview: ODSTABLE Procedure . . . 242
Concepts: ODSTABLE Procedure . . . 242
Syntax: ODSTABLE Procedure . . . 244
Usage: ODSTABLE Procedure . . . 265
Examples: ODSTABLE Procedure . . . 269

PART 6 The ODSTEXT Procedure 287

Chapter 9 / ODSTEXT Procedure . . . **289**
Overview: ODSTEXT Procedure . . . 290
Syntax: ODSTEXT Procedure . . . 290
Usage: ODSTEXT Procedure . . . 312
Examples: ODSTEXT Procedure . . . 315

PART 7 The TEMPLATE Procedure 323

Chapter 10 / Overview . . . **325**
Introduction to the TEMPLATE Procedure . . . 325
Terminology: TEMPLATE Procedure . . . 326
The Backward Compatibility of ODS Templates . . . 327
Syntax . . . 328
Using the TEMPLATE Procedure . . . 329
PROC TEMPLATE Statements by Category . . . 336
Where to Go from Here . . . 337

Chapter 11 / TEMPLATE Procedure: Managing Template Stores . . . **339**
Overview: TEMPLATE Procedure: Managing Template Stores . . . 340
Concepts: TEMPLATE Procedure: Managing Template Stores . . . 341
Syntax: TEMPLATE Procedure: Managing Template Stores . . . 342
Examples: TEMPLATE Procedure: Managing Template Stores . . . 358

Chapter 12 / TEMPLATE Procedure: Creating Crosstabulation Table Templates . . . **365**
Overview: TEMPLATE Procedure: Creating Crosstabulation Table Templates . . . 366
Concepts: TEMPLATE Procedure: Creating Crosstabulation Table Templates . . . 370
Syntax: TEMPLATE Procedure: Creating Crosstabulation Table Templates . . . 371
Usage: TEMPLATE Procedure: Creating Crosstabulation Table Templates . . . 395
Examples: TEMPLATE Procedure: Creating Crosstabulation Table Templates . . . 397

Chapter 13 / TEMPLATE Procedure: Creating ODS Graphics 427
Introduction to the Graph Template Language 427
Syntax 430
Where to Go from Here 431

Chapter 14 / TEMPLATE Procedure: Creating a Style Template 433
Overview: TEMPLATE Procedure: Creating a Style Template 434
Concepts: TEMPLATE Procedure: Creating a Style Template 438
Syntax: TEMPLATE Procedure: Creating a Style Template 452
Usage: TEMPLATE Procedure: Creating a Style Template 465
Examples: TEMPLATE Procedure: Creating a Style Template 466

Chapter 15 / TEMPLATE Procedure: Creating Tabular Templates 515
Overview: TEMPLATE Procedure: Creating Tabular Templates 516
Concepts: TEMPLATE Procedure: Creating Tabular Templates 519
Syntax: TEMPLATE Procedure: Creating Tabular Templates 520
Usage: TEMPLATE Procedure: Creating Tabular Templates 547
Examples: TEMPLATE Procedure: Creating Tabular Templates 550

Chapter 16 / Tabular Attributes 585
Understanding Table Templates, Table Elements, and Table Attributes 585
Column Attributes 586
Header and Footer Attributes 603
Table Attributes 614

Chapter 17 / TEMPLATE Procedure: Creating Markup Language Tagsets 627
Overview: TEMPLATE Procedure: Creating Markup Language Tagsets 628
Concepts: TEMPLATE Procedure: Creating Markup Language Tagsets 629
Syntax: TEMPLATE Procedure: Creating Markup Language Tagsets 637
Usage: TEMPLATE Procedure: Creating Markup Language Tagsets 680
Examples: TEMPLATE Procedure: Creating Markup Language Tagsets 689

PART 8 ODS Styles Reference 713

Chapter 18 / Overview 715
Understanding Styles, Style Elements, and Style Attributes 715
Using Styles with Base SAS Procedures 721

Chapter 19 / Style Templates 723
Viewing ODS Styles Supplied by SAS 723
Program for Viewing Multiple Styles 725
Table of Suggested ODS Styles 726
ODS Styles Gallery 729

Chapter 20 / Style Elements 777
ODS Style Elements 777
Style Elements Affecting Template-Based Graphics 788
Style Elements Affecting Device-Based Graphics 797

Chapter 21 / Style Attributes 807
Overview 807
Style Attributes Tables 808

Style Attributes Detailed Information . . . 838
Style Attributes Values . . . 874

PART 9 Appendices 881

Appendix 1 / Output Object Table Names . . . **883**
ODS Table Names Produced by Base SAS Procedures . . . 883
ODS Table Names Produced by Base SAS High-Performance Procedures . . . 897
ODS Table Names Produced by Base SAS Statistical Procedures . . . 897
ODS Table Names Produced by SAS/STAT Procedures . . . 898
ODS Table Names Produced by SAS/STAT High-Performance Procedures . . . 902
ODS Table Names Produced by SAS Enterprise Miner High-Performance Procedures . . . 903
ODS Table Names Produced by SAS/ETS Procedures . . . 904
ODS Table Names Produced by SAS/ETS High-Performance Procedures . . . 906
ODS Table Names Produced by SAS/QC Procedures . . . 906
ODS Table Names Produced by SAS/OR Procedures . . . 907
ODS Table Names Produced by Graph Algorithms and Network Analysis Procedure . . . 908

Appendix 2 / Example Code . . . **909**
Creating the DistrData Data Set . . . 909
Creating the Univ ODS Document . . . 909

Appendix 3 / Lua License . . . **911**
Lua License . . . 911

PART 1

Introduction

Chapter 1
Introduction 3

1

Introduction

Overview of the Output Delivery System 3

Overview of How ODS Works 4
Components of SAS Output 4
Features of ODS 5

Overview of the Output Delivery System

The Output Delivery System (ODS) gives you great flexibility in generating, storing, and reproducing SAS procedures and DATA step output, with a wide range of formatting options.

By default, ODS output is formatted according to instructions that a PROC step or DATA step defines. However, ODS provides ways for you to customize the output. You can customize the style for all of your output, or you can customize a single table or graph.

You can use ODS to accomplish the following tasks:

Create reports for viewers or browsers.
With ODS, you can use ODS destination statements to create output in popular formats such as HTML, PDF, and RTF. For example, you can use the ODS PDF statement to create PDF files for viewing with Adobe Acrobat or for printing. You can use the ODS EPUB statement to create output for e-book readers. The ODS RTF and ODS WORD statements creates output for Microsoft Word. For complete documentation on the ODS destination statements, see "Dictionary of ODS Language Statements" in *SAS Output Delivery System: User's Guide*.

Customize the report contents.
ODS enables you to modify the contents of your output. With ODS, you can embed graphics, select specific cell contents to display, and create embedded links in tables and graphs. You can select specific tables or graphs from procedure output to be printed or you can exclude them. You can create SAS data sets directly from tables or graphics.

Customize the presentation.

ODS enables you to change the appearance of your output. You can change the colors, fonts, and borders of your output. You can customize the layout, format, headers, and style. You can add images and embedded URLs.

Create more accessible SAS output.

The ODS HTML5 and ODS PDF destinations are the recommended destinations for creating SAS output that is accessible to the broadest audience. For information, see *Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics*.

Overview of How ODS Works

Components of SAS Output

The PROC or DATA step supplies data and the name of the template that contains the formatting instructions. ODS formats the output. You can use ODS to format output from individual procedures and from the DATA step in many different forms.

The following figure shows how SAS produces ODS output.

Figure 1.1 *ODS Processing: What Goes in and What Comes Out*

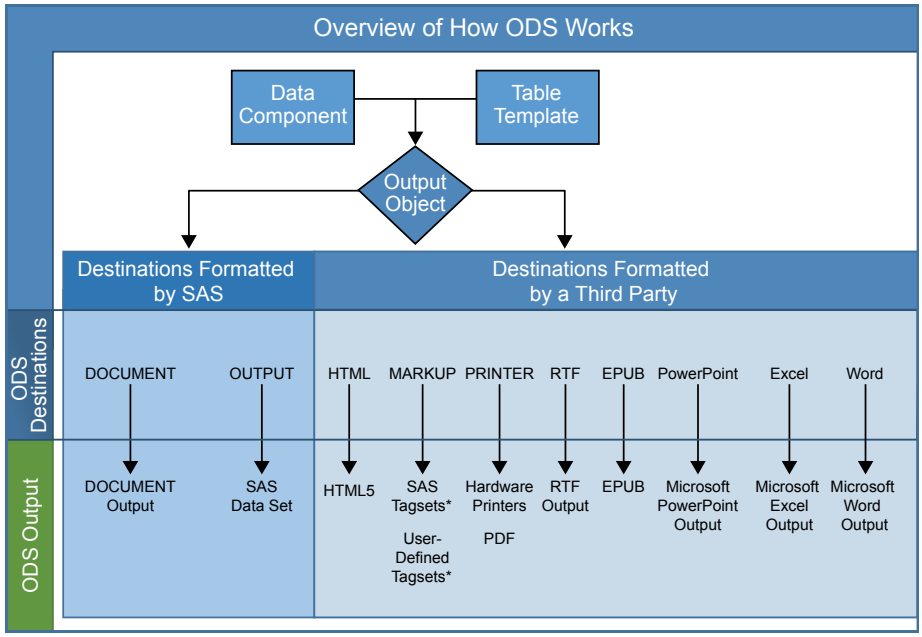


Table 1.1 ** List of Tagsets That SAS Supplies and Supports*

CSV	CSVALL	CSVBYLINE

EXCELXP	TAGSETS.RTF
DEFAULT	

Table 1.2 ** Additional Diagnostic Tagsets That SAS Supports*

EVENT_MAP	NAMEDHTML	SHORT_MAP	STYLE_DISPLAY
STYLE_POPUP	TEXT_MAP		

Note: Additional tagsets can be found at http://support.sas.com/rnd/base/ods/odsmarkup/ .

Features of ODS

ODS delivers output in a variety of formats, and makes the formatted output easy to access.

Important features of ODS include the following:

- ODS combines data with table and graph templates to produce one or more output objects. These objects can be sent to any or all ODS destinations. The currently available ODS destinations can produce the following types of output:
  - an output data set
  - an ODS document that contains a hierarchy file of the output objects
  - output that is formatted for a high-resolution printer such as PDF
  - output that is formatted in various markup languages such as HTML
  - RTF and Word output that is formatted for use with Microsoft Word
  - PowerPoint output that is formatted for use with Microsoft PowerPoint
  - output that is formatted for EPUB e-books
- ODS provides table and graph templates that define the structure of the output from SAS procedures and from the DATA step. You can customize the output by modifying these templates or by creating your own.
- ODS provides a way for you to choose individual output objects to send to ODS destinations. Most SAS procedures produce multiple output objects. You can easily create HTML output, an output data set, or printer output from any or all output objects. You can send different output objects to different destinations.
- Because formatting is centralized in ODS, the addition of a new ODS destination does not affect any procedures or the DATA step. As future destinations are added to ODS, they will automatically become available to the DATA step and all procedures that support ODS.
- With ODS, you can produce output for numerous destinations from a single program, but you do not need to maintain separate programs for each

destination. This feature saves you time and system resources by enabling you to produce multiple types of output with a single run of your procedure or data query.

PART 2

Customizing Procedure Output

Chapter 2
Working with General Procedure Output **9**

Chapter 3
Using Selection and Exclusion Lists **11**

Chapter 4
Other SAS Statements That Are Used with ODS Procedures **35**

2

Working with General Procedure Output

Customized Output for an Output Object 9
Customized Output for an Output Object 9
Customizing Titles and Footnotes 10
Customizing Titles and Footnotes 10

Customized Output for an Output Object

Customized Output for an Output Object

For a procedure, the name of the table template that is used for an output object comes from the procedure code.

To find out which table templates a procedure or the DATA step uses for the output objects, you must look at a trace record. To produce a trace record in your SAS log, submit the following SAS statements:

```
ods trace on;
your-proc-or-DATA-step
ods trace off;
```

A few procedures do not produce ODS output. If you produce a trace record for one of these procedures, no template appears in the trace record. Most procedures use one or more templates to produce their output. Each template appears in the trace record produced in the log.

For a detailed explanation of the trace record, see the “ODS TRACE Statement” on page 70.

You can use PROC TEMPLATE to modify templates. When a procedure or DATA step uses a table template, it uses the elements that are defined or referenced in its

table template. In most cases, the only way to modify the appearance of your output is by modifying one or more templates, which can include table, graph, style, column, and other templates.

Note: Three Base SAS procedures, PROC PRINT, PROC REPORT, and PROC TABULATE, do provide a way for you to access table elements from the procedure step itself. Accessing the table elements enables you to customize your report. For more information about these procedures, see *Base SAS Procedures Guide*.

Customizing Titles and Footnotes

Customizing Titles and Footnotes

You can use the global TITLE and FOOTNOTE statements to enhance the readability of any report. These statements have options that enable you to customize the style of the titles and footnotes. Examples of these style options are BOLD, COLOR=, and FONT=. Because these options control only the presentation of the titles and footnotes, they have no effect on titles or footnotes that go to the LISTING or OUTPUT destination. For a complete list of style options, detailed information about the style options and example code, see the TITLE statement and the FOOTNOTE statement in *SAS DATA Step Statements: Reference*.

When used with graphics, you can choose whether to render the titles and footnotes as part of the body of the document or as part of the graphics image. Where the titles and footnotes are rendered determines how you control the font, size, and color of the titles and footnotes text. For details about this ODS and SAS/GRAPH interaction, see Controlling Titles and Footnotes . For information about adding titles and footnotes to ODS Graphics, see “Adding Titles, Footnotes, and Text Entries to Your Graph” in *SAS Graph Template Language: User’s Guide*.

3

Using Selection and Exclusion Lists

Overview 11

Dictionary 12
ODS EXCLUDE Statement 12
ODS SELECT Statement 20

Overview

For each ODS destination, ODS maintains either a selection list or an exclusion list of output objects. You can use the default output objects selected or excluded for each destination or you can specify which output object you want to produce by selecting or excluding them from a list.

A selection list is a list of output objects that are sent to an ODS destination. An exclusion list is a list of output objects that are excluded from an ODS destination. ODS also maintains an overall selection or exclusion list of output objects. By checking the destination-specific lists and the overall list, ODS determines what output objects to produce. These lists can be modified by using the ODS SELECT statement and the ODS EXCLUDE statement.

> **TIP** You can maintain a selection list for one destination and an exclusion list for another. However, the results are less complicated if you maintain the same types of lists for all the destinations to which you route output.

You can view the contents of the exclusion and selection lists by using the ODS SHOW statement. The current selection list is written to the SAS log.

EXCLUDE ALL is the default setting for the ODS OUTPUT destination. SELECT ALL is the default setting for all other destinations. To change the default selection and exclusion lists, use the ODS SELECT or ODS EXCLUDE statements or use the

exclude and select actions that are available for some of the ODS statements. However, to set the exclusion list for the OUTPUT destination to something other than the default, use the "ODS OUTPUT Statement" in *SAS Output Delivery System: User's Guide*. For a list of ODS output destinations and explanations of each, see "Understanding ODS Destinations" in *SAS Output Delivery System: User's Guide*.

In order to view output objects that are selected or excluded from your program, use the ODS TRACE statement. The ODS TRACE statement prints the output objects that are selected and excluded and puts the information in a trace record that is written to the SAS log. The trace provides the path, the label, and other information about output objects that are selected and excluded. For complete documentation about viewing and selecting output objects, see the "ODS SELECT Statement" on page 20, the "ODS EXCLUDE Statement" on page 12, and the "ODS TRACE Statement" on page 70.

Dictionary

ODS EXCLUDE Statement

Specifies output objects to exclude from ODS destinations.

Category: ODS: Output Control

Syntax

ODS <*ODS-destination*> **EXCLUDE** *exclusion(s)* | ALL | NONE;

Required Arguments

exclusion(s)
specifies one or more output objects to add to an exclusion list.

By default, ODS automatically modifies exclusion lists at the end of a DATA step that uses ODS, or at the end of a procedure step. For information about modifying these lists, see "Selection and Exclusion Lists" in *SAS Output Delivery System: User's Guide* .

Each *exclusion* has the following form.

output-object <(PERSIST)>

output-object
specifies one or more output objects to exclude. To specify an output object, you need to know which output objects your SAS program produces. The ODS TRACE statement writes to the SAS log a trace record that includes the path, the label, and other information about each output object that is produced. You can specify an output object in any of the following ways:

- a full path. For example, the following is the full path of the output object:

```
Univariate.City_Pop_90.TestsForLocation
```

- a partial path. A partial path consists of any part of the full path that begins immediately after a period (.) and continues to the end of the full path. For example, suppose the full path is the following:

```
Univariate.City_Pop_90.TestsForLocation
```

  Then the partial paths are as follows:

```
City_Pop_90.TestsForLocation
TestsForLocation
```

- a label that is enclosed in quotation marks.

  For example:

```
"The UNIVARIATE Procedure" "Tests for Location"
```

- a label path. For example, the following is the label path for the output object:

```
"The UNIVARIATE Procedure"."CityPop_90"."Tests For Location"
```

  Note: The trace record shows the label path only if you specify the LABEL option in the ODS TRACE statement.

- a partial label path. A partial label path consists of any part of the label that begins immediately after a period (.) and continues to the end of the label. For example, suppose the label path is the following:

```
"The UNIVARIATE Procedure"."CityPop_90"."Tests For Location"
```

  Then the partial label paths are as follows:

```
"CityPop_90"."Tests For Location"
"Tests For Location"
```

- a mixture of labels and paths.
- any of the partial path specifications, followed by a number sign (#) and a number. For example, `TestsForLocation#3` refers to the third output object that is named `TestsForLocation`.

See "ODS TRACE Statement" on page 70.

(PERSIST)

keeps the *output-object* that precedes the *PERSIST* option in the exclusion list until you explicitly modify the list.

You can modify the list with any of the following ODS statements:

- any ODS SELECT statement
- ODS EXCLUDE NONE
- ODS EXCLUDE ALL
- an ODS EXCLUDE statement that applies to the same output object but does not specify PERSIST

This action is true even if the DATA or procedure step ends.

Requirement You must enclose PERSIST in parentheses.

ALL
specifies that ODS does not send any output objects to the open destination.

Alias DEFAULT

Interaction If you specify ALL without specifying a destination, ODS sets the overall list to EXCLUDE ALL and sets all other lists to their defaults.

Tips Using ODS EXCLUDE ALL is different from closing a destination. The destination remains open, but no output objects are sent to it.

To temporarily suspend a destination, use ODS SELECT NONE. Use ODS SELECT ALL when you want to resume sending output to the suspended destination.

NONE
specifies that ODS send all of the output objects to the open destination.

Interaction If you specify the NONE argument without specifying a destination, ODS sets the overall list to EXCLUDE NONE and sets all other lists to their defaults.

Tips ODS EXCLUDE NONE has the same effect as ODS SELECT ALL.

To temporarily suspend a destination, use ODS SELECT NONE. Use ODS SELECT ALL when you want to resume sending output to the suspended destination.

Optional Arguments

NOWARN
suppresses the warning that an output object was requested but not created.

Requirement The NOWARN option must be enclosed in parentheses.

Interaction The NOWARN option cannot be used with the ALL option or the NONE option.

Example The ODS EXCLUDE statement in the following example specifies that no warning is created if the output object Summary is requested but not created.

```
ods exclude summary (nowarn);

proc contents data=sashelp.class;
run;
```

ODS-destination
specifies to which ODS destination's exclusion list to write, where *ODS-destination* can be any valid ODS destination. For a discussion of ODS destinations, see "Understanding ODS Destinations" in *SAS Output Delivery System: User's Guide*.

Default If you omit *ODS-destination*, ODS writes to the overall exclusion list.

Tip To set the exclusion list for the output destination to something other than the default, use the "ODS OUTPUT Statement" in *SAS Output Delivery System: User's Guide*.

WHERE=*where-expression*
excludes output objects that meet a particular condition. For example, the following statement excludes only output objects with the word "Histogram" in their name:

```
ods exclude where=(_name_ ?  'Histogram');
```

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands. *where-expression* has this form.

(*subsetting-variable* <*comparison-operator where-expression-n*>)

subsetting-variable
is a special type of WHERE expression operand used by SAS to help you find common values in items. For example, this EXCLUDE statement excludes only output objects with the path `City_Pop_90.TestsForLocation` :

```
ods exclude / where=(_path_ = 'City_Pop_90.TestsForLocation' );
```

subsetting-variable is one of the following:

LABEL
is the label of the output object.

LABELPATH
is the label path of the output object.

NAME
is the name of the output object.

PATH
is the name path of the output object.

operator
compares a variable with a value or with another variable. *operator* can be AND, OR NOT, OR, AND NOT, or a comparison operator.

The following table lists some comparison operators:

Table 3.1 *Examples of Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to

Symbol	Mnemonic Equivalent	Definition
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Details

Using Selection and Exclusion Lists

For each ODS destination, ODS maintains either a selection list or an exclusion list of output objects. You can use the default output objects selected or excluded for each destination or you can specify which output object you want to produce by selecting or excluding them from a list.

A selection list is a list of output objects that are sent to an ODS destination. An exclusion list is a list of output objects that are excluded from an ODS destination. ODS also maintains an overall selection or exclusion list of output objects. By checking the destination-specific lists and the overall list, ODS determines what output objects to produce. These lists can be modified by using the ODS SELECT statement and the ODS EXCLUDE statement.

TIP You can maintain a selection list for one destination and an exclusion list for another. However, the results are less complicated if you maintain the same types of lists for all the destinations to which you route output.

You can view the contents of the exclusion and selection lists by using the ODS SHOW statement. The current selection list is written to the SAS log.

EXCLUDE ALL is the default setting for the ODS OUTPUT destination. SELECT ALL is the default setting for all other destinations. To change the default selection and exclusion lists, use the ODS SELECT or ODS EXCLUDE statements or use the exclude and select actions that are available for some of the ODS statements. However, to set the exclusion list for the OUTPUT destination to something other than the default, use the "ODS OUTPUT Statement" in *SAS Output Delivery System: User's Guide*. For a list of ODS output destinations and explanations of each, see "Understanding ODS Destinations" in *SAS Output Delivery System: User's Guide*.

In order to view output objects that are selected or excluded from your program, use the ODS TRACE statement. The ODS TRACE statement prints the output objects that are selected and excluded and puts the information in a trace record that is written to the SAS log. The trace provides the path, the label, and other information about output objects that are selected and excluded. For complete documentation about viewing and selecting output objects, see the "ODS SELECT Statement" on page 20, the "ODS EXCLUDE Statement" on page 12, and the "ODS TRACE Statement" on page 70.

Example: Conditionally Excluding Output Objects and Sending Them to Different Output Destinations

Features:

ODS EXCLUDE statement
- Options: ODS-Destination, WHERE=

ODS HTML5 statement options
- FILE=
- TEXT=

ODS PDF statement options
- TEXT=
- STARTPAGE=

PROC UNIVARIATE

Program

```
options nodate;
   data BPressure;
      length PatientID $2;
      input PatientID $ Systolic Diastolic @@;
      datalines;
   CK 120 50  SS 96  60 FR 100 70
   CP 120 75  BL 140 90 ES 120 70
   CP 165 110 JI 110 40 MC 119 66
   FC 125 76  RW 133 60 KD 108 54
   DS 110 50  JW 130 80 BH 120 65
   JW 134 80  SB 118 76 NS 122 78
   GS 122 70  AB 122 78 EC 112 62
   HH 122 82
   ;
   run;

ods html5 text='Systolic Blood Pressure'
          file='your-file-path/Systolic-body.html';

ods pdf file='your-file-path/Diastolic.pdf'
   text='Diastolic Blood Pressure'
   startpage=no;


ods html5 exclude where=(_path_ ? "Diastolic" ) ;
ods pdf exclude where=(_path_ ? "Systolic" ) ;

proc univariate data=BPressure;
   var Systolic Diastolic;
run;

ods html5 close;

ods pdf close;
```

Program Description

Create the BPressure data set.

```
options nodate;
   data BPressure;
      length PatientID $2;
      input PatientID $ Systolic Diastolic @@;
      datalines;
   CK 120 50  SS 96  60 FR 100 70
   CP 120 75  BL 140 90 ES 120 70
   CP 165 110 JI 110 40 MC 119 66
   FC 125 76  RW 133 60 KD 108 54
   DS 110 50  JW 130 80 BH 120 65
   JW 134 80  SB 118 76 NS 122 78
   GS 122 70  AB 122 78 EC 112 62
   HH 122 82
   ;
   run;
```

Create HTML5 output and add text.

```
ods html5 text='Systolic Blood Pressure'
          file='your-file-path/Systolic-body.html';
```

Create PDF output and add text.

```
ods pdf file='your-file-path/Diastolic.pdf'
   text='Diastolic Blood Pressure'
   startpage=no;
```

Exclude output objects from different output destinations. The first ODS EXCLUDE statement excludes from the HTML5 destination output objects that have 'Diastolic' in the pathname. The second ODS EXCLUDE statement excludes from the PDF destination output objects that have 'Systolic' in the pathname.

```
ods html5 exclude where=(_path_ ? "Diastolic" ) ;
ods pdf exclude where=(_path_ ? "Systolic" ) ;
```

Create the output objects. As PROC UNIVARIATE sends each output object to the Output Delivery System, ODS does not send the output objects from PROC UNIVARIATE that match the items in the exclusion list to the open destinations.

```
proc univariate data=BPressure;
   var Systolic Diastolic;
run;
```

Close the HTML5 destination. The ODS HTML5 CLOSE statement closes the HTML5 destination and all the files that are associated with it. If you do not close the destination, then you cannot view the HTML5 file specified by the FRAME attribute until you close your SAS session.

```
ods html5 close;
```

Close the PDF destination. This ODS PDF statement closes the PDF destination and all the files that are associated with it.

```
ods pdf close;
```

Output

Output 3.1 *Partial HTML5 Output with Systolic Output Objects*

Systolic Blood Pressure

Moments			
N	22	Sum Weights	22
Mean	121.272727	Sum Observations	2668
Std Deviation	14.283463	Variance	204.017316
Skewness	1.12787257	Kurtosis	3.39471271
Uncorrected SS	327840	Corrected SS	4284.36364
Coeff Variation	11.777968	Std Error Mean	3.04524455

Basic Statistical Measures			
Location		Variability	
Mean	121.2727	Std Deviation	14.28346
Median	120.0000	Variance	204.01732
Mode	120.0000	Range	69.00000
		Interquartile Range	13.00000

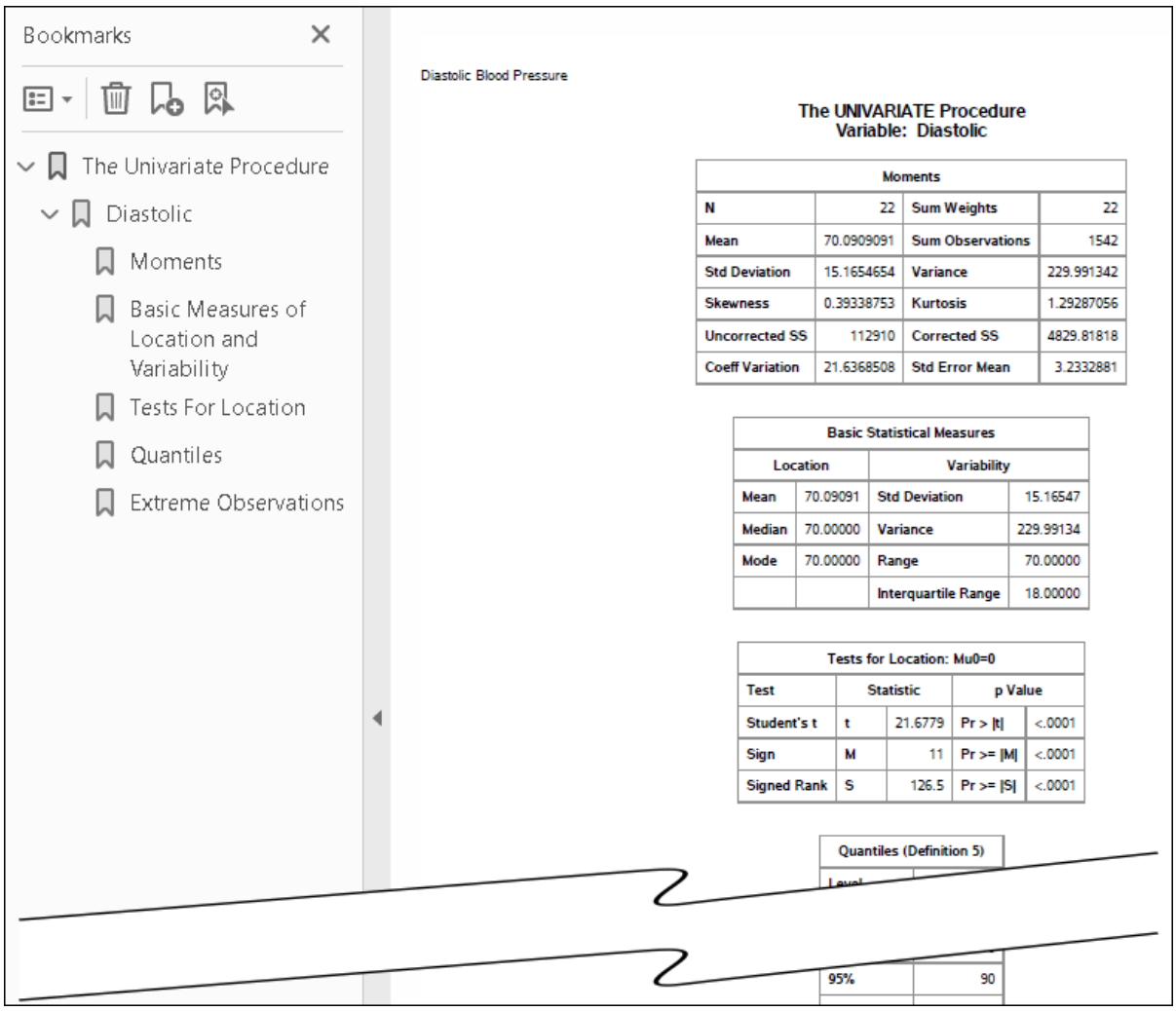
Note: The mode displayed is the smallest of 2 modes with a count of 4.

Tests for Location: Mu0=0				
Test	Statistic		p Value	
Student's t	t	39.82364	Pr > |t|	<.0001
Sign	M	11	Pr >= |M|	<.0001
Signed Rank	S	126.5	Pr >= |S|	<.0001

Quantiles (Definition 5)	
Level	Quantile
100% Max	165
99%	165
95%	140
90%	134
75% Q3	125

...more observations...

Output 3.2 *Partial PDF Output with Diastolic Output Objects*



See Also

Statements

- “ODS SELECT Statement” on page 20
- “ODS SHOW Statement” in *SAS Output Delivery System: User’s Guide*
- “ODS TRACE Statement” on page 70

ODS SELECT Statement

Specifies output objects for ODS destinations.

Category: ODS: Output Control

Tip: You can maintain a selection list for one destination and an exclusion list for another. However, it is easier to understand the results if you maintain the same types of lists for all of the destinations to which you route output.

See: "ODS EXCLUDE Statement" on page 12

Example: "Example 7: Table Header and Footer Border Formatting" on page 580

Syntax

ODS <*ODS-destination*> **SELECT** *selection(s)* | ALL | NONE;

Required Arguments

selection(s)

specifies output objects to add to a selection list. ODS sends the items in the selection list to all active ODS destinations. By default, ODS automatically modifies selection lists when a DATA step that uses ODS or a procedure step ends. For information about modifying these lists, see For information about ending DATA and procedure steps, see "DATA Step Processing" in *SAS Programmer's Guide: Essentials*.

Each *selection* has the following form.

output-object <(PERSIST)>

output-object

specifies the output object to select.

To specify an output object, you need to know which output objects your SAS program produces. The ODS TRACE statement writes to the SAS log a trace record that includes the path, the label, and other information about each output object that your SAS program produces. You can specify an output object as one of the following:

- a full path. For example, the following is the full path of the output object:

```
Univariate.City_Pop_90.TestsForLocation
```

- a partial path. A partial path consists of any part of the full path that begins immediately after a period (.) and continues to the end of the full path. For example, suppose the full path is the following:

```
Univariate.City_Pop_90.TestsForLocation
```

Then the partial paths are as follows:

```
City_Pop_90.TestsForLocation
TestsForLocation
```

- a label that is enclosed in quotation marks.

For example:

```
"The UNIVARIATE Procedure" "Tests For Location"
```

- a label path. For example, the label path for the output object is as follows:

```
"The UNIVARIATE Procedure"."CityPop_90"."Tests For Location"
```

Note: The trace record shows the label path only if you specify the LABEL option in the ODS TRACE statement.

- a partial label path. A partial label path consists of any part of the label that begins immediately after a period (.) and continues to the end of the label. For example, suppose the label path is the following:

```
"The UNIVARIATE Procedure"."CityPop_90"."Tests For Location"
```

Then the partial label paths are as follows:

```
"CityPop_90"."Tests For Location"
"Tests For Location"
```

- a mixture of labels and paths.
- any of the partial path specifications, followed by a number sign (#) and a number. For example, `TestsForLocation#3` refers to the third output object that is named `TestsForLocation`.

See "ODS TRACE Statement" on page 70

(PERSIST)
keeps the *output-object* that precedes the PERSIST option in the selection list, even if the DATA or procedure step ends, until you explicitly modify the list with one of the following:

- any ODS EXCLUDE statement
- ODS SELECT NONE
- ODS SELECT ALL
- an ODS SELECT statement that applies to the same output object but does not specify PERSIST

Requirement You must enclose PERSIST in parentheses.

ALL
specifies that ODS send all of the output objects to the open destination.

Alias DEFAULT

Interaction If you specify ALL without specifying a destination, ODS sets the overall list to SELECT ALL and sets all other lists to their defaults.

NONE
specifies that ODS does not send any output objects to the open destination.

Interaction If you specify NONE and you do not specify a destination, ODS sets the overall list to SELECT NONE and sets all other lists to their defaults.

Tips Using the NONE action is different from closing a destination. The output destination is still open, but ODS restricts the output that it sends to the destination.

To temporarily suspend a destination, use ODS SELECT NONE. Use ODS SELECT ALL when you want to resume sending output to the suspended destination.

Optional Arguments

(NOWARN)
suppresses the warning that an output object was requested but not created.

Requirement The NOWARN option must be enclosed in parentheses.

Interaction The NOWARN option cannot be used with the ALL option or the NONE option.

Example The ODS SELECT statement in the following example specifies that no warning is created if the output object Summary is requested but not created.

```
ods select summary (nowarn);

proc contents data=sashelp.class;
run;
```

ODS-destination
specifies to which ODS destination's selection list to write, where *ODS-destination* can be any valid ODS destination except for the OUTPUT destination.

Default If you omit *ODS-destination*, ODS writes to the overall selection list.

Restriction You cannot write to the OUTPUT destination's selection list.

Tip To set the selection list for the Output destination to something other than the default, see the "ODS OUTPUT Statement" in *SAS Output Delivery System: User's Guide*.

See "Understanding ODS Destinations" in *SAS Output Delivery System: User's Guide* for a discussion of ODS destinations.

WHERE=*where-expression*
selects output objects that meet a particular condition. For example, the following statement selects only output objects with the word "Histogram" in their name:

```
ods select where=(_name_ ?  'Histogram');
```

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands. *where-expression* has this form.

(*subsetting-variable* <*comparison-operator where-expression-n*>)

subsetting-variable
Subsetting variables are a special type of WHERE expression operand used by SAS to help you find common values in items. For example, this ODS SELECT statement selects only output objects with the path **`City_Pop_90.TestsForLocation`** :

```
ods select  / where=(_path_ = 'City_Pop_90.TestsForLocation' );
```

subsetting-variable is one of the following:

LABEL
is the label of the output object.

LABELPATH
is the label path of the output object.

NAME
is the name of the output object.

PATH
is the name path of the output object.

operator
compares a variable with a value or with another variable. *operator* can be AND, OR NOT, OR, AND NOT, or a comparison operator.

The following table lists some comparison operators:

Table 3.2 *Examples of Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Details

Using Selection and Exclusion Lists

For each ODS destination, ODS maintains either a selection list or an exclusion list of output objects. You can use the default output objects selected or excluded for each destination or you can specify which output object you want to produce by selecting or excluding them from a list.

A selection list is a list of output objects that are sent to an ODS destination. An exclusion list is a list of output objects that are excluded from an ODS destination. ODS also maintains an overall selection or exclusion list of output objects. By checking the destination-specific lists and the overall list, ODS determines what output objects to produce. These lists can be modified by using the ODS SELECT statement and the ODS EXCLUDE statement.

TIP You can maintain a selection list for one destination and an exclusion list for another. However, the results are less complicated if you maintain the same types of lists for all the destinations to which you route output.

You can view the contents of the exclusion and selection lists by using the ODS SHOW statement. The current selection list is written to the SAS log.

EXCLUDE ALL is the default setting for the ODS OUTPUT destination. SELECT ALL is the default setting for all other destinations. To change the default selection and exclusion lists, use the ODS SELECT or ODS EXCLUDE statements or use the exclude and select actions that are available for some of the ODS statements. However, to set the exclusion list for the OUTPUT destination to something other than the default, use the "ODS OUTPUT Statement" in *SAS Output Delivery System: User's Guide*. For a list of ODS output destinations and explanations of each, see "Understanding ODS Destinations" in *SAS Output Delivery System: User's Guide*.

In order to view output objects that are selected or excluded from your program, use the ODS TRACE statement. The ODS TRACE statement prints the output objects that are selected and excluded and puts the information in a trace record that is written to the SAS log. The trace provides the path, the label, and other information about output objects that are selected and excluded. For complete documentation about viewing and selecting output objects, see the "ODS SELECT Statement" on page 20, the "ODS EXCLUDE Statement" on page 12, and the "ODS TRACE Statement" on page 70.

Examples

Example 1: Using a Selection List with Multiple Procedure Steps

Features:

ODS SELECT statement
- with label
- with name
- with and without PERSIST
- ALL

ODS SHOW statement

ODS HTML statement options
- BODY=
- CONTENTS=
- FRAME=
- PAGE=

PROC GLM

PROC PRINT

PROC PLOT

Details

This example runs the same procedures multiple times to illustrate how ODS maintains and modifies a selection list. The ODS SHOW statement writes the overall selection list to the SAS log. The example does not alter selection lists for individual destinations. The contents file that is generated by the ODS HTML

statement shows which output objects are routed to both the HTML and the LISTING destinations.

This example creates and prints data sets from the parameter estimates that PROC GLM generates. This procedure is part of SAS/STAT software.

Program

```
ods html body='your-file-path/odspersist-body.htm';

ods show;

ods select ParameterEstimates
           "Type III Model ANOVA";

ods show;

proc glm data=iron;
   model loss=fe;
   title 'Parameter Estimates and Type III Model ANOVA';
run;

ods show;

quit;

ods show;

proc glm data=iron;
   model loss=fe;
   title 'All Output Objects Selected';
run;
quit;

ods select OverallANOVA(persist) "Fit Statistics";

proc glm data=iron;
   model loss=fe;
   title 'OverallANOVA and Fitness Statistics';
run;

quit;

ods show;

proc glm data=iron;
   model loss=fe;
   title 'OverallANOVA';
   title2 'Part of the Selection List Persists';
run;

quit;

proc print data=iron;
   title 'The IRON Data Set';
run;

ods select all;

proc plot data=iron;
   plot fe*loss='*' / vpos=25 ;
   label fe='Iron Content'
```

```
        loss='Weight Loss';
   title 'Plot of Iron Versus Loss';
run;

quit;

ods html close;
```

Program Description

Create HTML output and add text.

```
ods html body='your-file-path/odspersist-body.htm';
```

Write the overall selection list to the SAS log. The ODS SHOW statement writes to the SAS log the overall list, which is set to SELECT ALL by default. See the “SAS Log” on page 29.

```
ods show;
```

Specify the output objects that will be sent to the open destinations. The ODS SELECT statement determines which output objects ODS sends to the LISTING and HTML destinations. In this case, ODS sends all output objects that are named **`ParameterEstimates`** and all output objects that are labeled "**`Type III Model ANOVA`**" to the two destinations.

```
ods select ParameterEstimates
           "Type III Model ANOVA";
```

Write the modified overall selection list to the SAS log. The ODS SHOW statement writes to the SAS log the overall selection list, which now contains the two items that were specified in the ODS SELECT statement. See the “SAS Log” on page 29.

```
ods show;
```

Create the output objects and send the selected output objects to the open destinations. As PROC GLM sends each output object to the Output Delivery System, ODS sends the two output objects from PROC GLM that match the items in the selection list to the open destinations. See 1. in the table of contents in “HTML Output” on page 29. Note that it is the label of an output object, not its name, that appears in the table of contents. The label for ParameterEstimates is "Solution".

```
proc glm data=iron;
   model loss=fe;
   title 'Parameter Estimates and Type III Model ANOVA';
run;
```

Write the overall selection list to the SAS log. PROC GLM supports run-group processing. Therefore, the RUN statement does not end the procedure, and ODS does not automatically modify the selection list. See the “SAS Log” on page 29.

```
ods show;
```

End the GLM procedure. The QUIT statement ends the procedure. ODS removes all objects that are not specified with PERSIST from the selection list. Because this action removes all objects from the list, ODS sets the list to its default, SELECT ALL.

```
quit;
```

Write the current selection list to the SAS log. The ODS SHOW statement writes the current selection list to the SAS log. See the “SAS Log” on page 29.

```
ods show;
```

Create the output objects, send the selected output objects to the open destinations, and end the procedure. As PROC GLM sends each output object to the Output Delivery System, ODS sends all the output objects to the HTML and LISTING destinations. See 2. in the table of contents in "HTML Output" on page 29. The QUIT statement ends the procedure. Because the list uses the argument ALL, ODS does not automatically modify it when the PROC step ends.

```
proc glm data=iron;
   model loss=fe;
   title 'All Output Objects Selected';
run;
quit;
```

Modify the overall selection lists. This ODS SELECT statement modifies the overall selection list. It sends all output objects that are named OverallANOVA, and all output objects that are labeled Fit Statistics, to both the HTML and LISTING destinations. The PERSIST option specifies that OverallANOVA should remain in the selection list when ODS automatically modifies it.

```
ods select OverallANOVA(persist) "Fit Statistics";
```

Create the output objects and send the selected output objects to the open destinations. As PROC GLM sends each output object to the Output Delivery System, ODS sends the two output objects from PROC GLM that match the items in the selection list to the HTML and LISTING destinations. See 3. in the table of contents in "HTML Output" on page 29.

```
proc glm data=iron;
   model loss=fe;
   title 'OverallANOVA and Fitness Statistics';
run;
```

End the GLM procedure and automatically modify the selection list. When the QUIT statement ends the procedure, ODS automatically modifies the selection list. Because OverallANOVA was specified with the PERSIST option, it remains in the selection list. Because Fitness Statistics was not specified with the PERSIST option, ODS removes it from the selection list.

```
quit;
```

Write the current selection list to the SAS log. The ODS SHOW statement writes the current selection list to the SAS log. See the "SAS Log" on page 29.

```
ods show;
```

Create the output objects and send the selected output objects to the open destinations. As PROC GLM sends each output object to the Output Delivery System, ODS sends only the output object that is named OverallANOVA to the HTML and LISTING destinations. See 4. in the table of contents in "HTML Output" on page 29.

```
proc glm data=iron;
   model loss=fe;
   title 'OverallANOVA';
   title2 'Part of the Selection List Persists';
run;
```

End the GLM procedure and automatically modify the selection list. When the QUIT statement ends the procedure, ODS automatically modifies the selection list. Because OverallANOVA was specified with the PERSIST option, it remains in the selection list.

```
quit;
```

PROC PRINT does not produce any output that is named OverallANOVA. Therefore, no PROC PRINT output is sent to the ODS destinations.

```
proc print data=iron;
   title 'The IRON Data Set';
run;
```

Reset all selection lists. This ODS SELECT statement resets all selection lists to their defaults.

```
ods select all;
```

Create the plots. As PROC PLOT creates and sends each output object to the Output Delivery System, ODS sends each one to the HTML and LISTING destinations because their lists and the overall list are set to SELECT ALL (the default).

```
proc plot data=iron;
   plot fe*loss='*' / vpos=25 ;
   label fe='Iron Content'
       loss='Weight Loss';
   title 'Plot of Iron Versus Loss';
run;
```

End the PLOT procedure. The QUIT statement ends the PLOT procedure. Because the list uses the argument ALL, ODS does not automatically modify the list when the PROC step ends.

```
quit;
```

Close the HTML destination. This ODS HTML statement closes the HTML destination and all the files that are associated with it.

```
ods html close;
```

SAS Log

Output 3.3 *The ODS SHOW Statement Writes the Current Selection List to the SAS Log.*

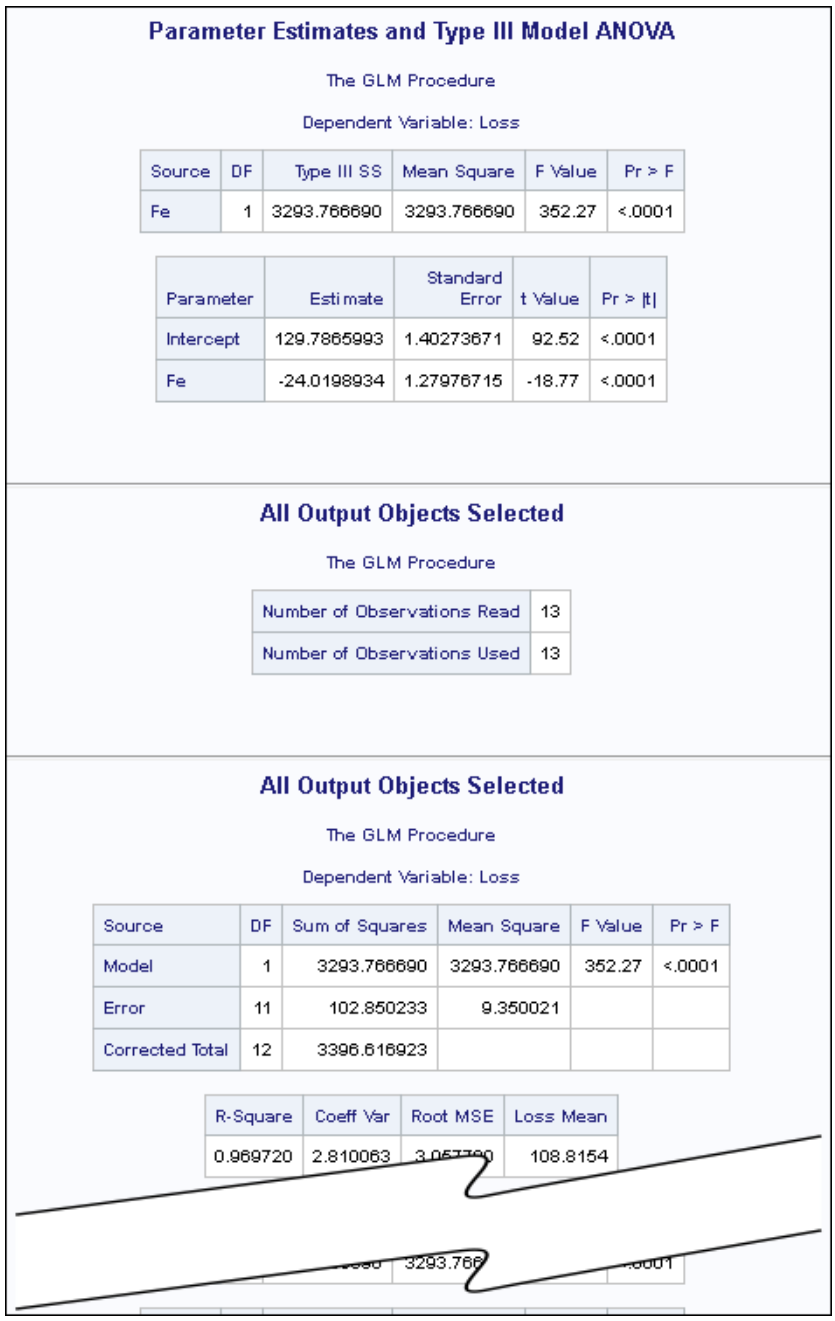
```
119  ods show;
Current OVERALL select list is:
1. OverallANOVA(PERSIST)
120  proc glm data=iron;
121     model loss=fe;
122     title 'OverallANOVA';
123     title2 'Part of the Selection List Persists';
124  run;
125  quit;
```

HTML Output

The contents file shows the output objects from each procedure that ODS sent to the open ODS destinations. You can see that no output was written to the HTML destination for PROC PRINT (because PROC PRINT did not produce anything

whose name matched the name in the selection list). You can also see that the PROC PLOT output was written to the HTML destination after the ODS SELECT ALL statement was executed.

Output 3.4 *Contents File Produced by the ODS HTML Statement*



Parameter Estimates and Type III Model ANOVA

The GLM Procedure

Dependent Variable: Loss

Source	DF	Type III SS	Mean Square	F Value	Pr > F
Fe	1	3293.766690	3293.766690	352.27	<.0001

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	129.7865993	1.40273671	92.52	<.0001
Fe	-24.0198934	1.27976715	-18.77	<.0001

All Output Objects Selected

The GLM Procedure

Number of Observations Read	13
Number of Observations Used	13

All Output Objects Selected

The GLM Procedure

Dependent Variable: Loss

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	3293.766690	3293.766690	352.27	<.0001
Error	11	102.850233	9.350021		
Corrected Total	12	3396.616923			

R-Square	Coeff Var	Root MSE	Loss Mean
0.969720	2.810063	3.057780	108.8154

Example 2: Conditionally Selecting Output Objects

Features:

ODS SELECT statement option
- WHERE=

ODS TRACE statement options
- LABEL
- EXCLUDED

ODS HTML statement
PROC UNIVARIATE

Program

```
data BPressure;
   length PatientID $2;
   input PatientID $ Systolic Diastolic @@;
   datalines;
   CK 120 50  SS 96  60 FR 100 70
   CP 120 75  BL 140 90 ES 120 70
   CP 165 110 JI 110 40 MC 119 66
   FC 125 76  RW 133 60 KD 108 54
   DS 110 50  JW 130 80 BH 120 65
   JW 134 80  SB 118 76 NS 122 78
   GS 122 70  AB 122 78 EC 112 62
   HH 122 82
   ;
   run;

title 'Systolic and Diastolic Blood Pressure';

ods trace on / label excluded;

ods select where=(_path_ ? "Diastolic" and  _name_='Moments') ;

proc univariate data=BPressure;
   var Systolic Diastolic;
run;
```

Program Description

Create the BPressure data set.

```
data BPressure;
   length PatientID $2;
   input PatientID $ Systolic Diastolic @@;
   datalines;
   CK 120 50  SS 96  60 FR 100 70
   CP 120 75  BL 140 90 ES 120 70
   CP 165 110 JI 110 40 MC 119 66
   FC 125 76  RW 133 60 KD 108 54
   DS 110 50  JW 130 80 BH 120 65
   JW 134 80  SB 118 76 NS 122 78
   GS 122 70  AB 122 78 EC 112 62
   HH 122 82
   ;
   run;
```

Add a title.

```
title 'Systolic and Diastolic Blood Pressure';
```

Specify that SAS write the trace record to the SAS log. This ODS TRACE statement writes the trace record to the SAS log. The LABEL option includes label

paths in the trace record. The EXCLUDED option includes information about output objects that SAS excludes from the output destination.

```
ods trace on / label excluded;
```

Select output objects. The ODS SELECT statement with the WHERE = option specified selects output objects that are named 'Moments' and that have 'Diastolic' in the pathname.

```
ods select where=(_path_ ? "Diastolic" and  _name_='Moments') ;
```

Create the output objects and send the selected output objects to the open destination. As PROC UNIVARIATE sends each output object to the Output Delivery System, ODS sends the output object from PROC UNIVARIATE that matches the items in the selection list to the open destination.

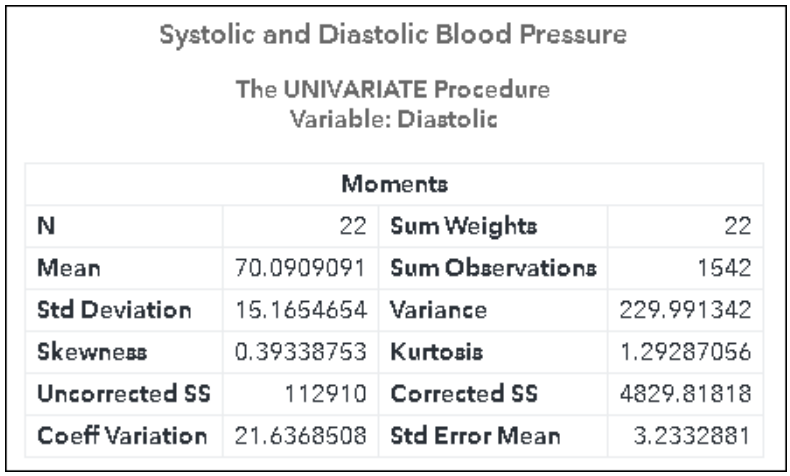
```
proc univariate data=BPressure;
   var Systolic Diastolic;
run;
```

SAS Log: Trace Record

Output 3.5 *SAS Log Including Trace Record*

```
100  ods trace on / label excluded;
101     ods select where=(_path_ ? "Diastolic" and  _name_='Moments') ;
102     proc univariate data=BPressure;
103        var Systolic Diastolic;
104     run;
Output Excluded:
-------------
Name:       Moments
Label:      Moments
Template:   base.univariate.Moments
Path:       Univariate.Systolic.Moments
Label Path: 'The Univariate Procedure'.'Systolic'.'Moments'
-------------
Output Excluded:
-------------
Name:       BasicMeasures
Label:      Basic Measures of Location and Variability
Template:   base.univariate.Measures
Path:       Univariate.Systolic.BasicMeasures
Label Path: 'The Univariate Procedure'.'Systolic'.'Basic Measures of Location and
Variability'
-------------
Output Excluded:
-------------
Name:       TestsForLocation
Label:      Tests For Location
Template:   base.univariate.Location
Path:       Univariate.Systolic.TestsForLocation
Label Path: 'The Univariate Procedure'.'Systolic'.'Tests For Location'
-------------
Output Excluded:
-------------
Name:       Quantiles
Label:      Quantiles
Template:   base.univariate.Quantiles
Path:       Univariate.Systolic.Quantiles
Label Path: 'The Univariate Procedure'.'Systolic'.'Quantiles'
-------------
Output Excluded:
-------------
Name:       ExtremeObs
Label:      Extreme Observations
Template:   base.univariate.ExtObs
Path:       Univariate.Systolic.ExtremeObs
Label Path: 'The Univariate Procedure'.'Systolic'.'Extreme Observations'
-------------
Output Added:
-------------
Name:       Moments
Label:      Moments
Template:   base.univariate.Moments
Path:       Univariate.Diastolic.Moments
Label Path: 'The Univariate Procedure'.'Diastolic'.'Moments'
-------------
Output Excluded:
-------------
Name:       BasicMeasures
Label:      Basic Measures of Location and Variability
Template:   base.univariate.Measures
Path:       Univariate.Diastolic.BasicMeasures
Label Path: 'The Univariate Procedure'.'Diastolic'.'Basic Measures of Location
and Variability'
-------------
Output Excluded:
-------------
Name:       TestsForLocation
```

HTML Output: Univariate Procedure



Systolic and Diastolic Blood Pressure

The UNIVARIATE Procedure
Variable: Diastolic

Moments			
N	22	Sum Weights	22
Mean	70.0909091	Sum Observations	1542
Std Deviation	15.1654654	Variance	229.991342
Skewness	0.39338753	Kurtosis	1.29287056
Uncorrected SS	112910	Corrected SS	4829.81818
Coeff Variation	21.6368508	Std Error Mean	3.2332881

See Also

Statements

- “ODS EXCLUDE Statement” on page 12
- “ODS SHOW Statement” in *SAS Output Delivery System: User’s Guide*
- “ODS TRACE Statement” on page 70

4

Other SAS Statements That Are Used with ODS Procedures

Dictionary 35
ODS GRAPHICS Statement 35
ODS TEXT= Statement 62
ODS TRACE Statement 70
ODS PROCLABEL Statement 76

Dictionary

ODS GRAPHICS Statement

Enables or disables ODS Graphics processing and sets graphics environment options. This statement affects ODS template-based (ODS Graphics) graphics only. The ODS GRAPHICS statement does not affect device-based graphics (SAS/GRAPH).

Valid in: Anywhere

Category: ODS: Output Control

Default: ON.

Interaction: SAS/GRAPH device-based global statements such as GOPTIONS, SYMBOL, PATTERN, AXIS, and LEGEND do not affect template-based graphics. The ODS GRAPHICS statement does not affect device-based graphics.

See: For information about SVG and Universal Printing, see "ODS HTML5 Statement" in *SAS Output Delivery System: User's Guide*.

Syntax

ODS GRAPHICS <OFF | ON> </ *options*> ;

Summary of Optional Arguments

ANTIALIAS
: smooths jagged edges of all components in the graph.

ANTIALIAS= ON | OFF
: specifies whether anti-aliasing is applied to the rendering of the line and markers in any graph.

ANTIALIASMAX= *n*
: specifies the maximum number of graphics elements before anti-aliasing is disabled.

ATTRPRIORITY=COLOR | NONE
: specifies a priority for cycling of the group attributes.

BORDER
: draws a border around the graph.

BORDER=ON | OFF
: specifies whether to draw a border around each graph.

BYLINE=NOBYLINE | TITLE | FOOTNOTE
: specifies how the BY line is displayed in graphs.

DATASKINMAX=*n*
: specifies the maximum number of graphical elements allowed per plot when a data skin is applied.

DISCRETEMAX=*n*
: specifies the maximum number of discrete values to be shown in any graph.

DRILLTARGET="_blank" | "_self" | "_parent" | "_top" | "frame-name"
: specifies the window that displays the drill-down output.

GROUPMAX=*n*
: specifies the maximum number of group values to be shown in any graph.

HEIGHT=*dimension*
: specifies the height of a graph.

IMAGEMAP
: specifies to generate data tips.

IMAGEMAP=ON | OFF
: specifies whether data tips are generated.

IMAGENAME="*filename*"
: specifies the base image file name.

LABELMAX=*n*
: specifies the maximum number of labeled areas before data label collision avoidance is disabled.

LABELPLACEMENT= GREEDY | SA
: specifies the label-placement algorithm to use for positioning labels in the graphs.

LEGENDAREAMAX=*n*

specifies an integer that is interpreted as the maximum percentage of the overall graphics area that a legend can occupy.

LINEPATTERNOBSMAX=*n*
specifies the maximum the number of vertices for a patterned line.

LOESSOBSMAX=*n*
specifies an upper limit for the number of observations that can be used with a loess plot.

NBINSMAX=*n*
specifies the maximum number of bins that are processed for histograms.

NOANTIALIAS
does not smooth jagged edges of components other than text in the graph.

NOBORDER
does not draw a border around the graph.

NOIMAGEMAP
specifies not to generate data tips.

NOSCALE
does not scale the components of a graph proportionally.

NOSCALEMARKERS
does not scale the markers with the graph size.

NOSUBPIXEL
never uses subpixel rendering.

NXYBINSMAX=*n*
specifies the maximum number of bins that are processed for heat maps.

OBSMAX=*n*
specifies the maximum number of observations that are processed.

OFF
disables ODS Graphics processing.

ON
enables ODS Graphics processing.

OUTPUTFMT=*file–type* | STATIC
specifies the output format used to generate image or vector graphic files.

PANELCELLMAX=*n*
specifies the maximum number of cells in a graph panel where the number of cells is determined dynamically by classification variables.

POP
restores the most recently pushed settings from the stack.

PUSH
pushes the current ODS GRAPHICS settings to a stack.

RESET
resets all *options* to their defaults.

RESET=ALL | *option*
resets one or all of the ODS GRAPHICS options to their defaults.

SCALE
scales the components of a graph proportionally.

SCALE=ON | OFF
specifies whether the content of a graph is scaled proportionally.

SCALEMARKERS
scales the markers with the graph size.

SCALEMARKERS=ON | OFF
specifies whether the plot markers are to be scaled with the graph size.

SHOW
: writes the current ODS Graphics settings to the SAS log.

STACKDEPTHMAX=*n*
: specifies the maximum stack depth for PUSH and POP requests.

SUBPIXEL
: always uses subpixel rendering, when applicable, for rendering lines and bars.

SUBPIXEL=ON | OFF
: specifies whether subpixel rendering should be used for rendering ODS Graphics.

TIPMAX=*n*
: specifies the maximum number of distinct data tip areas allowed before data tips are disabled.

TOTALCELLMAX=*n*
: specifies the maximum number of total cells in a graph where the number of cells is determined dynamically by classification variables.

WIDTH=*dimension*
: specifies the width of any graph.

Without Arguments

If ODS Graphics is currently disabled, then specifying the ODS GRAPHICS statement without options enables it. If ODS Graphics is currently enabled, then specifying the ODS GRAPHICS statement leaves it enabled.

Optional Arguments

You can specify one of the following arguments before '/:'

ON
enables ODS Graphics processing. This is the default if no argument is used.

Alias YES

Example Enable ODS graphics:

```
ods graphics on </option(s)>;
```

OFF
disables ODS Graphics processing.

Alias NO

Example Disable ODS graphics:

```
ods graphics off;
```

Optional Arguments that Follow '/'

You can specify one or more of the following arguments after '/:'

ANTIALIAS
smooths jagged edges of all components in the graph. Anti-aliasing smooths the appearance of lines and some markers. Text displayed in the graph is always anti-aliased. For graphical displays that plot large numbers of points it is

recommended that NOANTIALIAS or ANTIALIAS=OFF be specified for performance considerations.

Default ANTIALIAS

Restriction If the number of markers or lines in the plot exceeds the number specified by the ANTIALIASMAX= option, then the ANTIALIAS option is disabled. This is true even if you specify the option ANTIALIAS=ON or ANTIALIAS.

ANTIALIAS= ON | OFF

specifies whether anti-aliasing is applied to the rendering of the line and markers in any graph. Anti-aliasing smooths the appearance of lines and some markers. Text displayed in the graph is always anti-aliased. For graphical displays that plot large numbers of points it is recommended that ANTIALIAS=OFF be specified for performance considerations.

ON

smooths jagged edges of all components in the graph.

Alias YES

OFF

does not smooth jagged edges of components other than text in the graph.

Alias NO

Default ANTIALIAS or ANTIALIAS=ON | YES

Restriction If the number of markers or lines in the plot exceeds the number specified by the ANTIALIASMAX= option, then the ANTIALIAS option is disabled. This is true even if you specify the option ANTIALIAS=ON or ANTIALIAS.

ANTIALIASMAX= *n*

specifies the maximum number of graphics elements before anti-aliasing is disabled. For example, if there are more than 400 scatter point markers to be anti-aliased and ANTIALIASMAX=400, then no markers are anti-aliased. The default value is 4000.

Note: The ANTIALIASMAX= option specifies the maximum number of graphics elements to be anti-aliased in each plot on a per-plot basis. The default remains at 4000. If any plot in a graph contains more than 4000 elements, anti-aliasing is disabled for that plot. Anti-aliasing is enabled for the rest of the graph in that case.

n

specifies a positive integer.

Default 4000

ATTRPRIORITY=COLOR | NONE

specifies a priority for cycling of the group attributes.

COLOR

assigns priority to the color attribute rotation by cycling through the list of colors while holding the marker symbol and line pattern constant. When all of the colors are exhausted, the marker symbol and line style attributes

increment to the next element, and then the colors in the list are repeated. This pattern repeats as needed.

NONE
does not use an attribute priority in the rotation pattern, even if one is set in the active style's AttrPriority attribute. The rotation pattern cycles progressively through the attribute lists.

Default The AttrPriority attribute of the graph style element, or NONE if the current style does not define the AttrPriority style attribute.

Interaction The default lists of data colors, contrast colors, marker symbols, and line patterns are set in the active style's GraphData1–GraphData*N* elements.

Tip Use the ATTRPRIORITY=NONE option if you want groups to be distinguished by color, marker, and line changes for all styles that use color.

BORDER
draws a border around the graph.

Default BORDER

BORDER=ON | OFF
specifies whether to draw a border around each graph.

ON
draws a border around the graph.

Alias YES

OFF
does not draw a border around the graph.

Alias NO

Default BORDER=ON

BYLINE=NOBYLINE | TITLE | FOOTNOTE
specifies how the BY line is displayed in graphs. The option specifies how the BY line is displayed when an analysis is run with a BY statement. By default, no BY line is displayed.

The following code is an example of how the placement of the BY line is controlled in most graph templates:

```
if (_BYTITLE_)
   entrytitle _BYLINE_ / textattrs=GraphValueText;
else
   if (_BYFOOTNOTE_)
      entryfootnote halign=left _BYLINE_;
   endif;
endif;
```

You can modify the graph template if you want to change how the BY line is displayed. Because most graphs have titles and few graphs have footnotes, the BY line looks better when it is displayed as a footnote. For complete documentation about the Graph Template Language, see *SAS Graph Template Language: User's Guide*.

When the BYLINE= option is specified, and there are BY groups, ODS creates a BY line and sets the appropriate special dynamic variables. The following table lists the special dynamic variables for BY lines. For complete documentation about special dynamic variables, see "Special Dynamic Variables" in *SAS Graph Template Language: User's Guide*.

Table 4.1 *Special Dynamic Variables for BY Lines*

BYFOOTNOTE	This variable is set to 1 when you specify a BY statement and the ODS GRAPHICS BYLINE= option is set to FOOTNOTE. Otherwise, the variable is set to 0 or is NULL.
BYTITLE	This variable is set to 1 when you specify a BY statement and the ODS GRAPHICS BYLINE= option is set to TITLE. Otherwise, the variable is set to 0 or is NULL.
BYLINE	This variable contains the text string that can be displayed as a title or footnote.

The variables in the table are set automatically only for analytical procedures that support ODS Graphics. For all other procedures, the variables are not set automatically (NULL). To determine whether the procedure that you are using supports ODS Graphics, refer to the procedure documentation.

NOBYLINE
specifies that no BY line is displayed. NOBYLINE is the default.

FOOTNOTE
specifies that the BY line is displayed as a left-justified graph footnote. This is the recommended setting.

TITLE
specifies that the BY line is displayed as a centered graph title. Specifying TITLE is not recommended because graphs are not designed to have additional title lines.

Default NOBYLINE

Restriction This option does not work with the ODS Graphics procedures such as SGPLOT and SGPANEL. To remove BY lines in those procedures, use the NOBYLINE SAS system option. Example:
`options nobyline;`

DATASKINMAX=*n*
specifies the maximum number of graphical elements allowed per plot when a data skin is applied.

n
specifies a positive integer.

Default 200

DISCRETEMAX=*n*
specifies the maximum number of discrete values to be shown in any graph. Bar charts and box plots are examples of affected plot types. Scatter plots and other plot types can be affected if the data to be plotted is discrete or the axis is discrete.

n
specifies a positive integer.

Default 1000

Tips Some plot layers might be unaffected by the DISCRETEMAX= option, and those layers are rendered. If all layers are affected, a blank graph is rendered.

If the value specified by the DISCRETEMAX= option is exceeded by any plot layer in the graph, that layer is not drawn and a warning message is issued. In that case, use the DISCRETEMAX= option to increase the maximum number of discrete values that are allowed. The log message includes a suggested value for DISCRETEMAX=.

DRILLTARGET="_blank" | "_self" | "_parent" | "_top" | "frame-name"
specifies the window that displays the drill-down output.

Note: This option is supported only for HTML.

"_blank"
opens a new browser window to display the drilldown output.

Default _blank is the default.

Requirements You must enclose *_blank* in quotation marks.

You must specify *_blank* in lowercase.

"_self"
opens the drill-down output in the same window.

Requirements You must enclose *_self* in quotation marks.

You must specify *_self* in lowercase.

"_parent"
opens the drill-down output in the parent frame.

Requirements You must enclose *_parent* in quotation marks.

You must specify *_parent* in lowercase.

"_top"
opens the drill-down output in the full body of the window.

Requirements You must enclose *_top* in quotation marks.

You must specify *_top* in lowercase.

"frame-name"
opens the drill down output in the named frame in the current window. If the name does not exist, the output is opened in a new window.

Requirement You must enclose *frame–name* in quotation marks.

GROUPMAX=*n*
specifies the maximum number of group values to be shown in any graph. Any graph that supports the GROUP= option is affected.

n
specifies a positive integer.

Default 1000

Tip If the value specified by the GROUPMAX= option is exceeded by any plot layer in the graph, that layer is rendered. The system ignores the GROUP= option and issues a warning message. In that case, use the GROUPMAX= option to increase the maximum number of group values that are allowed. The log message includes a suggested value for GROUPMAX=.

HEIGHT=*dimension*
specifies the height of a graph.

dimension
is a nonnegative number followed by one of these units of measure:

Table 4.2 *Units of Measure for Dimension*

cm	Centimeters
in	Inches
mm	Millimeters
pct or %	Percentage
pt	Point size (72 points = 1 inch)
px	Pixels

Default Determined by the software, or the value of the DesignHeight= option in a STATGRAPH template. Typically, the value is 480px.

Tips If only the HEIGHT= option is specified, then the default aspect ratio of the graph is maintained.

When you change the size of a graph, by default graph elements such as markers, lines, text, and titles are scaled proportionally. This scaling takes effect even when you specify dimensions for the graph elements. To disable the scaling, specify the NOSCALE option.

IMAGEMAP
specifies to generate data tips.

Default NOIMAGEMAP

Restriction This option affects HTML output only.

Note Image maps are supported with SVG output using HTML5. However, image maps are supported only when the HTML5 SVG mode is INLINE (the default value).

IMAGEMAP=ON | OFF
controls data tips and drill down generation. Data tips are pieces of explanatory text that appear when you hold the mouse pointer over the data portions of a graph contained in an HTML page.

ON
specifies to generate data tips.

Alias YES

OFF
specifies not to generate data tips.

Alias NO

Default NOIMAGEMAP or IMAGEMAP=OFF | NO

Restriction This option affects HTML output only.

Note Image maps are supported with SVG output using HTML5. However, image maps are supported only when the HTML5 SVG mode is INLINE (the default value).

IMAGENAME="*filename*"
specifies the base image file name. If more than one image is generated, each is assigned a file name that consists of a base name followed by a number in order to create unique names. This numbering can be reset with the RESET=INDEX option. Path information (if needed) can be set with the GPATH= option on the ODS destination statement. The default path is the current output directory. A file extension for *filename* is automatically generated based on the OUTPUTFMT= option.

Note: If a BY statement is in effect, you can uniquely name image files based on BY groups. To do this, insert #BY text into the image name. For more information, see "Substituting BY Values in the Image Name" on page 55.

Default The name of the output object.

Restrictions *filename* must be a single name. It must not include any path specification or image-format name extension.

filename can contain only the following types of characters: letter, digit, underscore, and space.

Requirement You must enclose *filename* in quotation marks.

See "Specifying and Resetting the Image Name" on page 54

LABELMAX=*n*
specifies the maximum number of labeled areas before data label collision avoidance is disabled. For example, if there are more than 50 points to be labeled and LABELMAX=50, then collision avoidance is turned off and the labels are all displayed at the top right of the data points.

n
specifies a positive integer.

Default 200

Restriction Data label collision avoidance is turned off under the following conditions:

- The number of observations with nonmissing labels exceeds the value specified by LABELMAX=.
- The number of observations exceeds five times the value specified by LABELMAX=.

A message is then sent to the SAS log.

Tip To turn off collision avoidance, specify LABELMAX=0.

LABELPLACEMENT= GREEDY | SA
specifies the label-placement algorithm to use for positioning labels in the graphs. The following labels are affected:

- data labels for needle plots, scatter plots, series plots, step plots, and vector plots
- vertex labels for line charts
- curve labels when the curve label is positioned at the start or end of the curve

GREEDY
specifies the Greedy method for managing label collision. The Greedy method tries different placement combinations in order to find an optimal approximation that avoids collisions. Label placement using this method is often less optimal than label placement using the Simulated Annealing (SA) method. However, depending on the number of data points and the potential for label collisions, the Greedy process can be significantly faster.

SA
specifies the Simulated Annealing method for managing label collision. The SA method attempts to determine the global minimization-of-cost function, which is based on a simulated annealing algorithm. The resulting label placement is usually better than placement using the Greedy method. However, depending on the number of data points and the potential for label collisions, the SA method can be significantly slower.

Restriction For BANDPLOT and LINECHART, the SA method has no effect on the curve labels when the CURVELABELPOSITION= option specifies START or END.

Default GREEDY

LEGENDAREAMAX=*n*
specifies an integer that is interpreted as the maximum percentage of the overall graphics area that a legend can occupy.

n
specifies a positive integer.

Alias MAXLEGENDAREA=

Default 20

Range 0–100

Tip To turn off the legend, specify LEGENDAREAMAX=0. No warning is issued when the legend is turned off in this way.

LINEPATTERNOBSMAX=*n*
specifies the maximum the number of vertices for a patterned line. If the number of vertices exceeds the specified limit, the plot is not drawn.

n
specifies a positive integer.

Default 10000

Notes If the number of vertices exceeds the specified limit, the plot is not drawn and a note is written to the SAS log. In that case, increase LINEPATTERNOBSMAX= to the value suggested in the log message, or change the line pattern to SOLID.

This option applies only to graphs created with the Graph Template Language and with the ODS Graphics procedures.

LOESSOBSMAX=*n*
specifies an upper limit for the number of observations that can be used with a loess plot. If the number of observations of the loess plot exceeds the specified limit, the loess plot is not drawn.

For example, the following specifies that the most observations a loess plot can have is 1000.

```
LOESSOBSMAX=1000
```

Alias LOESSMAXOBS=

Default 5000

NBINSMAX=*n*
specifies the maximum number of bins that are processed for histograms. This option affects both computed and parameterized histograms, when available.

n
specifies a positive integer.

Default 10000

Interaction If you specify the number of bins in the histogram plot statement, that option is honored regardless of the NBINSMAX= option.

NOANTIALIAS
does not smooth jagged edges of components other than text in the graph.

NOBORDER
does not draw a border around the graph.

NOIMAGEMAP
specifies not to generate data tips.

NOSCALE
does not scale the components of a graph proportionally.

NOSCALEMARKERS
does not scale the markers with the graph size.

NOSUBPIXEL
never uses subpixel rendering.

NXYBINSMAX=*n*
specifies the maximum number of bins that are processed for heat maps. This option affects both computed and parameterized heat maps. The option applies to the product of the X and Y bins.

n
specifies a positive integer.

Default 100000

Interaction If you specify the number of bins in the heat map statement, that option is honored regardless of the NXYBINSMAX= option.

OBSMAX=*n*
specifies the maximum number of observations that are processed. If the number of observations in the data set exceeds the value specified for OBSMAX=, the procedure step terminates with a log message.

n
specifies a positive integer.

Alias MAXOBS=

Default 2 million observations

Interaction There are other ways to control the number of observations: CASDATALIMIT= system option, DATALIMIT= option in the CAS LIBNAME statement, and DATALIMIT= data set option. If the CAS data transfer limit is set lower than OBSMAX=, then OBSMAX= has no effect.

OUTPUTFMT=*file–type* | STATIC
specifies the output format used to generate image or vector graphic files. If the image or vector graphic format is not valid for the active output destination, the format is automatically changed to the default format for that destination.

file-type
is the image or vector graphic format to be generated. See “Supported Image Formats for Output Destinations” on page 58.

STATIC
uses the best quality static image format for the active output destination. This is the default output format.

> **TIP** The STATIC keyword can be used to reset the output format to its default state.

Default STATIC

See “Specifying the Image Format” on page 57

PANELCELLMAX=*n*
specifies the maximum number of cells in a graph panel where the number of cells is determined dynamically by classification variables. If the number of cells in the panel exceeds the specified limit, the panel is not drawn.

n
specifies a positive integer.

Default 1000

Note Graphs with DataPanel or DataLattice templates layouts are affected. In the ODS Graphics Procedures, this option affects graphs that are created with the SGPANEL procedure. If the value specified by the PANELCELLMAX= option is exceeded by any of these layouts, an empty graph is rendered and a warning message is issued.

Tip You can use the TOTALCELLMAX= option to control the total maximum number of cells in the graph.

POP
restores the most recently pushed settings from the stack. For each PUSH action, you can specify a POP request. ODS issues a warning if you specify POP without a corresponding PUSH. In that case, nothing is popped because nothing has been pushed.

PUSH
pushes the current ODS GRAPHICS settings to a stack. This feature enables you to temporarily save your custom settings in a stack and later restore those settings.

The pushed settings remain in the stack in the current SAS session until they are popped or the stack is emptied.

Interaction You can specify PUSH as many times as you like up to the limit that is defined by the STACKDEPTHMAX= option. You can also use STACKDEPTHMAX= to empty the stack. For more information, see "Managing the Stack Depth" on page 61.

Note Order of specification is important when using the PUSH and POP options. For more information, see "About PUSH and POP" on page 60.

Tip Use the SHOW option to show the current ODS GRAPHICS settings.

See "Temporarily Saving and Restoring ODS GRAPHICS Settings" on page 60

RESET
resets all *options* to their defaults.

RESET=ALL | *option*
resets one or all of the ODS GRAPHICS options to their defaults.

ALL
resets all of the ODS GRAPHICS options to their defaults.

option
specifies the name of an option to reset.

Option can be one of the following option names:

ANTIALIAS	IMAGENAME	OUTPUTFMT
ANTIALIASMAX	INDEX	PANELCELLMAX
ATTRPRIORITY	LABELMAX	SCALE
BORDER	LABELPLACEMENT	SCALEMARKERS
BYLINE	LEGENDAREAMAX	STACKDEPTHMAX
DATASKINMAX	LINEPATTERNOBSMAX	SUBPIXEL
DISCRETEMAX	LOESSOBSMAX	TIPMAX
DRILLTARGET	LOESSOBSMAX	TOTALCELLMAX
GROUPMAX	NBINSMAX	WIDTH
HEIGHT	NXYBINSMAX	
IMAGEMAP	OBSMAX	

SCALE
scales the components of a graph proportionally.

Default SCALE

SCALE=ON | OFF
specifies whether the content of a graph is scaled proportionally.

ON
scales the components of a graph proportionally.

Alias YES

OFF
does not scale the components of a graph proportionally.

Alias NO

Default SCALE=ON

SCALEMARKERS
scales the markers with the graph size.

Default SCALEMARKERS

SCALEMARKERS=ON | OFF
specifies whether the plot markers are to be scaled with the graph size. The scaling factor is based on the height of the graph cells and the height of the graph.

ON
scales the markers with the graph size.

Alias YES

OFF
does not scale the markers with the graph size.

Alias NO

Default SCALEMARKERS=ON

Restriction Scaling is done only if the graph contains multiple cells or single nested cells.

SHOW

writes the current ODS Graphics settings to the SAS log. This option enables you to verify which settings are in effect. The option is especially useful when you use the PUSH and POP options to restore settings. For more information, see "Temporarily Saving and Restoring ODS GRAPHICS Settings" on page 60.

If no options have been specified, then SHOW lists those options for which ODS currently knows the default values.

The following statement resets all settings and shows the default values.

```
ods graphics / reset=all show;
```

Here are the default values displayed in the SAS log:

```
ODS Graphics Settings
---------------------
Output format:                     STATIC
By line:                           NOBYLINE
Antialias:                         ON
Maximum Loess observations:        5000
Maximum stack depth:               1024
Stack depth:                       0
MaxObs:                            2000000
Maximum Histogram Bins:            10000
Maximum Heatmap Bins:              100000
Maximum Obs for Patterned Lines:   10000
Maximum Total Cells per BY-group:  2000
```

If you have specified the settings for one or more options, then SHOW includes those settings along with the defaults.

Order of specification is important when using the SHOW option. For example, the following statement shows the current settings and then sets the NOBORDER option.

```
ods graphics / show noborder;
```

However, the following statement sets the NOBORDER option and then shows the settings. The NOBORDER setting is shown in the log along with the other settings that are in effect.

```
ods graphics / noborder show;
```

The following statement resets all settings. It then sets the image width and shows the default settings along with the specified width.

```
ods graphics / reset=all width=5in show;
```

Here are the default values plus the image width, as displayed in the SAS log:

```
ODS Graphics Settings
---------------------
Output format:                     STATIC
By line:                           NOBYLINE
Antialias:                         ON
Maximum Loess observations:        5000
Image width:                       5in
Maximum stack depth:               1024
Stack depth:                       0
MaxObs:                            2000000
Maximum Histogram Bins:            10000
```

```
Maximum Heatmap Bins:              100000
Maximum Total Cells per BY-group:  2000
```

Tip If you have specified the settings for some options but want to see the default values without losing your specified settings, issue the following two statements. The first statement pushes your specified settings, resets all settings, and then lists options for which ODS currently knows the default values. The second statement restores your previous settings.

```
ods graphics / push reset=all show;
ods graphics / pop;
```

STACKDEPTHMAX=*n*

specifies the maximum stack depth for PUSH and POP requests. The stack is used to temporarily store ODS GRAPHICS settings when you issue PUSH requests. PUSH saves the current settings to the stack and increments the stack depth. POP restores the most recently saved settings from the stack and decrements the stack depth.

n

specifies a positive integer.

If *n* is less than the current stack depth, then the stack is popped until its depth equals *n*. Popping the stack does not affect other option settings.

Defaults 1024 is the default maximum depth

0 is the default depth

Tips To empty the stack and then reset it to the default maximum depth, issue the following statement:

```
ods graphics / stackdepthmax=0 reset=stackdepthmax;
```

You can use any of the following commands to reset the stack to its default maximum depth:

```
reset=stackdepthmax
reset=all
reset
stackdepthmax=1024
```

See “Managing the Stack Depth” on page 61

SUBPIXEL

always uses subpixel rendering, when applicable, for rendering lines and bars.

Default Subpixel rendering is always enabled for vector-graphics output. It is enabled by default for image output, unless the graph contains a scatter plot or a scatter-plot matrix. In those cases, subpixel rendering is disabled by default.

Requirement Antialiasing must be enabled for this option to have any effect. Antialiasing is enabled by default. To re-enable antialiasing, use the ANTIALIAS or ANTIALIAS=ON option in the ODS GRAPHICS statement.

Note Subpixel rendering is built in for the SGPIE and SGMAP procedures, and the SUBPIXEL option has no effect on those procedures.

Tip For a large amount of data, antialiasing is disabled when the number of observations exceeds the default maximum of 4000 observations. In that case, subpixel rendering is also disabled. To increase the maximum, use the ANTIALIASMAX= option in the ODS GRAPHICS statement.

See "Subpixel Rendering" in *SAS ODS Graphics: Procedures Guide*

SUBPIXEL=ON | OFF
specifies whether subpixel rendering should be used for rendering ODS Graphics. Subpixel rendering produces smoother curves and more precise bar spacing.

ON
always uses subpixel rendering, when applicable, for rendering lines and bars.

Alias YES

OFF
never uses subpixel rendering.

Alias NO

Default Subpixel rendering is always enabled for vector-graphics output. It is enabled by default for image output, unless the graph contains a scatter plot or a scatter-plot matrix. In those cases, subpixel rendering is disabled by default.

Requirement Antialiasing must be enabled for this option to have any effect. Antialiasing is enabled by default. To re-enable antialiasing, use the ANTIALIAS or ANTIALIAS=ON option in the ODS GRAPHICS statement.

Note Subpixel rendering is built in for the SGPIE and SGMAP procedures, and the SUBPIXEL option has no effect on those procedures.

Tip For a large amount of data, antialiasing is disabled when the number of observations exceeds the default maximum of 4000 observations. In that case, subpixel rendering is also disabled. To increase the maximum, use the ANTIALIASMAX= option in the ODS GRAPHICS statement.

See "Subpixel Rendering" in *SAS ODS Graphics: Procedures Guide*

TIPMAX=*n*
specifies the maximum number of distinct data tip areas allowed before data tips are disabled. For example, if there are more than 400 points in a scatterplot, and TIPMAX=400, then no data tips appear. The default maximum value is 500.

Note: The TIPMAX= option specifies the maximum number of data-tip areas allowed before data tips are disabled. This threshold is applied separately for each plot. The default is 500. If any plot in a graph contains more than 500 data-tip areas, data tips are disabled for that plot. Data tips are enabled for the remaining plots in the graph.

n
specifies a positive integer.

Default 500

TOTALCELLMAX=*n*
specifies the maximum number of total cells in a graph where the number of cells is determined dynamically by classification variables.

If the number of cells exceeds the specified limit, the graph is not drawn. This option is useful when multiple classification variables or large amounts of data result in a large number of cells.

n
specifies a positive integer.

Default 2000

Notes If the number of cells exceeds the limit specified for TOTALCELLMAX=, the graph is not drawn and an error message is written to the SAS log. The error message includes a suggested value for TOTALCELLMAX=. You can use the TOTALCELLMAX= option to increase the maximum number of cells. Note, however, that the processing time to render the graph increases with an increase in the maximum number of cells.

Graphs with DataPanel or DataLattice templates layouts are affected. In the ODS Graphics Procedures, this option affects graphs that are created with the SGPANEL procedure.

Tip You can use the PANELCELLMAX= option to control the maximum number of cells in a graph panel.

WIDTH=*dimension*
specifies the width of any graph.

dimension
is a nonnegative number followed by one of these units of measure:

Table 4.3 *Units of Measure for Dimension*

cm	Centimeters
in	Inches
mm	Millimeters
pct or %	Percentage
pt	Point size (72 points = 1 inch)
px	Pixels

Default Determined by the software, or the value of the DesignWidth= option in a STATGRAPH template. Typically, this value is 640px.

Tips If only the WIDTH= option is specified, then the default aspect of the graph is maintained.

When you change the size of a graph, by default graph elements such as markers, lines, text, and titles are scaled proportionally. This scaling takes effect even when you specify dimensions for the graph elements. To disable the scaling, specify the NOSCALE option.

Details

Using the ODS GRAPHICS Statement

You can enable ODS Graphics by using one of the following equivalent statements:

```
ods graphics on;
ods graphics;
```

When you specify one of these statements before your procedure invocation, Base, SAS/STAT, SAS/ETS, and SAS/QC procedures support ODS Graphics, either by default, or when you specify procedure options for requesting particular graphs.

To disable ODS Graphics, specify the following statement:

```
ods graphics off;
```

Note: ODS Graphics is ON by default for these procedures: SGMAP, SGPANEL, SGPIE, SGPLOT, SGRENDER, and SGSCATTER. For other products, the initial state of ODS Graphics is determined by a SAS Registry setting.

Using the ODS GRAPHICS Statement for Batch Jobs

To generate ODS Graphics output in Linux batch jobs, you must set the DISPLAY system option before creating the output. To set the display, enter the following command:

```
export DISPLAY=<ip_address>:0
```

The *ip_address* is the TCP/IP address, or the name of a Linux terminal. Usually, the IP address of the Linux system where SAS is running would be used. If you do not set the DISPLAY variable, then you get an error message in the SAS log.

Specifying and Resetting the Image Name

Specifying the Image Name

For ODS Graphics output, by default, the ODS object name is used as the “root” name for the image output file. The following example creates a GIF image named REGPLOT:

```
ods graphics / imagename="regplot" outputfmt=gif;
```

The assigned name REGPLOT is treated as a "root" name and the first output created is named REGPLOT. Subsequent graphs are named REGPLOT1, REGPLOT2, and so on, with an increasing index counter. This numbering can be reset with the RESET=INDEX option.

Resetting the Image Name

The RESET=INDEX option enables you to reset the file name numbering sequence.

For a usage example, suppose that you are developing a paper or a presentation and it takes several submissions to get the desired output. You can use the RESET or RESET=INDEX option to force each output to replace itself:

```
ods graphics / reset=index ... ;
```

This specification causes all subsequent images to be created with the default or current image name.

When specifying this option, you can also specify the value for the index counter. The value that you specify determines the suffix for the next subsequent image. For example:

```
ods graphics / reset=index(100) imagename="MyName";
```

The next graph that you produce is named MYNAME100.

This feature is useful for creating animated graphics. For example, for a sequence of 100 images, you might begin with the following statement:

```
ods graphics / reset=index(1) imagename="MyName";
```

In the example, your program produces 100 images named MYNAME1, MYNAME2, ..., MYNAME100. If you later add more images to the animation, you might submit the following:

```
ods graphics / reset=index(101) imagename="MyName";
```

The next generated image is named MYNAME101.

Substituting BY Values in the Image Name

If a BY statement is in effect for the data, then you can uniquely name image files based on BY groups. You do this by inserting any of the #BY* substitution items into the image file name. You can specify BY variable values or BY variable names for IMAGENAME=. You can combine the #BY* substitution item with other text. To do this, insert the #BY* item in the specified text string at the position where you want the substitution text to appear.

As explained in the IMAGENAME= description, the file extension is automatically generated based on the OUTPUTFMT= option.

Here are descriptions of the #BY* substitution items:

#BYVAL*n* | **#BYVAL(*BY-variable-name*)**
substitutes the current value of the specified BY variable for #BYVAL in the file name string. Specify the variable with one of these:

n
specifies a variable by its position in the BY statement. You must provide the position even if your BY statement contains only one variable. Otherwise, no substitution occurs.

BY-variable-name
specifies a variable from the BY statement by its name. *BY-variable-name* is not case sensitive.

Examples The following example specifies the first variable in the BY statement. The value of that variable is used for the image file name.

```
imagename="#byval1"
```

The following example specifies the second variable in the BY statement preceded by the text *sales_*:

```
imagename="sales_#byval2"
```

The following example specifies the YEAR variable in the BY statement. The value of that variable is used for the image file name:

```
imagename="#byval(year)"
```

The following example specifies the YEAR variable in the BY statement followed with the text *_sales*:

```
imagename="#byval(year)_sales"
```

#BYVAR*n* | #BYVAR(*BY-variable-name*)
substitutes the name of the BY variable or the label associated with the variable (whatever the BY line would normally display) for #BYVAR in the file name string. Specify the variable with one of these:

n
specifies a variable by its position in the BY statement. You must provide the position even if your BY statement contains only one variable. Otherwise, no substitution occurs.

BY-variable-name
specifies a variable from the BY statement by its name. *BY-variable-name* is not case sensitive.

Examples The following example specifies the first variable in the BY statement. The name of that variable is used for the image file name.

```
imagename="#byvar1"
```

The following example specifies the second variable in the BY statement preceded by the text *sales_*:

```
imagename="sales_#byvar2"
```

The following example specifies the YEAR variable in the BY statement. The name of that variable is used for the image file name:

```
imagename="#byvar(year)"
```

The following example specifies the YEAR variable in the BY statement followed with the text *_sales*:

```
imagename="#byvar(year)_sales"
```

Note: When using this feature, consider any file name length restrictions that might apply to your platform host. The file name length can vary greatly depending on the input data.

Specifying the Image Format

Each ODS destination uses a default format for its output. You can use the OUTPUTFMT= option in the ODS GRAPHICS statement to change the output format.

Note: Unless you have a special requirement for changing the image format, we recommend that you not change it. The default PNG format or default vector graphic format is far superior to other formats, such as GIF, because either supports transparency and a large number of colors. Also, PNG and vector graphics images require much less disk storage space than JPEG or TIFF formats.

If you want to generate vector graphics images, you can use the following OUTPUTFMT= values for each destination:

Table 4.4 *Generating Vector Graphics Output with ODS*

ODS Destination	**OUTPUTFMT=*value***
ODS EPUB	SVG graphics images are generated by default
ODS destination for Excel	SVG graphics images are generated by default
ODS HTML5	SVG graphics images are generated by default
ODS PDF	Vector graphics images are generated by default
ODS destination for PowerPoint	SVG graphics images are generated by default
ODS RTF	`OUTPUTFMT=EMF`
ODS Measured RTF	`OUTPUTFMT=EMF`
ODS destination for Word	SVG graphics images are generated by default

There are cases in which vector graphics images cannot be generated for the output format that you specify. In those cases, a PNG image is generated instead. The following table lists the cases in which vector graphics images cannot be generated for the EMF, PDF, and SVG output formats.

Table 4.5 *Cases in Which Vector Graphics Are Not Supported*

	Supported (YES or NO)		
Case	**EMF**	**PDF**	**SVG**
3-D bivariate histograms	NO	NO	NO
Data skins and transparency[1]	NO	YES	YES
Gradient-fill contour plots[2]	NO	NO	NO

Case	Supported (YES or NO)		
	EMF	PDF	SVG
Images and transparency[1]	NO	YES	YES
Surface plots	NO	NO	NO

1 Data transparency or fill transparency.

2 GTL CONTOURPLOTPARM statement with GRADIENT, LINEGRADIENT, or LABELEDLINEGRADIENT specified for CONTOURTYPE=.

Note: The SGMAP procedure does not support vector-based output.

Supported Image Formats for Output Destinations

The following table lists all of the supported image formats for some ODS output destinations.

Table 4.6 *Supported Image Formats for Output Destinations*

Output Destination	Default Image Format	Supported Image Formats
EPUB	SVG	GIF, JPEG, JPG, PNG, SVG
ODS destination for Excel	SVG	EMF, JPEG, JPG, PNG, SVG
HTML5	SVG	GIF, JPEG, JPG, PNG, SVG
PDF	Native PDF	Native PDF, GIF, JPEG, JPG, PNG
ODS destination for PowerPoint	SVG	BMP, EMF, GIF, JPEG, JPG, PNG, SVG, TIFF
RTF and Measured RTF	EMF	EMF, JFIF, JPEG, JPG, PNG
ODS destination for Word	SVG	BMP, EMF, GIF, JPEG, JPG, PNG, SVG, TIFF

Description of Supported Image Formats

The following table provides descriptions of the supported image formats for ODS output destinations

Table 4.7 Description of Supported Image Formats

Image Format	Description
BMP (Microsoft Windows Device Independent Bitmap)	Supports color-mapped and true color images that are stored as uncompressed or run-length encoded data. BMP was developed by Microsoft Corporation.
CGM (Computer Graphics Metafile)	A free and open international standard file format for 2-D vector graphics, raster graphics, and text. This format is defined by ISO/IEC 8632.
DIB (Microsoft Windows Device Independent Bitmap)	See the description of BMP.
EMF (Enhanced Metafile Format)	Supports standard Enhanced Metafile Format.
EMF Plus (Enhanced Metafile Format Plus Extensions)	Supports Enhanced Metafile Plus Extensions that provides additional functionality, such as support of RGBA colors.
EMF Dual (Enhanced Metafile Format and Enhanced Metafile Format Plus Extensions)	Produces both EMF and EMF Plus formats simultaneously in the same output.
GIF (Graphics Interchange Format)	Supports only color-mapped images. GIF is owned by CompuServe, Inc.
JFIF (JPEG File Interchange Format)	Supports JPEG image compression. JFIF software is developed by the Independent JPEG Group.
JPEG or JPG (Joint Photographic Experts Group)	A file format that is used for storing noninteractive images.
PBM (Portable Bitmap Utilities)	Supports gray-scale, color, RGB, and bitmap files. The Portable Bitmap Utilities are a set of free utility programs that were developed primarily by Jef Poskanzer.
PDF (Portable Document Format)	A file format for electronic distribution and exchange of documents.
PNG (Portable Network Graphic)	Supports true color, gray-scale, and 8-bit images.
STATIC	Chooses the best image format for the current ODS destination.

Image Format	Description
SVG (Scalable Vector Graphics)	Is an XML language for describing two-dimensional vector graphics.
TIFF (Tagged Image File Format)	Internally supports a number of compression types and image types, including bitmapped, color-mapped, gray-scaled, and true color. TIFF was developed by Aldus Corporation and Microsoft Corporation and is used by a wide variety of applications (available if licensed).
XBM	X Window Bitmap
XPM	X Window Pixmap

Temporarily Saving and Restoring ODS GRAPHICS Settings

About PUSH and POP

Although you can use the RESET option to restore the default ODS GRAPHICS settings, there might be times when you want to save your current custom settings and later restore them. ODS enables you to temporarily store your custom settings in a stack created for this purpose, perform some other task with different settings, and then restore the previous settings.

The PUSH option saves the current ODS GRAPHICS settings to the stack and increments the stack depth. The POP option restores the most recently stored settings from the stack and decrements the stack depth.

This feature is useful when you write macros. Within a macro you can PUSH at the start of the macro and POP at the end. This enables your macro to have custom ODS GRAPHICS behaviors without affecting the calling environment.

You can specify PUSH as many times as you like up to the limit that is defined by the STACKDEPTHMAX= option. The pushed settings remain in the stack in the current SAS session until they are popped or the stack is emptied. For more information, see “Managing the Stack Depth” on page 61. For each PUSH option, you can specify a POP option. ODS issues a warning if you specify POP without a corresponding PUSH. In that case, nothing is popped because nothing has been pushed to the stack.

Order of specification is important when using the PUSH option. For example, the following statement pushes the NOBORDER option to the stack along with any other custom settings that are in effect.

```
ods graphics / noborder push;
```

A subsequent POP option restores the pushed settings including NOBORDER.

However, the following statement pushes the current custom settings and then sets the NOBORDER option.

```
ods graphics / push noborder;
```

Here, the subsequent POP option restores whatever border setting was in effect when the PUSH option was specified.

> **TIP** Use the SHOW option to show the ODS GRAPHICS settings that are currently in effect.

Settings That Can Be Pushed

The PUSH and POP options apply to all ODS GRAPHICS options except the following: PUSH, POP, RESET=INDEX, and SHOW.

How Code Errors Affect the PUSH Operation

If the ODS GRAPHICS statement contains a syntax error, then the PUSH option is ignored.

For example, the PUSH option is ignored in the following statement:

```
ods graphics / antialias=bogus push;
```

A syntax error (BOGUS) in ANTIALIAS causes the parser to ignore the remaining options. However, a simple semantics error does not prevent the remaining options from being handled. In the following statement, the PUSH option is honored.

```
ods graphics / antialiasmax=-1 push;
```

In this statement, ANTIALIASMAX= –1 is invalid. The option expects a zero or a positive integer. In this case, a warning is issued to the log, but the PUSH occurs.

Note: Syntax errors in your code can have other unexpected results that are not described here.

Managing the Stack Depth

By default, the stack supports up to 1024 pushes. You can change the default by using the STACKDEPTHMAX= option.

If the specified STACKDEPTHMAX= value is less than the current stack depth, then the stack is popped until its depth equals the specified value. Popping the stack does not affect other option settings.

If you want to empty the stack, issue the following statement:

```
ods graphics / stackdepthmax=0 reset=stackdepthmax;
```

This statement first empties the stack of all PUSH requests and then restores the stack size to 1024.

ODS TEXT= Statement

Inserts text into your ODS output.

Category: ODS: Output Control

Tip: The ODS TEXT= statement is sent only to output destinations that are open. Therefore, it must be specified after an ODS destination statement.

Syntax

ODS TEXT= '*text-string*'

Required Argument

text-string
specifies the text to insert into your output. This text is sent to all open output destinations.

Requirement You must enclose '*text-string*' in quotes.

Tip The UserText style element controls text specified with the TEXT= statement.

Examples

Example 1: Adding Text to Multiple Destinations

Features:
ODS HTML statement
ODS PDF statement
ODS RTF statement
ODS TEXT= statement
PROC TEMPLATE:
DEFINE STYLE statement
PARENT= statement
STYLE statement
PROC PRINT

Details

The following example uses a single ODS TEXT= statement to add text to PDF, HTML, and traditional RTF output. PROC TEMPLATE modifies the UserText style element that controls the font style, font color, and other attributes of the text that the ODS TEXT= statement adds.

Program

```
options obs=10;

proc template;
   define style Styles.MyStyle;
   parent=styles.htmlblue;
   style usertext from usertext /
      foreground=red;
   end;
run;

ods html file="text.html" style=Styles.MyStyle ;
ods pdf file="text.pdf" startpage=never notoc style=Styles.MyStyle ;
ods rtf file="text_trad.rtf" style=Styles.MyStyle ;

title "January Orders ";
footnote " For All Employees";

ods text="My Text 1";
ods text="My Text 2";

proc print data=exprev;
run;

ods text="My Text 3";

ods pdf close;
ods rtf close;
ods html close;
title;
footnote;

proc template;
   delete Styles.MyStyle ;
   run;
```

Program Description

```
options obs=10;
```

Create the Styles.MyStyle style template. The Styles.MyStyle style templates modifies the Usertext style element to change the font color of text created by the TEXT= statement to red.

```
proc template;
```

```
   define style Styles.MyStyle;
   parent=styles.htmlblue;
   style usertext from usertext /
      foreground=red;
   end;
run;
```

Send output to multiple ODS destinations. The following statements open the HTML, PDF, and RTF destinations. The STYLE= options specifies that the style Styles.MyStyle is applied to the output.

```
ods html file="text.html" style=Styles.MyStyle ;
ods pdf file="text.pdf" startpage=never notoc style=Styles.MyStyle ;
ods rtf file="text_trad.rtf" style=Styles.MyStyle ;
```

Add a title and footnote. The TITLE and FOOTNOTE statements specify the title and footnote. You must place the TITLE and FOOTNOTE statements before the ODS TEXT= statements.

```
title "January Orders ";
footnote " For All Employees";
```

Add text strings before the output is printed. The ODS TEXT= statements add the text strings "My Text 1" and "My Text 2" The text is added to the output before the data set is printed.

```
ods text="My Text 1";
ods text="My Text 2";
```

Print the data set Exprev. The PRINT procedure prints the Exprev data set.

```
proc print data=exprev;
run;
```

Add a third text string after the data set. The third ODS TEXT= statement adds the text string "My Text 3" after the data set is printed.

```
ods text="My Text 3";
```

Close the RTF, HTML, and PDF destinations and remove the titles and footnotes. The ODS RTF CLOSE statement closes the RTF destination. The ODS PDF CLOSE statement closes the PDF destination. The ODS HTML CLOSE statement closes the HTML destination. The TITLE and FOOTNOTE statements remove any titles and footnotes previously specified.

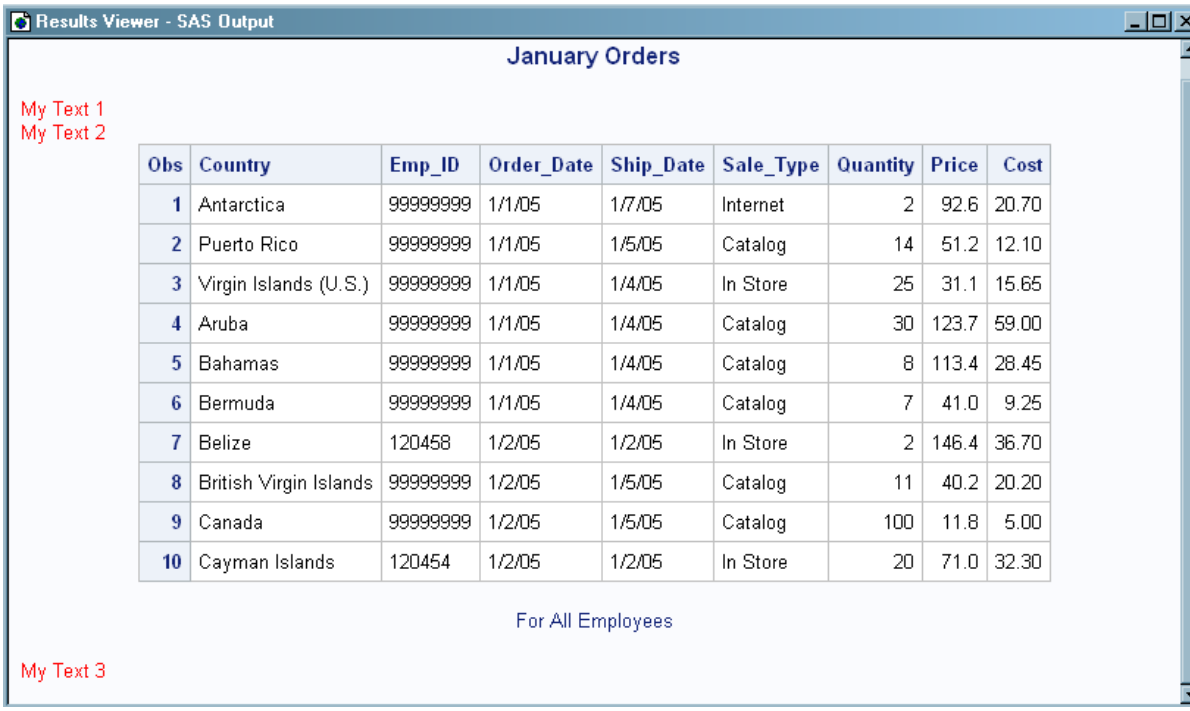
```
ods pdf close;
ods rtf close;
ods html close;
title;
footnote;
```

After your output is created, you can delete the Styles.MyStyle style template. The DELETE statement deletes the Styles.MyStyle style template.

```
proc template;
   delete Styles.MyStyle ;
   run;
```

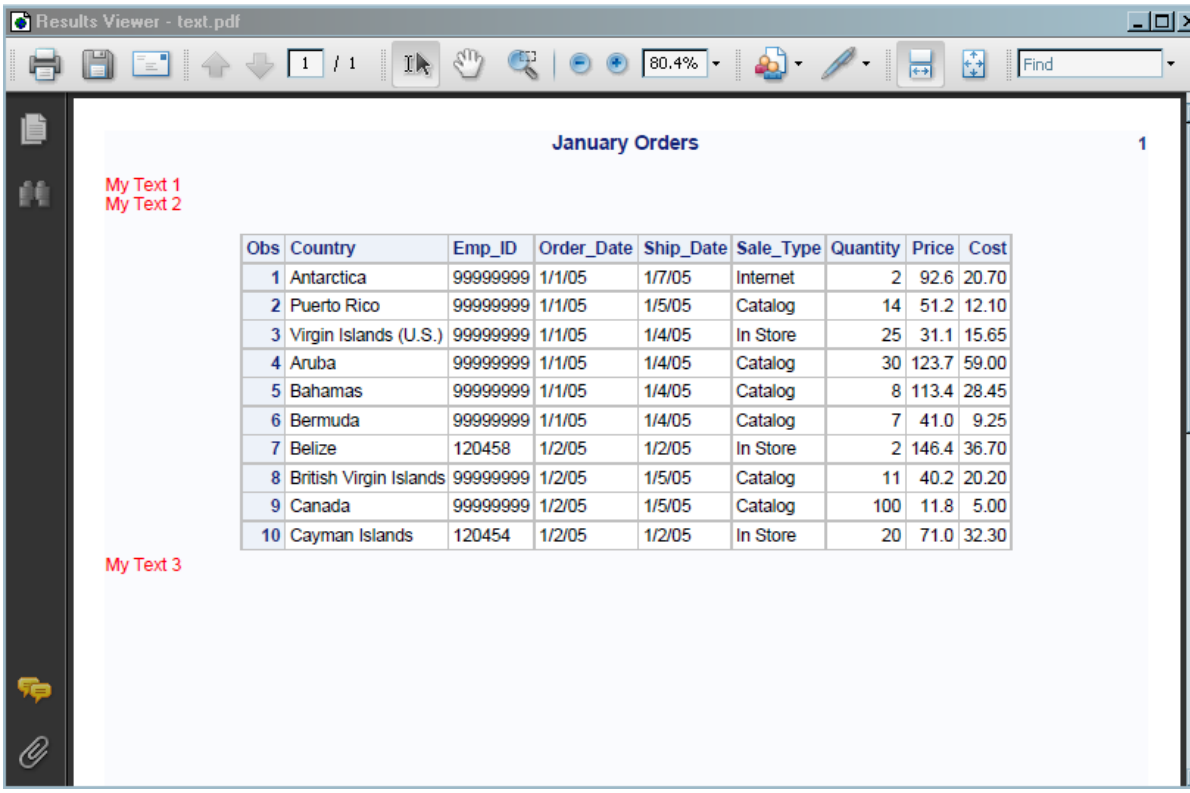
Output

Output 4.1 *HTML Output with Text Added*



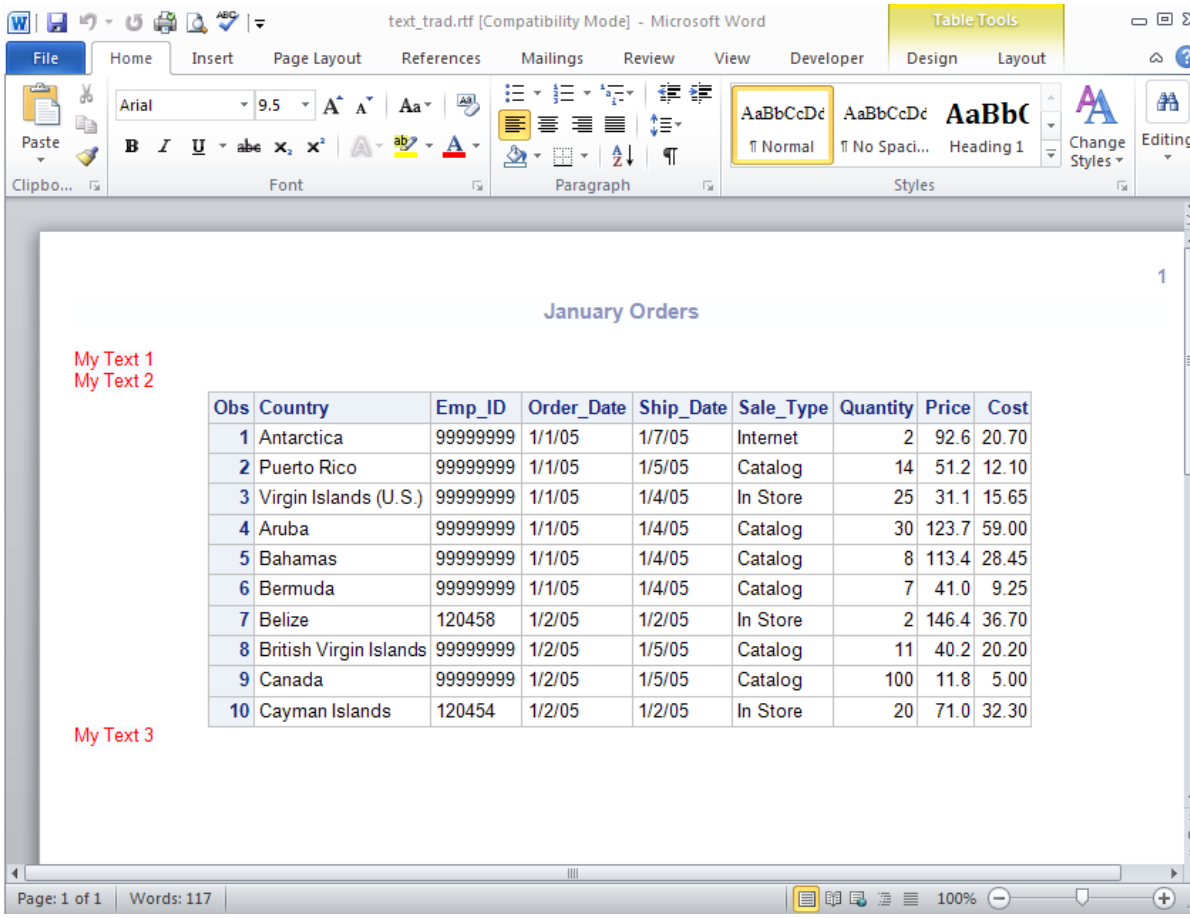
Obs	Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
1	Antarctica	99999999	1/1/05	1/7/05	Internet	2	92.6	20.70
2	Puerto Rico	99999999	1/1/05	1/5/05	Catalog	14	51.2	12.10
3	Virgin Islands (U.S.)	99999999	1/1/05	1/4/05	In Store	25	31.1	15.65
4	Aruba	99999999	1/1/05	1/4/05	Catalog	30	123.7	59.00
5	Bahamas	99999999	1/1/05	1/4/05	Catalog	8	113.4	28.45
6	Bermuda	99999999	1/1/05	1/4/05	Catalog	7	41.0	9.25
7	Belize	120458	1/2/05	1/2/05	In Store	2	146.4	36.70
8	British Virgin Islands	99999999	1/2/05	1/5/05	Catalog	11	40.2	20.20
9	Canada	99999999	1/2/05	1/5/05	Catalog	100	11.8	5.00
10	Cayman Islands	120454	1/2/05	1/2/05	In Store	20	71.0	32.30

Output 4.2 *PDF Output with Text Added*



Obs	Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
1	Antarctica	99999999	1/1/05	1/7/05	Internet	2	92.6	20.70
2	Puerto Rico	99999999	1/1/05	1/5/05	Catalog	14	51.2	12.10
3	Virgin Islands (U.S.)	99999999	1/1/05	1/4/05	In Store	25	31.1	15.65
4	Aruba	99999999	1/1/05	1/4/05	Catalog	30	123.7	59.00
5	Bahamas	99999999	1/1/05	1/4/05	Catalog	8	113.4	28.45
6	Bermuda	99999999	1/1/05	1/4/05	Catalog	7	41.0	9.25
7	Belize	120458	1/2/05	1/2/05	In Store	2	146.4	36.70
8	British Virgin Islands	99999999	1/2/05	1/5/05	Catalog	11	40.2	20.20
9	Canada	99999999	1/2/05	1/5/05	Catalog	100	11.8	5.00
10	Cayman Islands	120454	1/2/05	1/2/05	In Store	20	71.0	32.30

Output 4.3 *Traditional RTF Output with Text Added*



Obs	Country	Emp_ID	Order_Date	Ship_Date	Sale_Type	Quantity	Price	Cost
1	Antarctica	99999999	1/1/05	1/7/05	Internet	2	92.6	20.70
2	Puerto Rico	99999999	1/1/05	1/5/05	Catalog	14	51.2	12.10
3	Virgin Islands (U.S.)	99999999	1/1/05	1/4/05	In Store	25	31.1	15.65
4	Aruba	99999999	1/1/05	1/4/05	Catalog	30	123.7	59.00
5	Bahamas	99999999	1/1/05	1/4/05	Catalog	8	113.4	28.45
6	Bermuda	99999999	1/1/05	1/4/05	Catalog	7	41.0	9.25
7	Belize	120458	1/2/05	1/2/05	In Store	2	146.4	36.70
8	British Virgin Islands	99999999	1/2/05	1/5/05	Catalog	11	40.2	20.20
9	Canada	99999999	1/2/05	1/5/05	Catalog	100	11.8	5.00
10	Cayman Islands	120454	1/2/05	1/2/05	In Store	20	71.0	32.30

Example 2: Adding Text That Imitates a System Title

Features:

ODS PDF statement option
- FILE=
- NOTOC
- STARTPAGE=NO
- STYLE=

ODS NOPROCTITLE statement

ODS HTML

ODS TEXT statement

PROC TEMPLATE

OPTIONS statement

PROC MEANS

PROC PRINT

TITLE statement

Details

SAS titles and footnotes are displayed once per page in the PDF destination. Therefore, when the STARTPAGE= option is set to NO (OFF) and output from more than one procedure or DATA _NULL_ step is routed to ODS PDF, only the first set of titles and footnotes are written to the output file. This example shows how to use the TEXT= option to mimic an interim title, displayed above the second procedure output. PROC TEMPLATE is used to create a custom style template that mimics the style of the Systemtitle element. Systemtitle is the style element that controls the appearance of titles.

This example requires some knowledge of style templates, style elements, and style attributes.

- For complete documentation about styles, see “Understanding Styles, Style Elements, and Style Attributes” on page 441.
- For a table of style attributes, see “Style Attributes Tables” on page 808.
- For a table of style elements affecting pages, see “Style Elements Affecting Pages” on page 779.
- For a table of style elements affecting the table of contents, see “Style Elements Affecting Tables of Contents” on page 778.

Program

```
options nodate;


proc template;
   define style styles.mimictitle;
   parent=styles.pearl;
   class usertext from systemtitle /
     just=c;
   end;
run;

ods pdf file="/your-file-path/file.pdf" notoc startpage=no
style=styles.mimictitle;


title "Overriding the Default Procedure Title";

ods noproctitle;
proc means data=sashelp.cars;
   class cylinders;
   var mpg_city mpg_highway;
run;



ods text="My Custom PROC PRINT Output Title";
```

```
proc print data=sashelp.cars noobs;
   where mpg_highway gt 45;
run;


ods pdf close;
ods html;

proc template;
delete Styles.Mimictitle;
run;
```

Program Description

Set the SAS system options.

```
options nodate;
```

Create a custom style template. The Usertext style element controls the appearance of user text, which includes text specified by the TEXT= option. The Systemtitle style element controls the appearance of the default SAS titles, which include the text specified by the TITLE= statement. This PROC TEMPLATE step creates a new style named Styles.Mimictitle, which contains all of the style elements and style attributes that Styles.Pearl contains. However, the CLASS statement modifies the Usertext style element to produce center-justified text, just as the Systemtitle style element does. This ensures that text specified by the TEXT= option will look the same as the text specified by the TITLE= statement.

```
proc template;
   define style styles.mimictitle;
   parent=styles.pearl;
   class usertext from systemtitle /
     just=c;
   end;
run;
```

Open the PDF destination and specify the ODS PDF statement options. The ODS PDF statement opens the PDF destination and the FILE= option specifies the PDF filename. The NOTOC option specifies that no table of contents is created. The STARTPAGE=NO option specifies that no new pages are inserted at the beginning of each procedure, or within certain procedures, even if new pages are requested by the procedure code. The STYLE= option specifies the style to use for the output, which is Styles.Mimictitle in this example.

```
ods pdf file="/your-file-path/file.pdf" notoc startpage=no
style=styles.mimictitle;
```

Specify a title.

```
title "Overriding the Default Procedure Title";
```

Create the MEANS procedure output and suppress the procedure title. The ODS NOPROCTITLE statement suppresses the writing of the procedure title that produces the results.

```
ods noproctitle;
proc means data=sashelp.cars;
   class cylinders;
   var mpg_city mpg_highway;
```

```
run;
```

Specify the text to use as the second procedure title. The TEXT= option specifies the text that appears above the PRINT procedure output. Because the custom style Styles.Mimictitle was specified in the ODS PDF statement, this text will look like a title specified by the TITLE= statement.

```
ods text="My Custom PROC PRINT Output Title";
```

Create the PRINT procedure output.

```
proc print data=sashelp.cars noobs;
   where mpg_highway gt 45;
run;
```

Close the PDF destination and open the HTML destination. The ODS PDF CLOSE statement closes the PDF destination and all of the files that are associated with it. You must close the destinations before you can view the output with a browser or before you can send the output to a physical printer. The ODS HTML statement opens the HTML destination and returns SAS to the default ODS destination.

```
ods pdf close;
ods html;
```

After your output is produced, you can remove the custom style. The DELETE statement in PROC TEMPLATE removes the custom style.

```
proc template;
delete Styles.Mimictitle;
run;
```

PDF Output

The following image shows the text “My Custom PROC PRINT Output Title” above the PROC PRINT table in the same style as the title.

Output 4.4 *PDF Output with Custom Procedure Title*

Overriding the Default Procedure Title

Cylinders	N Obs	Variable	Label	N	Mean	Std Dev	Minimum	Maximum
3	1	MPG_City MPG_Highway	MPG (City) MPG (Highway)	1 1	60.0000000 66.0000000	. .	60.0000000 66.0000000	60.0000000 66.0000000
4	136	MPG_City MPG_Highway	MPG (City) MPG (Highway)	136 136	24.9411765 31.8897059	5.2093430 4.8544333	18.0000000 23.0000000	59.0000000 51.0000000
5	7	MPG_City MPG_Highway	MPG (City) MPG (Highway)	7 7	19.8571429 26.8571429	0.8997354 1.0690450	18.0000000 25.0000000	21.0000000 28.0000000
6	190	MPG_City MPG_Highway	MPG (City) MPG (Highway)	190 190	18.5157895 25.5526316	1.7630130 3.1697760	14.0000000 17.0000000	23.0000000 32.0000000
8	87	MPG_City MPG_Highway	MPG (City) MPG (Highway)	87 87	15.8735632 21.8850575	1.8912565 3.5777909	10.0000000 12.0000000	18.0000000 28.0000000
10	2	MPG_City MPG_Highway	MPG (City) MPG (Highway)	2 2	11.0000000 16.5000000	1.4142136 4.9497475	10.0000000 13.0000000	12.0000000 20.0000000
12	3	MPG_City MPG_Highway	MPG (City) MPG (Highway)	3 3	12.6666667 19.0000000	0.5773503 0	12.0000000 19.0000000	13.0000000 19.0000000

My Custom PROC PRINT Output Title

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Honda	Civic Hybrid 4dr manual (gas/electric)	Hybrid	Asia	Front	$20,140	$18,451	1.4
Honda	Insight 2dr (gas/electric)	Hybrid	Asia	Front	$19,110	$17,911	2.0
Toyota	Prius 4dr (gas/electric)	Hybrid	Asia	Front	$20,510	$18,926	1.5
Volkswagen	Jetta GLS TDI 4dr	Sedan	Europe	Front	$21,055	$19,638	1.9

ODS TRACE Statement

Writes to the SAS log a record of each output object that is created, or suppresses the writing of this record.

Category: ODS: Output Control

Default: OFF

Example: "Creating Merged Data Sets" in *SAS Output Delivery System: User's Guide*

Syntax

ODS TRACE ON </*options*> ;

ODS TRACE OFF;

Required Arguments

OFF

turns off the writing of the trace record.

Alias NO

ON
turns on the writing of the trace record.

Aliases OUTPUT

YES

Optional Arguments

DOM<="*external-file*">
specifies that the ODS document object model is written to the SAS log or an external file.

external-file
is the name of an external output file.

Requirement You must enclose *external-file* in quotation marks.

See For complete documentation about the ODS document object model, see "Working with the ODS Document Object Model" in *SAS Output Delivery System: Advanced Topics*.

EXCLUDED
includes, in the trace record, information for excluded output objects.

Example "Example 2: Conditionally Selecting Output Objects" on page 30

LABEL
includes the label path for the output object in the record. You can use a label path anywhere that you can use a path.

Tip This option is helpful for users who are running a localized version of SAS, because the labels are translated from English to the local language. The names and paths of output objects are not translated because they are part of the syntax of the Output Delivery System.

Details

Contents of the Trace Record

ODS produces an output object by combining data from the data component with a table template. The trace record provides information about the data component, the table template, and the output object. By default, the record that the ODS TRACE statement produces contains these items:

Name
is the name of the output object. You can use the name to reference this output object and others with the same name. For details about how to reference an output object, see "How ODS Determines the Destinations for an Output Object" in *SAS Output Delivery System: User's Guide*. For example, you could use this name in an ODS OUTPUT statement to make a data set from the output object. You could also use this name in an ODS SELECT or an ODS EXCLUDE statement.

> **TIP** The name is the rightmost part of the path that appears in the trace record.

Label
: briefly describes the contents of the output object. This label also identifies the output object in the Results window.

Data name
: is the name of the data component that was used to create this output object. The data name appears only if it differs from the name of the output object.

Data label
: describes the contents of the data.

Template
: is the name of the table template that ODS uses to format the output object. You can modify this definition with PROC TEMPLATE. See the "EDIT Statement" on page 536 for more information.

Path
: is the path of the output object. You can use the path to reference this output object. For example, you could use the path in the ODS OUTPUT statement to make a data set from the output. You could also use the path in an ODS SELECT or an ODS EXCLUDE statement.

The LABEL option modifies the trace record by including the label path for the object in the record. See the discussion of the option LABEL on page 72.

Specifying an Output Object

After you have determined which output objects your SAS program produces, you can specify the output objects in statements such as ODS EXCLUDE, ODS SELECT, and so on. You can specify an output object by using one of the following:

- a full path. For example, the following is the full path of the output object:

  ```
  Univariate.City_Pop_90.TestsForLocation
  ```

- a partial path. A partial path consists of any part of the full path that begins immediately after a period (.) and continues to the end of the full path. For example, suppose the full path is the following:

  ```
  Univariate.City_Pop_90.TestsForLocation
  ```

  Then the partial paths are as follows:

  ```
  City_Pop_90.TestsForLocation
  TestsForLocation
  ```

- a label that is enclosed in quotation marks. For example:

  ```
  "The UNIVARIATE Procedure"
  ```

- a label path. For example, the following is the label path for the output object:

  ```
  "The UNIVARIATE Procedure"."CityPop_90"."Tests For Location"
  ```

  Note: The trace record shows the label path only if you specify the LABEL option in the ODS TRACE statement.

- a partial label path. A partial label path consists of any part of the label that begins immediately after a period (.) and continues to the end of the label. For example, suppose the label path is the following:

  ```
  "The UNIVARIATE Procedure"."CityPop_90"."Tests For Location"
  ```

  Then the partial label paths are as follows:

  ```
  "CityPop_90"."Tests For Location"
  "Tests For Location"
  ```

- a mixture of labels and paths.
- any of the partial path specifications, followed by a pound sign (#) and a number. For example, `TestsForLocation#3` refers to the third output object that is named `TestsForLocation`.

Example: Determining Which Output Objects a Procedure Creates

Features: ODS TRACE statement:
LABEL
OFF
ON
PROC UNIVARIATE

Details

This example shows how to determine the names and labels of the output objects that a procedure creates. You can use this information to select and exclude output objects.

Program

```
ods trace on / label;

proc univariate data=statepop mu0=3.5;
   var citypop_90 citypop_80;
run;

ods trace off;
```

Program Description

Specify that SAS write the trace record to the SAS log and include label paths. This ODS TRACE statement writes the trace record to the SAS log. The LABEL option includes label paths in the trace record.

```
ods trace on / label;
```

Create descriptive statistics for two variables. PROC UNIVARIATE computes descriptive statistics for two variables, CityPop_80 and CityPop_90. As PROC UNIVARIATE sends each output object to the Output Delivery System, ODS writes the pertinent information for that output object to the trace record.

```
proc univariate data=statepop mu0=3.5;
   var citypop_90 citypop_80;
run;
```

Specify that SAS stop writing the trace record. The ODS TRACE OFF statement stops the writing of the trace record to the SAS log.

```
ods trace off;
```

SAS Log

This partial SAS log shows the trace record that the ODS TRACE statement creates. For each analysis variable, PROC UNIVARIATE creates five output objects : **`Moments`**, **`BasicMeasures`**, **`TestsForLocation`**, **`Quantiles`**, and **`ExtremeObs`**. Notice that an output object has the same name and label, regardless of which variable is analyzed. Therefore, you can select all the moments tables that PROC UNIVARIATE produces by using the name or label in an ODS SELECT statement. The path and label path are unique for each output object because they include the name of the variable that is analyzed. You can, therefore, select an individual moments table by using the path or the label path in an ODS SELECT statement.

```
Output Added:
-------------
Name:       Moments
Label:      Moments
Template:   base.univariate.Moments
Path:       Univariate.CityPop_90.Moments
Label Path: "The Univariate Procedure"."CityPop_90"."Moments"
-------------
Output Added:
-------------
Name:       BasicMeasures
Label:      Basic Measures of Location and Variability
Template:   base.univariate.Measures
Path:       Univariate.CityPop_90.BasicMeasures
Label Path: "The Univariate Procedure"."CityPop_90"."Basic Measures of Location and
Variability"
-------------
Output Added:
-------------
Name:       TestsForLocation
Label:      Tests For Location
Template:   base.univariate.Location
Path:       Univariate.CityPop_90.TestsForLocation
Label Path: "The Univariate Procedure"."CityPop_90"."Tests For Location"
-------------
Output Added:
-------------
Name:       Quantiles
Label:      Quantiles
Template:   base.univariate.Quantiles
Path:       Univariate.CityPop_90.Quantiles
Label Path: "The Univariate Procedure"."CityPop_90"."Quantiles"
-------------
Output Added:
-------------
Name:       ExtremeObs
Label:      Extreme Observations
Template:   base.univariate.ExtObs
Path:       Univariate.CityPop_90.ExtremeObs
Label Path: "The Univariate Procedure"."CityPop_90"."Extreme Observations"
-------------
```

```
Output Added:
-------------
Name:       Moments
Label:      Moments
Template:   base.univariate.Moments
Path:       Univariate.CityPop_80.Moments
Label Path: "The Univariate Procedure"."CityPop_80"."Moments"
-------------
Output Added:
-------------
Name:       BasicMeasures
Label:      Basic Measures of Location and Variability
Template:   base.univariate.Measures
Path:       Univariate.CityPop_80.BasicMeasures
Label Path: "The Univariate Procedure"."CityPop_80"."Basic Measures of Location and
Variability"
-------------
Output Added:
-------------
Name:       TestsForLocation
Label:      Tests For Location
Template:   base.univariate.Location
Path:       Univariate.CityPop_80.TestsForLocation
Label Path: "The Univariate Procedure"."CityPop_80"."Tests For Location"
-------------
Output Added:
-------------
Name:       Quantiles
Label:      Quantiles
Template:   base.univariate.Quantiles
Path:       Univariate.CityPop_80.Quantiles
Label Path: "The Univariate Procedure"."CityPop_80"."Quantiles"
-------------
Output Added:
-------------
Name:       ExtremeObs
Label:      Extreme Observations
Template:   base.univariate.ExtObs
Path:       Univariate.CityPop_80.ExtremeObs
Label Path: "The Univariate Procedure"."CityPop_80"."Extreme Observations"
-------------
```

See Also

Statements

- “ODS EXCLUDE Statement” on page 12
- “ODS SELECT Statement” on page 20

ODS PROCLABEL Statement

Enables you to change a procedure label.

Category: ODS: Output Control

Interaction: This statement applies to all open destinations, except for the output destination, where a procedure label is not an option. However, this setting lasts for only one procedure

step. You must issue an ODS PROCLABEL statement for each procedure step that you have.

Examples: "Example 2: Adding Text That Imitates a System Title" on page 66

"Customizing the Table of Contents" in *SAS Output Delivery System: User's Guide*

Syntax

ODS PROCLABEL '*string*';

ODS PROCLABEL= '*string*';

Required Argument

'*string*'

is the procedure label that you specify.

Interaction The NOLABEL system option overrides the ODS PROCLABEL statement. Therefore, to produce labels using the ODS PROCLABEL statement, you must specify the LABEL system option also.

Details

ODS PROCLABEL affects the item names in the outer list of the table of contents.

See Also

System Option

- LABEL System Option

PART 3

The DOCUMENT Procedure

Chapter 5
ODS DOCUMENT Statement **81**

Chapter 6
DOCUMENT Procedure **87**

5

ODS DOCUMENT Statement

Dictionary 81
ODS DOCUMENT Statement 81

Dictionary

ODS DOCUMENT Statement

Opens, manages, or closes the DOCUMENT destination, which produces a hierarchy of output objects that enables you to produce multiple ODS output formats without rerunning a PROC or DATA step.

Category: ODS: Output Control

Interaction: The combination of the ODS DOCUMENT statement and the DOCUMENT procedure enables you to store a report's individual components and then modify and replay the report. The ODS DOCUMENT statement stores the actual ODS objects that are created when running a report. You can then use the DOCUMENT procedure to rearrange, duplicate, or remove output from the results of a procedure or a database query without invoking the procedures from the original report. For complete documentation about the DOCUMENT procedure, see Chapter 6, "DOCUMENT Procedure," on page 87.

Syntax

ODS DOCUMENT *action*;

ODS DOCUMENT

< NAME=<*libref.*> *member-name* <(*access–option*)> >

< DIR=(< PATH=*path*<(*access-option*)> < LABEL="*label*"> >)>

< CATALOG=*permanent-catalog* | _NULL_> ;

Optional Arguments

CATALOG=*permanent-catalog* | _NULL_

CAUTION
If you do not specify a value (other than _NULL_) for this option, then you can replay temporary GRSEGs only during the session in which they are created, not in subsequent sessions.

permanent-catalog
copies any temporary GRSEG to the specified permanent catalog and keeps a reference to the permanent GRSEG in the document. This value persists until the ODS DOCUMENT statement is closed, or until you delete it by specifying CATALOG=_NULL_.

The *permanent catalog* has the following form.

<libref.> < member-name> ;

NULL
deletes the catalog name that was previously specified for the CATALOG= option. Thereafter, temporary GRSEGs are not copied into the permanent catalog and thus are unavailable in subsequent sessions.

Alias CAT=

Default By default, no value is assigned to CATALOG=, which means that temporary GRSEGs are not copied to a permanent catalog.

DIR=(< PATH=*path*< (*access-option*) > >< LABEL='label' >);
specifies the directory path and/or label for ODS output.

LABEL=*label*
assigns a label to a path.

Requirement The label that you assign must be enclosed in quotation marks.

Interaction If LABEL= is used with the PATH= option, then the label applies to the path. If LABEL= is used without the PATH= option, then the label applies to the entire document.

PATH=*path* < (*access-option*) >
is specified as a sequence of entries that are delimited by backslashes.

path
can have the following form.

path<#sequence–number>

path
is the name of the path.

#sequence–number
is a number which, when combined with a pathname, uniquely identifies the entry in the directory that contains it.

Default The default path is "\" (root).

Tip You can specify a directory that contains entries that do not exist in the document.

access-option
specifies the access mode for the ODS document.

WRITE
opens a document and provides Write access as well as Read access.

Interaction If a label is specified with the LABEL= option, then overrides any existing label assigned to the document.

Tip If the ODS document does not exist, then it is created.

CAUTION **If the ODS document already exists, then it is overwritten.**

UPDATE
opens an ODS document and appends new content to the document. UPDATE provides Update access as well as Read access.

Interaction If a label is specified with the LABEL= option, then it is assigned to the document.

Tip If the ODS document does not exist, then the document is created.

CAUTION **If the document already exists, then its contents are not changed.**

Default UPDATE

Note Procedure output or data queries are added at the end of the directory.

NAME= <*libref.*> *member-name*< (access-option) >

libref
specifies the SAS library where the document is stored.

Default If no library name is specified, the Work library is used.

member-name
specifies the document name.

Defaults If no NAME= is specified, the specified options apply to the currently open document.

If you do not specify an *access-option* with NAME=, then your directories will open in UPDATE mode.

access-option
specifies the access mode for the ODS document.

WRITE
opens a document and provides Write access as well as Read access.

Interaction If a label is specified with the LABEL= option, then it overrides any existing label assigned to the document.

Tip If the ODS document does not exist, then it is created.

CAUTION **If the ODS document already exists, then it is overwritten.**

UPDATE
opens an ODS document and appends new content to the document. UPDATE provides Update access as well as Read access.

Interaction If a label has been specified with the LABEL= option, then it is assigned to the document.

Tip If the ODS document does not exist, then the document is created.

CAUTION **If the document already exists, then its contents is not changed.**

Default UPDATE

Interaction When a DOCUMENT destination is open, using the NAME= option in an ODS DOCUMENT statement forces ODS to close the destination and all files associated with it, and to open a new instance of the destination.

Actions

CLOSE
closes the destination and any files that are associated with it.

Tip When an ODS destination is closed, ODS does not send output to that destination. Closing an unneeded destination frees some system resources.

EXCLUDE *exclusion(s)* | ALL | NONE
excludes one or more output objects from the DOCUMENT destination.

Default NONE

Restriction The DOCUMENT destination must be open for this action to take effect.

See “ODS EXCLUDE Statement” on page 12

SELECT *selection(s)* | ALL | NONE
selects one or more output objects for the DOCUMENT destination.

Default ALL

Restriction The DOCUMENT destination must be open for this action to take effect.

See “ODS SELECT Statement” on page 20

SHOW
writes the current selection or exclusion list for the destination to the SAS log.

Restriction The destination must be open for this action to take effect.

Tip If the selection or exclusion list is the default list (SELECT ALL), then SHOW also writes the entire selection or exclusion list.

See "ODS SHOW Statement" in *SAS Output Delivery System: User's Guide*

Chapter 6

DOCUMENT Procedure

Overview: DOCUMENT Procedure ***87***
What Does the DOCUMENT Procedure Do? 88
DOCUMENT Procedure Terminology 88

Concepts: DOCUMENT Procedure ***89***
Understanding an ODS Document Path 89
Understanding Sequence Numbers 90
ODS Documents and Base SAS Procedures 90
Getting Familiar with Output Objects 90

Syntax: DOCUMENT Procedure ***91***
PROC DOCUMENT Statement 93
COPY TO Statement 95
DELETE Statement 101
DIR Statement 106
DOC Statement 107
DOC CLOSE Statement 109
HIDE Statement 109
IMPORT TO Statement 109
LINK Statement 111
LIST Statement 112
MAKE Statement 119
MOVE TO Statement 120
NOTE Statement 126
OBANOTE Statement 127
OBBNOTE Statement 128
OBDYNAM Statement 130
OBFOOTN Statement 130
OBPAGE Statement 131
OBSTITLE Statement 132
OBTEMPL Statement 133
OBTITLE Statement 134
RENAME TO Statement 135
REPLAY Statement 135
SETLABEL Statement 142
UNHIDE Statement 143

Usage: DOCUMENT Procedure ***144***
Working with ODS Documents 144
Replaying Graphics 146

Customizing Labels, Titles, and Footnotes with BY Variables 146
Using WHERE Expressions in PROC DOCUMENT 148
Working with Output Objects and Table Templates 152

Examples: DOCUMENT Procedure ***153***
Example 1: Navigating the Directory and Listing the Entries 153
Example 2: Managing Entries 157
Example 3: Listing BY-Group Entries 163
Example 4: Create a Table of Dynamic Values 169
Example 5: Output Dynamics to a Data Set, Modify the Dynamics, and Use Them in Subsequent Graphs 172
Example 6: Opening and Listing ODS Documents Using WHERE Expressions . . 178
Example 7: Restructuring Output 183
Example 8: Using PROC DOCUMENT to Work with Table Templates 188

Overview: DOCUMENT Procedure

What Does the DOCUMENT Procedure Do?

The combination of the ODS DOCUMENT statement and the DOCUMENT procedure enables you to store a report's individual components and then modify and replay the report. The ODS DOCUMENT statement stores the actual ODS objects (or references to those objects) that are created when running a report. You can then use the DOCUMENT procedure to rearrange, duplicate, or remove output from the results of a procedure or a database query without invoking the procedures from the original report. You can also use the DOCUMENT procedure to do the following:

- transform a report without rerunning an analysis or repeating a database query
- modify the structure of output
- display output to any ODS output format
- navigate the current directory and list entries
- open and list ODS documents
- manage output

With the DOCUMENT procedure, you are not limited to simply regenerating the same report. You can change the order in which objects are rendered, the table of contents, the templates that are used, macro variables, and ODS system options.

DOCUMENT Procedure Terminology

current document
: is the open document.

current directory
: is your current location in the open document. The '^' symbol represents the current directory.

entry
: is a directory, output object, note, or link.

graph segment
: is an output object that contains a graph.

ODS document
: is the hierarchy of output objects that are created by the DOCUMENT procedure. These objects are unformatted and are placed in a SAS item store.

path
: is the route through an ODS document, leading to a particular entry. The '^' symbol represents the current directory and the '^^' symbol represents the parent directory.

replay
: is the regeneration of output, in the same or different format, without rerunning analyses or data queries.

root directory
: is the top level of an ODS document. The root is not contained within another directory and it does not have a name assigned.

Concepts: DOCUMENT Procedure

Understanding an ODS Document Path

Definition of ODS Document Path

An ODS document is stored as an item store. This file format enables client applications to define a hierarchical file system within a file. This is similar to a directory system in a Linux operating environment. Therefore, an ODS document path indicates the location of an entry.

Entry Names

Entry names follow these rules:

- must be alphanumeric

- must begin with an alphabetical character
- can contain underscores
- can have no more than 32 characters
- are preserved with casing that is specified in the operating environment
- can have labels that are no more than 256 characters

Entries in an ODS document can be displayed in the following three ways:

- ordered by insertion, which is the default order
- ordered by ascending date-and-time stamp
- ordered alphabetically

Understanding Sequence Numbers

Entry names are not required to be unique within an ODS document. However, they are uniquely identifiable because they contain sequence numbers. Every entry in an ODS document, except for the root directory, has a sequence number. A sequence number is a positive integer that is unique with respect to the name of the entry within the same directory. Entries are assigned sequence numbers according to the sequence in which they are added to a directory. For example, the first entry **`myname`** is assigned a sequence number 1, **`myname#1`**. The second entry **`myname`** is assigned a sequence number 2, **`myname#2`**. Sequence numbers are never reassigned, unless all entries with the same name are deleted. In this case, the sequence numbers are reset to have an initial number of 1.

ODS Documents and Base SAS Procedures

You can create an ODS document from almost any SAS procedure. The PRINT, REPORT, and TABULATE procedures use table templates that are created by the user and not defined by an external template in ODS. These procedures use custom table templates, custom data components, and custom formats for their output objects. Nevertheless, the ODS document and all of its features are supported for the PRINT, TABULATE, and REPORT procedures.

Getting Familiar with Output Objects

An output object is one of the following:

- equation
- graph
- note
- table

Output objects have associated information and attributes. Some or all of the following attributes pertain to output objects:

after-note
: is the note assigned to the output object by the procedure that produced the object. This note is displayed every time the output object is displayed. After-notes are displayed after the output object.

before-note
: is the note assigned to the output object by the procedure that produced the object. This note is displayed every time the output object is displayed. Before-notes are displayed before the output object.

footnote
: is created by the FOOTNOTE statement and is displayed at the bottom of the page.

page break
: causes a page break before displaying the output object and any associated titles and notes.

subtitle
: is the title that is assigned to the output object by the procedure that produced the output object. This title is displayed every time a new page of output is started.

title
: is created by the TITLE statement and is displayed at the top of the page.

Here is the order in which the attributes of an output object are displayed:

1. page break
2. titles
3. subtitles
4. before-notes
5. output object
6. after-notes
7. footnotes

Syntax: DOCUMENT Procedure

PROC DOCUMENT <*options*>;

COPY *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …> **TO** *path* </ *options*>;

DELETE *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …> < / LEVELS= ALL | *value*>;

DIR <*path*>;

DOC <*options*>;

DOC CLOSE;

HIDE *path* <, *path-2*, …>;

IMPORT DATA= *data-set-name*<*data-set-options*> | GRSEG=*grseg* | TEXTFILE=*filename* | *fileref* **TO** *path* </ *options*>;

LINK *path* **TO** *path* </ *options* >;

LIST *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …> </ *options*>;

MAKE *path* <, *path-2*, …> </ *options*>;

MOVE *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …> **TO** *path* </ *options* >;

NOTE *path* <*'text'*> </ *options*>;

OBANOTE<*n*> *output-object* <*'text'*> </ *option*>;

OBBNOTE<*n*> *output-object* <*'text'*> </ *option*>;

OBDYNAM *output-object*;

OBFOOTN<*n*> *output-object* <*'text'*>;

OBPAGE *output-object* </ *options*>;

OBSTITLE<*n*> *output-object* <*'text'*> </ *options*>;

OBTEMPL *output-object*;

OBTITLE<*n*> *output-object* <*'text'*>;

RENAME *path-1* **TO** *path-2*;

REPLAY *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …> </ *options*>;

SETLABEL *path 'label'*;

UNHIDE *path* <, *path-2*, …>;

QUIT;

Statement	Task	Example
PROC DOCUMENT	Render ODS output without rerunning procedures and gain more control over the structure and hierarchy of the output	Ex. 6, Ex. 2, Ex. 3, Ex. 4, Ex. 5
COPY TO	Insert a copy of an entry into a specified path	
DELETE	Delete entries from a specified path or paths	
DIR	Set or display the current directory	Ex. 1, Ex. 6, Ex. 2, Ex. 4
DOC	Open a document and its contents to browse or edit	Ex. 1, Ex. 4
DOC CLOSE	Close the current document	
HIDE	Prevent output from being displayed when the document is replayed	
IMPORT TO	Import a data set or graph segment into the current directory	
LINK	Create a symbolic link from one output object to another output object	

Statement	Task	Example
LIST	List the content of one or more entries	Ex. 1, Ex. 6, Ex. 2, Ex. 3, Ex. 4, Ex. 5
MAKE	Create one or more new directories	
MOVE TO	Move entries from one directory to another directory	
NOTE	Create text strings in the current directory	Ex. 2
OBANOTE	Create or modify lines of text after the specified output object	Ex. 2
OBBNOTE	Create or modify lines of text before the specified output object	Ex. 2
OBDYNAM	Print a table of dynamic values for the specified output object	Ex. 4, Ex. 5
OBFOOTN	Create or modify lines of text at the bottom of the page in which the output object is displayed	Ex. 2
OBPAGE	Create or delete a page break for an output object	Ex. 2
OBSTITLE	Create or modify subtitles	Ex. 2
OBTEMPL	Write the source code of the ODS template that is associated with a specified output object	Ex. 3
OBTITLE	Create or modify lines of text at the top of the page where the output object is displayed	Ex. 2
RENAME TO	Assign a different name to a directory or output object	
REPLAY	Replay one or more entries to the specified open ODS destinations	Ex. 6, Ex. 2, Ex. 5
SETLABEL	Assign a label to the current entry	
UNHIDE	Enable the output of a hidden entry to be displayed when it is replayed	

PROC DOCUMENT Statement

Creates or opens a document to modify.

Default: Documents are opened in the UPDATE access mode.

Restriction: User-defined format names must be unique within an ODS DOCUMENT.

Note: If the DOCUMENT destination is not closed with an ODS DOCUMENT CLOSE statement, then ODS continues to append files to the document.

Syntax

PROC DOCUMENT <NAME= <*libref.*>*member-name* <READ | UPDATE | WRITE> > <LABEL>;

Optional Arguments

NAME= <*libref.*>*member-name* <READ | UPDATE | WRITE>
specifies the name of a new or existing document and its access mode.

<libref.>member-name
identifies a new or existing ODS document.

Default If no library is specified, then the Work library is used.

Restriction The ODS document must be a SAS library member.

READ
opens a document and provides Read-Only access.

Requirement To open a document in the READ access mode, the document must already exist.

Interaction If a label has been specified with the LABEL= option, then the label is ignored.

UPDATE
opens an ODS document and appends new content to the document. UPDATE provides Update access as well as Read access. UPDATE is the default access setting.

Interaction If a label has been specified with the LABEL= option, then it is assigned to the document.

Tip If the ODS document does not exist, then the document is created.

CAUTION **If the document already exists, then its contents is not changed.**

WRITE
opens a document and provides Read and Write access.

Interaction If a label has been specified with the LABEL= option, then it overrides any existing label assigned to the document.

Tip If the ODS document does not exist, then it is created.

Example The following PROC DOCUMENT statement opens the document Work.YourDoc in Write mode:

```
proc document name=yourdoc(write);
run;
```

CAUTION **If the ODS document already exists, then it is overwritten.**

LABEL= '*label*'
assigns a label to a document.

Restriction Labels can be assigned only to documents with Write-access permissions.

Requirement Enclose labels in quotation marks.

Example The following PROC DOCUMENT statement opens the document Work.YourDoc in Write mode and assigns a label to it:

```
proc document name=yourdoc(write) label='repeated measures results';
run;
```

COPY TO Statement

Copies an entry into the specified path.

Default: If you do not specify a location to insert the entry into the path, then the entry is inserted at the end of the path.

Syntax

COPY *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …> **TO** *path* </ *options*>;

Required Argument

path
is the location where a link, output object, or file is copied.

Requirement Separate multiple paths with commas.

Tip The '^' symbol represents the current directory and the '^^' symbol represents the parent directory.

Optional Argument

(WHERE=(*where-expression-1<operator >*))
conditionally selects a subset of entries in an ODS document.

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

operand
specifies the constant, SAS function, or variable to be operated on.

An operand is one of the following:

constant
is a fixed value such as a date literal, a value, or a BY variable value.

SAS function
returns a value from a computation or system manipulation.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

subsetting variable
is a special type of WHERE expression operand used by the DOCUMENT procedure to help you find common values in ODS documents.

Here are the subsetting variables:

CDATE
is the creation date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
   \ work.mydoc\monthly;
run;
```

CDATETIME
is the creation datetime of the current entry.

Example The following COPY TO statement copies all entries with a creation datetime of May 1, 2003, at 9:30 to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_cdatetime_ = '01may04:9:30:00'dt))
to \work.mydoc\monthly;
run;
```

CTIME
is the creation time of the current entry.

Example The following DELETE statement deletes all entries with a creation time of 9:25:19 PM:

```
delete ^(where=(_ctime_ = '9:25:19pm't));
run;
```

LABEL
is the label of the current entry.

Example The following LIST statement lists all tables containing the label 'Type III Model' within the GLM procedure:

```
list glm(where=(_type_ = 'table' _label_ ? 'Type III Model'));
run;
```

LABELPATH
is the path to the label of the current entry. Document label paths are formed by concatenating the labels and sequence numbers, and then separating them with the forward slash (/) symbol.

Document label paths are similar to the label paths specified by the ODS TRACE statement.

Note In document label paths, the instances of '.' are replaced with '\'.

See "ODS TRACE Statement" on page 70

Example The following LIST statement lists all items containing "Fit Statistics" in the label path.

```
list glm (where=(_labelpath_ ? "Fit Statistics"))/ levels=all;
run;
```

MAX
is the last observation.

Restrictions _MAX_ is used only for output objects.

MAX is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

MDATE
is the modification date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a modification date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _mdate_ = '16JUL2004'd)) to
   \work.mydoc\monthly;
run;
```

MDATETIME
is the modification datetime of the current entry.

Example The following REPLAY statement replays all entries with a modification datetime of May 1, 2003, at 9:30:

```
replay ^(where=(_mdatetime_ = '01may04:9:30:00'dt));
run;
```

MIN
is the first observation.

Restrictions _MIN_ is used only for output objects.

MIN is always set to 1

MIN is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the first observation:

```
replay class(where=(_obs_ <  _min_));
```

MTIME
is the modification time of the current entry.

Example The following COPY TO statement copies all entries with a modification time of 9:25:19 PM to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_mtime_ = '9:25:19pm't)) to \work.mydoc\monthly;
```

```
run;
```

NAME
is the name of the current entry.

Example The following DELETE statement deletes all entries that contain the name “stemleng” within the GLM procedure:

```
delete glm(where=(_name_ ? 'stemleng'));
```

OBS
is the current observation number in an output object.

Restrictions _OBS_ is used only for output objects.

OBS is used only in the REPLAY statement.

Examples The following REPLAY statement replays all but the first ten observations:

```
replay class(where=(_obs_ > 10));
```

The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

The following REPLAY statement replays the first, third, fifth, seventh, and ninth observations:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-number
is the observation number to be replayed.

Restrictions *observation-number* is used only for output objects.

observation-number is used only in the REPLAY statement.

Example The following REPLAY statement replays the first, third, fifth, seventh, and ninth observation:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-variable
is the name of an observation.

Restrictions *observation-variable* is used only for output objects.

observation-variable is used only in the REPLAY statement.

Examples The following REPLAY statement replays all observations where the variable Weight is greater than 100:

```
replay class(where=(weight>100));
```

The following REPLAY statement replays all observations where the variable Sex is equal to ‘F’.

```
replay class(where=(sex='F'));
```

PATH
is the path of the current entry.

Example The following LIST statement lists all entries with a path containing the substring 'Anova' at all levels of the current directory:

```
list ^(where=(_path_ ? 'Anova'));
run;
```

SEQNO
is the sequence number of the current entry.

See "Understanding Sequence Numbers " on page 90

Example The following REPLAY statement replays all entries that have a sequence number of 2 in the GLM procedure:

```
replay glm(where=(_seqno_ = 2));
```

TYPE
is the type of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of July 16, 2004, to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
     \work.mydoc\monthly;
run;
```

variable-name
is the name of a BY variable.

Example The following MOVE TO statement moves all entries where the value of the variable Gender is 'F' to the Monthly directory of Work.Mydoc:

```
move ^(where=(gender='F')) to \work.mydoc\monthly;
run;
```

operator
compares one variable with a value or another variable.

operator can be AND, OR NOT, OR, AND NOT, or a comparison operator.

Table 6.1 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to

Symbol	Mnemonic Equivalent	Definition
	IN	Equal to one from a list of values

Restriction For the REPLAY statement, the WHERE= option applies to directories and output objects. For the following statements, the WHERE= option applies to directories only:

- COPY TO
- DELETE
- LIST
- MOVE TO

Requirement Enclose *where-expression* in quotation marks.

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 6: Opening and Listing ODS Documents Using WHERE Expressions" on page 178

Options to Use after a Slash Delimiter

AFTER= *path*
inserts a copy of an entry after the specified path.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

BEFORE= *path*
inserts a copy of an entry before the specified path.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

FIRST
inserts a copy of an entry at the beginning of the specified directory.

Example The following COPY TO statement inserts a copy of the entry Monday_Report at the beginning of the root directory:

```
copy weekly\monday_report to \ / first;
run;
```

LAST
inserts a copy of an entry at the end of the specified directory.

LEVELS= ALL | *value*
specifies the number of levels that you want to copy.

ALL
specifies all levels.

value
specifies the numeric value of the path level.

Example The following COPY TO statement copies two levels of the entry Weekly to the entry Monthly:

```
copy weekly to \work.mydoc\monthly / levels = 2;
run;
```

Default ALL

Restriction The LEVELS= option is valid only when you specify a directory.

DELETE Statement

Deletes entries from the current directory.

Restriction: The root directory cannot be deleted or moved.

Note: The DELETE statement affects all levels of a directory below the specified path.

Syntax

DELETE *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …>< / LEVELS= ALL | value>;

Required Argument

path
specifies the location of one or more links, output objects, or directories.

Requirement Separate multiple paths with commas.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

Example The following DELETE statement removes the ClassLevels and Nobs entries from the current directory:

```
delete classlevels, nobs;
run;
```

Optional Argument

(WHERE=(*where-expression-1<operator >*))
conditionally selects a subset of entries in an ODS document.

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

operand
specifies the constant, SAS function, or variable to be operated on.

An operand is one of the following:

constant
is a fixed value such as a date literal, a value, or a BY variable value.

SAS function
returns a value from a computation or system manipulation.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

subsetting variable
is a special type of WHERE expression operand used by the DOCUMENT procedure to help you find common values in ODS documents.

Here are the subsetting variables:

CDATE
is the creation date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
   \ work.mydoc\monthly;
run;
```

CDATETIME
is the creation datetime of the current entry.

Example The following COPY TO statement copies all entries with a creation datetime of May 1, 2003, at 9:30 to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_cdatetime_ = '01may04:9:30:00'dt))
to \work.mydoc\monthly;
run;
```

CTIME
is the creation time of the current entry.

Example The following DELETE statement deletes all entries with a creation time of 9:25:19 PM:

```
delete ^(where=(_ctime_ = '9:25:19pm't));
run;
```

LABEL
is the label of the current entry.

Example The following LIST statement lists all tables containing the label 'Type III Model' within the GLM procedure:

```
list glm(where=(_type_ = 'table' _label_ ? 'Type III Model'));
run;
```

LABELPATH
is the path to the label of the current entry. Document label paths are formed by concatenating the labels and sequence numbers, and then separating them with the forward slash (/) symbol.

Document label paths are similar to the label paths specified by the ODS TRACE statement.

Note In document label paths, the instances of '.' are replaced with '\'.

See "ODS TRACE Statement" on page 70

Example The following LIST statement lists all items containing "Fit Statistics" in the label path.

```
list glm (where=(_labelpath_ ? "Fit Statistics"))/ levels=all;
run;
```

MAX
is the last observation.

Restrictions _MAX_ is used only for output objects.

MAX is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

MDATE
is the modification date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a modification date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _mdate_ = '16JUL2004'd)) to
   \work.mydoc\monthly;
run;
```

MDATETIME
is the modification datetime of the current entry.

Example The following REPLAY statement replays all entries with a modification datetime of May 1, 2003, at 9:30:

```
replay ^(where=(_mdatetime_ = '01may04:9:30:00'dt));
run;
```

MIN
is the first observation.

Restrictions _MIN_ is used only for output objects.

MIN is always set to 1

MIN is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the first observation:

```
replay class(where=(_obs_ <  _min_));
```

MTIME
is the modification time of the current entry.

Example The following COPY TO statement copies all entries with a modification time of 9:25:19 PM to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_mtime_ = '9:25:19pm't)) to \work.mydoc\monthly;
```

```
run;
```

NAME
is the name of the current entry.

Example The following DELETE statement deletes all entries that contain the name “stemleng” within the GLM procedure:

```
delete glm(where=(_name_ ? 'stemleng'));
```

OBS
is the current observation number in an output object.

Restrictions _OBS_ is used only for output objects.

OBS is used only in the REPLAY statement.

Examples The following REPLAY statement replays all but the first ten observations:

```
replay class(where=(_obs_ > 10));
```

The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

The following REPLAY statement replays the first, third, fifth, seventh, and ninth observations:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-number
is the observation number to be replayed.

Restrictions *observation-number* is used only for output objects.

observation-number is used only in the REPLAY statement.

Example The following REPLAY statement replays the first, third, fifth, seventh, and ninth observation:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-variable
is the name of an observation.

Restrictions *observation-variable* is used only for output objects.

observation-variable is used only in the REPLAY statement.

Examples The following REPLAY statement replays all observations where the variable Weight is greater than 100:

```
replay class(where=(weight>100));
```

The following REPLAY statement replays all observations where the variable Sex is equal to ‘F’.

```
replay class(where=(sex='F'));
```

PATH
is the path of the current entry.

Example The following LIST statement lists all entries with a path containing the substring 'Anova' at all levels of the current directory:

```
list ^(where=(_path_ ? 'Anova'));
run;
```

SEQNO
is the sequence number of the current entry.

See "Understanding Sequence Numbers " on page 90

Example The following REPLAY statement replays all entries that have a sequence number of 2 in the GLM procedure:

```
replay glm(where=(_seqno_ = 2));
```

TYPE
is the type of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of July 16, 2004, to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
     \work.mydoc\monthly;
run;
```

variable-name
is the name of a BY variable.

Example The following MOVE TO statement moves all entries where the value of the variable Gender is 'F' to the Monthly directory of Work.Mydoc:

```
move ^(where=(gender='F')) to \work.mydoc\monthly;
run;
```

operator
compares one variable with a value or another variable.

operator can be AND, OR NOT, OR, AND NOT, or a comparison operator.

Table 6.2 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to

Symbol	Mnemonic Equivalent	Definition
	IN	Equal to one from a list of values

Restriction For the REPLAY statement, the WHERE= option applies to directories and output objects. For the following statements, the WHERE= option applies to directories only:

- COPY TO
- DELETE
- LIST
- MOVE TO

Requirement Enclose *where-expression* in quotation marks.

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 6: Opening and Listing ODS Documents Using WHERE Expressions" on page 178

Options to Use after a Slash Delimiter

LEVELS= ALL | *value*
specifies the number of levels that you want to delete.

ALL
specifies all levels.

value
specifies the numeric value of the path level.

Default ALL

Restriction The LEVELS= option is valid only when you specify a directory.

DIR Statement

Sets or displays the current directory.

Examples:
"Example 1: Navigating the Directory and Listing the Entries" on page 153
"Example 6: Opening and Listing ODS Documents Using WHERE Expressions" on page 178
"Example 2: Managing Entries" on page 157

Syntax

DIR <*path*>;

Without Arguments

If no options are specified, then the DIR statement displays the current path.

Optional Argument

path
sets the current directory.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

Example The following DIR statement sets the current directory to '\report\glm' within the current document:

```
dir \report\glm;
run;
```

DOC Statement

Opens a document and its contents to browse or edit.

Default: Documents are opened in the UPDATE access mode.

Examples: “Example 1: Navigating the Directory and Listing the Entries” on page 153
“Example 6: Opening and Listing ODS Documents Using WHERE Expressions” on page 178

Syntax

DOC <LABEL= '*label*'> <LIBRARY=*library-name*> <NAME= *libref.member-name* READ | UPDATE | WRITE>;

Without Arguments

If no options are specified, then the DOC statement lists the ODS documents in all SAS libraries in alphabetical order. Document labels, if any, are displayed.

Optional Arguments

LABEL= '*label*'
assigns a label to a document.

Restriction A label can be assigned only to documents with Write access permission.

Requirements To use the LABEL= option, specify the NAME= option in the DOC statement.

Enclose labels in quotation marks.

Example The following DOC statement opens the document Work.YourDoc in Write mode and assigns a label to it:

```
doc name=yourdoc(write) label='repeated measures results';
run;
```

LIBRARY=*library-name*
specifies that only the documents in the specified *library-name* are listed.

Alias LIB=

Interaction The LIBRARY= option cannot be specified with the NAME= or LABEL= options.

NAME= *libref.member-name* READ | UPDATE | WRITE
specifies the name that you assign to a document and its access mode.

<libref.>member-name
identifies a document.

Default If no library is specified, then the Work library is used.

Restriction The document must be a SAS library member.

READ
opens a document and provides Read-Only access.

Interaction If a label has been specified with the LABEL= option, then the label is ignored.

WRITE
opens a document and provides Write access, but only if you have Write permission.

CAUTION
If the document already exists, then it is overwritten. If the document does not exist, then it is created.

Interaction If a label has been specified with the LABEL= option, then it overrides any existing label assigned to the document.

UPDATE
opens a document and provides Update access, but only if you have Update permission.

Interaction If a label has been specified with the LABEL= option, then it is assigned to the document.

Tip If the document already exists, then its contents is not changed and the new contents is appended to the document. If the document does not exist, then it is created.

DOC CLOSE Statement

Closes the current document.

Syntax

DOC CLOSE;

HIDE Statement

Prevents output from being displayed when the document is replayed.

Tip: To see entries that might be hidden in the current document, use the LIST statement.

Syntax

HIDE *path* <, *path-2*, …>;

Required Argument

path
specifies the location of the file or files that you want to hide.

Requirement Separate multiple paths with commas.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

IMPORT TO Statement

Imports the specified SAS data set or graph segment to the specified path.

Syntax

IMPORT DATA= *data-set-name*<*data-set-options*> | GRSEG=*grseg* | TEXTFILE=*filename* | *fileref* **TO** *path* </ *options*>;

Required Arguments

DATA= *data-set-name*
specifies an existing SAS data set that you want to import.

GRSEG= *grseg*
stores a reference to a graph segment.

grseg
specifies the 3-level catalog pathname (for example, GRSEG=Sasuser.grseg.mygraph).

See GRSEG= option in *SAS/GRAPH: Reference*

path
specifies the location where you want to import the data set or graph segment.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

TEXTFILE= <*filename* | *fileref*>
imports a text file into an ODS document that can be replayed to open ODS destinations.

filename
specifies the filename. *filename* can be a listing file, a SAS program, or any other text file.

Requirement *filename* must be enclosed in quotation marks.

fileref
is a file reference that has been assigned to an external file. Use the FILENAME statement to assign a fileref.

See For information about the FILENAME statement, see *SAS DATA Step Statements: Reference*

Optional Argument

data-set-options
specify actions that apply only to the SAS data set.

See *SAS Data Set Options: Reference* for information about SAS data sets and their options

Options to Use after a Slash Delimiter

AFTER= *path*
imports the data set or graph segment into the directory after the specified path.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

BEFORE= *path*
imports the data set or graph segment into the directory before the specified path.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

Example The following IMPORT TO statement imports the data set Sashelp.Class to the current directory, and inserts the data set before the entry MyInfo:

```
import data=sashelp.class to ^ / before=MyInfo;
run;
```

FIRST
imports the data set or graph segment at the beginning of the directory.

LAST
imports the data set or graph segment at the end the directory.

LINK Statement

Creates a symbolic link from one specified entry to another specified entry.

Syntax

LINK *path* TO *path* </ *options*>;

Required Argument

path
specifies the locations of the entries that you want to link to one another.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

Options to Use after a Slash Delimiter

AFTER= *path*
links to the entry that follows the specified path in the current directory.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

BEFORE= *path*
links to the entry that precedes the specified path in the current directory.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

FIRST
links to the first entry in the current directory.

HARD
specifies a type of link that refers to a copy of an output object within the ODS document. All data is shared between the link and the target, except names and labels.

Restriction A hard link can reference only an output object, and the source and target paths must be in the same ODS document. The target must exist when you create the hard link.

Interaction A hard link and its target exist independently. Deleting a hard link does not affect the target. Similarly, deleting a target does not affect the link.

Example The following LINK statement creates a hard link from the output object ErrorSSCP to the output object LinkedErrorSSCP in the current directory:

```
link errorSSCP to linkederrorSSCP / hard;
run;
```

LABEL
copies the source label to the link.

Default The source label is not copied unless the LABEL option is specified.

LAST
links to the last entry in the current directory.

LIST Statement

Lists the contents of one or more entries.

Defaults: Only summary information is displayed if the DETAILS option is omitted.

If the ORDER= option is omitted, then the contents of the specified entries are listed in the order specified by the INSERT option.

Tip: To see any entries that might be hidden in the current directory, use the LIST statement.

Examples: “Example 1: Navigating the Directory and Listing the Entries” on page 153

“Example 6: Opening and Listing ODS Documents Using WHERE Expressions” on page 178

“Example 2: Managing Entries” on page 157

“Example 3: Listing BY-Group Entries” on page 163

Syntax

LIST *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …> </ *options*>;

Required Argument

path
specifies the location of an entry. An entry can be one or more directories, links, or output objects.

Requirement Separate multiple paths with commas.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

Example The following LIST statement lists all of the entries within the Report entry:

```
list \sasuser.imports\report;
run;
```

Optional Argument

(WHERE=(*where-expression-1<operator >*))
conditionally selects a subset of entries in an ODS document.

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

operand
specifies the constant, SAS function, or variable to be operated on.

An operand is one of the following:

constant
is a fixed value such as a date literal, a value, or a BY variable value.

SAS function
returns a value from a computation or system manipulation.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

subsetting variable
is a special type of WHERE expression operand used by the DOCUMENT procedure to help you find common values in ODS documents.

Here are the subsetting variables:

CDATE
is the creation date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
   \ work.mydoc\monthly;
```

```
run;
```

CDATETIME
is the creation datetime of the current entry.

Example The following COPY TO statement copies all entries with a creation datetime of May 1, 2003, at 9:30 to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_cdatetime_ = '01may04:9:30:00'dt))
to \work.mydoc\monthly;
run;
```

CTIME
is the creation time of the current entry.

Example The following DELETE statement deletes all entries with a creation time of 9:25:19 PM:

```
delete ^(where=(_ctime_ = '9:25:19pm't));
run;
```

LABEL
is the label of the current entry.

Example The following LIST statement lists all tables containing the label 'Type III Model' within the GLM procedure:

```
list glm(where=(_type_ = 'table' _label_ ? 'Type III Model'));
run;
```

LABELPATH
is the path to the label of the current entry. Document label paths are formed by concatenating the labels and sequence numbers, and then separating them with the forward slash (*/*) symbol.

Document label paths are similar to the label paths specified by the ODS TRACE statement.

Note In document label paths, the instances of '.' are replaced with '\'.

See “ODS TRACE Statement” on page 70

Example The following LIST statement lists all items containing “Fit Statistics” in the label path.

```
list glm (where=(_labelpath_ ? "Fit Statistics"))/ levels=all;
run;
```

MAX
is the last observation.

Restrictions _MAX_ is used only for output objects.

MAX is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

MDATE
is the modification date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a modification date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _mdate_ = '16JUL2004'd)) to
   \work.mydoc\monthly;
run;
```

MDATETIME
is the modification datetime of the current entry.

Example The following REPLAY statement replays all entries with a modification datetime of May 1, 2003, at 9:30:

```
replay ^(where=(_mdatetime_ = '01may04:9:30:00'dt));
run;
```

MIN
is the first observation.

Restrictions _MIN_ is used only for output objects.

MIN is always set to 1

MIN is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the first observation:

```
replay class(where=(_obs_ <  _min_));
```

MTIME
is the modification time of the current entry.

Example The following COPY TO statement copies all entries with a modification time of 9:25:19 PM to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_mtime_ = '9:25:19pm't)) to \work.mydoc\monthly;
run;
```

NAME
is the name of the current entry.

Example The following DELETE statement deletes all entries that contain the name “stemleng” within the GLM procedure:

```
delete glm(where=(_name_ ? 'stemleng'));
```

OBS
is the current observation number in an output object.

Restrictions _OBS_ is used only for output objects.

OBS is used only in the REPLAY statement.

Examples The following REPLAY statement replays all but the first ten observations:

```
replay class(where=(_obs_ > 10));
```

The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

The following REPLAY statement replays the first, third, fifth, seventh, and ninth observations:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-number
is the observation number to be replayed.

Restrictions *observation-number* is used only for output objects.

observation-number is used only in the REPLAY statement.

Example The following REPLAY statement replays the first, third, fifth, seventh, and ninth observation:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-variable
is the name of an observation.

Restrictions *observation-variable* is used only for output objects.

observation-variable is used only in the REPLAY statement.

Examples The following REPLAY statement replays all observations where the variable Weight is greater than 100:

```
replay class(where=(weight>100));
```

The following REPLAY statement replays all observations where the variable Sex is equal to ‘F’.

```
replay class(where=(sex='F'));
```

PATH
is the path of the current entry.

Example The following LIST statement lists all entries with a path containing the substring 'Anova' at all levels of the current directory:

```
list ^(where=(_path_ ? 'Anova'));
run;
```

SEQNO
is the sequence number of the current entry.

See “Understanding Sequence Numbers ” on page 90

Example The following REPLAY statement replays all entries that have a sequence number of 2 in the GLM procedure:

```
replay glm(where=(_seqno_ = 2));
```

TYPE
is the type of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of July 16, 2004, to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
```

```
        \work.mydoc\monthly;
run;
```

variable-name
is the name of a BY variable.

Example The following MOVE TO statement moves all entries where the value of the variable Gender is 'F' to the Monthly directory of Work.Mydoc:

```
move ^(where=(gender='F')) to \work.mydoc\monthly;
run;
```

operator
compares one variable with a value or another variable.

operator can be AND, OR NOT, OR, AND NOT, or a comparison operator.

Table 6.3 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Restriction For the REPLAY statement, the WHERE= option applies to directories and output objects. For the following statements, the WHERE= option applies to directories only:

- COPY TO
- DELETE
- LIST
- MOVE TO

Requirement Enclose *where-expression* in quotation marks.

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 6: Opening and Listing ODS Documents Using WHERE Expressions" on page 178

Options to Use after a Slash Delimiter

BYGROUPS
creates, in the entry list, columns for BY variables. The name of the BY variable becomes the column name. The values of the BY variables are listed in the columns.

Note: When you specify the BYGROUPS option, only entries containing BY group information are listed.

Interaction It is recommended that when you specify the BYGROUPS option, specify the LEVELS=ALL option also. If the LEVELS=ALL option is not specified, then ODS cannot find BY group information within all levels of the directories.

Example "Example 3: Listing BY-Group Entries" on page 163

DETAILS
specifies the properties of the entries.

Specifying `list/levels=all details;` generates a table with the following columns:

Created	shows the date on which the entry was created.
Label	shows the entry's label.
Modified	shows the date on which the entry was last modified.
Page Break	shows before or after page breaks, if present.
Path	shows the entry's path.
Size in Bytes	shows the entry's size in bites.
Symbolic Link	shows a symbolic link associated with the entry, if present.
Template	shows the template associated with the entry, if present.
Type	shows the entry's type.

Example The following LIST statement lists the details of three levels of the Report entry:

```
list \sasuser.imports\report / details levels=3;
run;
```

FOLLOW
resolves all links and lists the contents of the entries.

LEVELS= ALL | *value*
specifies the number of levels that you want to list.

ALL
specifies all levels.

value
specifies the numeric value of the path level.

Default If you omit the LEVELS= option, then the default value of the level is 1.

Restriction The LEVELS= option is valid only when you specify a directory.

Example The following LIST statement lists the details of three levels of the Report entry:

```
list \sasuser.imports\report / details levels=3;
run;
```

ORDER= ALPHA | DATE | INSERT
specifies the order in which the entries are listed.

ALPHA
lists the entries in alphabetical order.

DATE
lists the directories in ascending order based on the date and time the files were created.

INSERT
lists the directories in the order in which the entries were inserted.

MAKE Statement

Creates one or more new directories.

Default: If no location is specified, the newly created directory is appended to the end of the current directory.

Syntax

MAKE *path* <, *path-2*, …> </ *options*>;

Required Argument

path
specifies the newly created directory.

Requirement Separate multiple paths with commas.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

Options to Use after a Slash Delimiter

AFTER= *path*
adds the newly created directory after the specified path in the current directory.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

BEFORE= *path*
adds the newly created directory before the specified path in the current directory.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

FIRST
adds the newly created directory to the beginning of the current directory.

LAST
adds the newly created directory to the end of the current directory.

MOVE TO Statement

Moves entries from the specified location to another location.

Restriction: The root directory cannot be moved or deleted.

Requirement: Separate multiple paths with commas.

Tip: The MOVE TO statement affects all levels of a directory below the specified starting level.

Syntax

MOVE *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …> **TO** *path* </ *options*>;

Required Argument

path
specifies the location of links, output objects, or files that you want to move.

CAUTION
The MOVE TO statement affects all levels of a directory below the specified starting level.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

Optional Argument

(WHERE=(*where-expression-1<operator >*))
conditionally selects a subset of entries in an ODS document.

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

operand
specifies the constant, SAS function, or variable to be operated on.

An operand is one of the following:

constant
is a fixed value such as a date literal, a value, or a BY variable value.

SAS function
returns a value from a computation or system manipulation.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

subsetting variable
is a special type of WHERE expression operand used by the DOCUMENT procedure to help you find common values in ODS documents.

Here are the subsetting variables:

CDATE
is the creation date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
   \ work.mydoc\monthly;
run;
```

CDATETIME
is the creation datetime of the current entry.

Example The following COPY TO statement copies all entries with a creation datetime of May 1, 2003, at 9:30 to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_cdatetime_ = '01may04:9:30:00'dt))
to \work.mydoc\monthly;
run;
```

CTIME
is the creation time of the current entry.

Example The following DELETE statement deletes all entries with a creation time of 9:25:19 PM:

```
delete ^(where=(_ctime_ = '9:25:19pm't));
run;
```

LABEL
is the label of the current entry.

Example The following LIST statement lists all tables containing the label 'Type III Model' within the GLM procedure:

```
list glm(where=(_type_ = 'table' _label_ ? 'Type III Model'));
run;
```

LABELPATH
is the path to the label of the current entry. Document label paths are formed by concatenating the labels and sequence numbers, and then separating them with the forward slash (/) symbol.

Document label paths are similar to the label paths specified by the ODS TRACE statement.

Note In document label paths, the instances of '.' are replaced with '\'.

See “ODS TRACE Statement” on page 70

Example The following LIST statement lists all items containing “Fit Statistics” in the label path.

```
list glm (where=(_labelpath_ ? "Fit Statistics"))/ levels=all;
run;
```

MAX
is the last observation.

Restrictions _MAX_ is used only for output objects.

MAX is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

MDATE
is the modification date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a modification date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _mdate_ = '16JUL2004'd)) to
   \work.mydoc\monthly;
run;
```

MDATETIME
is the modification datetime of the current entry.

Example The following REPLAY statement replays all entries with a modification datetime of May 1, 2003, at 9:30:

```
replay ^(where=(_mdatetime_ = '01may04:9:30:00'dt));
run;
```

MIN
is the first observation.

Restrictions _MIN_ is used only for output objects.

MIN is always set to 1

MIN is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the first observation:

```
replay class(where=(_obs_ <  _min_));
```

MTIME
is the modification time of the current entry.

Example The following COPY TO statement copies all entries with a modification time of 9:25:19 PM to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_mtime_ = '9:25:19pm't)) to \work.mydoc\monthly;
run;
```

NAME
is the name of the current entry.

Example The following DELETE statement deletes all entries that contain the name “stemleng” within the GLM procedure:

```
delete glm(where=(_name_ ? 'stemleng'));
```

OBS
is the current observation number in an output object.

Restrictions _OBS_ is used only for output objects.

OBS is used only in the REPLAY statement.

Examples The following REPLAY statement replays all but the first ten observations:

```
replay class(where=(_obs_ > 10));
```

The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

The following REPLAY statement replays the first, third, fifth, seventh, and ninth observations:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-number
is the observation number to be replayed.

Restrictions *observation-number* is used only for output objects.

observation-number is used only in the REPLAY statement.

Example The following REPLAY statement replays the first, third, fifth, seventh, and ninth observation:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-variable
is the name of an observation.

Restrictions *observation-variable* is used only for output objects.

observation-variable is used only in the REPLAY statement.

Examples The following REPLAY statement replays all observations where the variable Weight is greater than 100:

```
replay class(where=(weight>100));
```

The following REPLAY statement replays all observations where the variable Sex is equal to ‘F’.

```
replay class(where=(sex='F'));
```

PATH
is the path of the current entry.

Example The following LIST statement lists all entries with a path containing the substring 'Anova' at all levels of the current directory:

```
list ^(where=(_path_ ? 'Anova'));
run;
```

SEQNO
is the sequence number of the current entry.

See "Understanding Sequence Numbers " on page 90

Example The following REPLAY statement replays all entries that have a sequence number of 2 in the GLM procedure:

```
replay glm(where=(_seqno_ = 2));
```

TYPE
is the type of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of July 16, 2004, to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
     \work.mydoc\monthly;
run;
```

variable-name
is the name of a BY variable.

Example The following MOVE TO statement moves all entries where the value of the variable Gender is 'F' to the Monthly directory of Work.Mydoc:

```
move ^(where=(gender='F')) to \work.mydoc\monthly;
run;
```

operator
compares one variable with a value or another variable.

operator can be AND, OR NOT, OR, AND NOT, or a comparison operator.

Table 6.4 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than

Symbol	Mnemonic Equivalent	Definition
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Restriction For the REPLAY statement, the WHERE= option applies to directories and output objects. For the following statements, the WHERE= option applies to directories only:

- COPY TO
- DELETE
- LIST
- MOVE TO

Requirement Enclose *where-expression* in quotation marks.

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 6: Opening and Listing ODS Documents Using WHERE Expressions" on page 178

Options to Use after a Slash Delimiter

AFTER= *path*
moves the entry after the specified entry in the path.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

BEFORE= *path*
moves the entry before the specified entry in the path.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

FIRST
moves the entry to the beginning of the specified directory.

LAST
moves the entry to the end of the specified directory.

LEVELS= ALL | *value*
specifies the number of levels that you want to move.

ALL
specifies all levels.

value
specifies the numeric value of the path level.

Example The following MOVE TO statement moves two levels of the directory Weekly to the Monthly directory of Work.Mydoc:

```
move weekly to \work.mydoc\monthly / levels = 2;
run;
```

Default ALL

Restriction The LEVELS= option is valid only when you specify a directory.

NOTE Statement

Creates text strings in the current directory.

Default: If you omit the JUST= option, then the note is centered between the left and right margins.

Example: "Example 2: Managing Entries" on page 157

Syntax

NOTE *path* <'*text*'> </ *options*>;

Required Argument

path
specifies the location where the note is stored.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

Optional Argument

'*text*'
specifies the text string. If no text string is specified, then the NOTE statement creates a blank note.

Requirement All text strings must be enclosed in quotation marks.

Options to Use after a Slash Delimiter

AFTER= *path*
inserts the text string after the specified path.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

BEFORE= *path*
inserts the text string before the specified path.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

FIRST
inserts the text string at the beginning of the directory.

LAST
inserts the text string at the end of the directory.

JUST= LEFT | CENTER | RIGHT
specifies the alignment of the text string.

LEFT
aligns the text string with the left margin.

CENTER
centers the text string between the left and right margins.

RIGHT
aligns the text string with the right margin.

OBANOTE Statement

Creates or modifies an object footer (lines of text) after the specified output object.

Example: "Example 2: Managing Entries" on page 157

Syntax

OBANOTE <*n*> *output-object* <'*text* '> <SHOW></ JUST= LEFT | CENTER | RIGHT>;

Required Argument

output-object
specifies the name of the ODS output object.

Optional Arguments

n
specifies the relative line that contains the object footer.

Default If you omit *n*, then SAS assumes a value of 1. Therefore, specify either OBANOTE or OBANOTE1 for the first text line.

Range 1–10

Tips The OBANOTE line with the highest number appears on the bottom line.

You can create notes that contain blank lines between them. For example, if you specify a text string with an OBANOTE1 statement that is followed by an OBANOTE3 statement, then a blank line separates the two lines of text.

SHOW
specifies that a table containing the output object's heading is written to active destinations.

'*text*'
specifies the text string that becomes the object footer.

You can customize object footers by inserting BY variable values (#BYVALn), BY variable names (#BYVARn), or BY lines (#BYLINE) into object footers that are specified in PROC DOCUMENT steps. After you specify the object footer, embed the items at the position where you want them to appear. For more information, see "Customizing Labels, Titles, and Footnotes with BY Variables " on page 146.

Length The maximum *text* length is 32000 characters.

Requirement All text strings must be enclosed in quotation marks.

CAUTION **If no text string is specified, then the OBANOTE statement deletes all object footers for the specified output object only.**

Options to Use after a Slash Delimiter

JUST= LEFT | CENTER | RIGHT
specifies the alignment of the object footer.

LEFT
aligns the object footer with the left margin.

CENTER
centers the object footer between the left and right margins.

RIGHT
aligns the object footer with the right margin.

OBBNOTE Statement

Creates or modifies an object heading (lines of text) before the output object.

Example: "Example 2: Managing Entries" on page 157

Syntax

OBBNOTE <*n*> *output-object* <'*text*'> <SHOW></ JUST= LEFT | CENTER | RIGHT>;

Required Argument

output-object
specifies the name of the ODS output object.

Optional Arguments

n
specifies the relative line that contains the object heading.

Default If you omit *n*, then SAS assumes a value of 1. Therefore, specify either OBBNOTE or OBBNOTE1 for the first text line.

Range 1– 10

Tips The OBBNOTE line with the highest number appears on the bottom line.

You can create notes that contain blank lines between them. For example, if you specify a text string with an OBBNOTE statement that is followed by an OBBNOTE3 statement, then a blank line separates the two lines of text.

SHOW
specifies that a table containing the output object's footers is written to active destinations.

'*text*'
specifies the text string that becomes the object heading.

You can customize object headings by inserting BY variable values (#BYVALn), BY variable names (#BYVARn), or BY lines (#BYLINE) into object headings that are specified in PROC DOCUMENT steps. After you specify the object heading text, embed the items at the position where you want them to appear. For more information, see "Customizing Labels, Titles, and Footnotes with BY Variables " on page 146 .

Length The maximum *text* length is 32000 characters.

Requirement All text strings must be enclosed in quotation marks.

CAUTION **If no text string is specified, then the OBBNOTE statement deletes all existing object headings for the specified output object only.**

Options to Use after a Slash Delimiter

JUST= LEFT | CENTER | RIGHT
specifies the alignment of the object heading.

LEFT
aligns the object heading with the left margin.

CENTER
centers the object heading between the left and right margins.

RIGHT
aligns the object heading with the right margin.

OBDYNAM Statement

Prints a table of dynamic values for the specified output object.

Example: "Example 4: Create a Table of Dynamic Values" on page 169

Syntax

OBDYNAM *output-object*

Required Argument

output-object
specifies the ODS output object.

Details

The output object created by the OBDYNAM statement is a factoid data set named Dynamics, and the description is "Output and data object dynamics". You can modify the data set and use it with the DYNAMDATA option in the REPLAY statement.

OBFOOTN Statement

Creates or modifies lines of text at the bottom of the page on which the output object is displayed.

Restriction: You can print up to ten lines of text.

Tip: The OBFOOTN statement is similar to the global FOOTNOTE statement.

Example: "Example 2: Managing Entries" on page 157

Syntax

OBFOOTN <*n*> *output-object* <SHOW><'*text*'>;

Required Argument

output-object
specifies the ODS output object.

Optional Arguments

n
specifies the relative line that contains the footnote.

Range 1–10

Tips The OBFOOTN line with the highest number appears on the bottom line. If you omit *n*, then SAS assumes a value of 1. Therefore, specify OBFOOTN or OBFOOTN1 for the first text line.

You can create footnotes that contain blank lines between them. For example, if you specify a text string with an OBFOOTN statement that is followed by an OBFOOTN3 statement, then a blank line separates the two lines of text.

SHOW
specifies that a table containing the output object's footnotes is written to active destinations.

'*text*'
specifies the text string that becomes the footnote.

You can customize footnotes by inserting BY variable values (#BYVALn), BY variable names (#BYVARn), or BY lines (#BYLINE) into footnotes that are specified in PROC DOCUMENT steps. After you specify the text, embed the items at the position where you want them to appear. For more information, see "Customizing Labels, Titles, and Footnotes with BY Variables " on page 146 .

Length The maximum *text* length is 32000 characters.

Requirement All text strings must be enclosed in quotation marks.

CAUTION **If you use the OBFOOTN statement without a text string, then all existing footnotes for the specified output object are deleted.**

OBPAGE Statement

Creates or deletes a page break for an output object.

Example: "Example 2: Managing Entries" on page 157

Syntax

OBPAGE <*output-object*> < / <DELETE > <AFTER>>;

Without Arguments

If no options are specified, then the OBPAGE statement inserts a page break before an output object.

Optional Argument

output-object
specifies the name of the output object.

Options to Use after a Slash Delimiter

AFTER
inserts a page break after an output object.

Tip To delete a page break after an output object, use the AFTER option as well as the DELETE option.

DELETE
removes the page break for an output object.

TIP There is an initial page break for each destination for each use of the REPLAY statement. There might be subsequent page breaks depending on the destination and the output being generated.

OBSTITLE Statement

Create or modify subtitles.

Example: "Example 2: Managing Entries" on page 157

Syntax

OBSTITLE<*n*> *output-object* <'*text*'> <SHOW></ JUST= LEFT | CENTER | RIGHT>;

Required Argument

output-object
specifies the ODS output object.

Optional Arguments

n
specifies the relative line that contains the subtitle.

Range 1–10

Tips The OBSTITLE line with the highest number appears on the bottom line. If you omit *n*, then SAS assumes a value of 1. Therefore, you can specify OBSTITLE or OBSTITLE1 for the first text line.

You can create subtitles that contain blank lines between them. For example, if you specify a text string with an OBSTITLE statement that is followed by an OBSTITLE3 statement, then a blank line separates the two lines of text.

SHOW
specifies that a table containing the output object's subtitles is written to active destinations.

'*text*'
specifies the text string.

You can customize subtitles by inserting BY variable values (#BYVALn), BY variable names (#BYVARn), or BY lines (#BYLINE) into subtitles that are specified in PROC DOCUMENT steps. After you specify text, embed the items at the position where you want them to appear. For more information, see "Customizing Labels, Titles, and Footnotes with BY Variables " on page 146.

Length The maximum *text* length is 32000 characters.

Requirement All text strings must be enclosed in quotation marks.

Tip If no arguments are specified, then the OBSTITLE statement deletes all existing subtitles for the specified output object only.

Options to Use after a Slash Delimiter

JUST= LEFT | CENTER | RIGHT
specifies the alignment of the text string.

LEFT
aligns the text string with the left margin.

CENTER
aligns the text string in the center between the left and right margins.

RIGHT
aligns the text string with the right margin.

OBTEMPL Statement

Writes, to any open ODS destination, the source code of the ODS template, if any, that is associated with the specified output object.

Restriction: If the output object that is specified has no ODS template associated with it, then no output is created.

Syntax

OBTEMPL *output-object*;

Required Argument

output-object
specifies the pathname of the output object.

See "Getting Familiar with Output Objects" on page 90

Example "Example 3: Listing BY-Group Entries" on page 163

OBTITLE Statement

Creates or modifies title lines for the output.

Tip: The OBTITLE is similar to the global TITLE statement.

Example: "Example 2: Managing Entries" on page 157

Syntax

OBTITLE<*n*> *output-object* <SHOW><'*text*'>;

Required Argument

output-object
specifies the name of the output object.

Optional Arguments

n
specifies the relative line that contains the title.

Range 1–10

Tips The OBTITLE line with the highest number appears on the bottom line. If you omit *n*, then SAS assumes a value of 1. Therefore, specify OBTITLE or OBTITLE1 for the first text line.

You can create titles that contain blank lines between them. For example, if you specify a text string with an OBTITLE statement that is followed by an OBTITLE3 statement, then a blank line separates the two lines of text.

SHOW
specifies that a table containing the output object's titles is written to active destinations.

'text'
specifies the text string.

You can customize titles by inserting BY variable values (#BYVALn), BY variable names (#BYVARn), or BY lines (#BYLINE) into output titles that are specified in PROC DOCUMENT steps. After you specify the text, embed the items at the position where you want them to appear. For more information, see "Customizing Labels, Titles, and Footnotes with BY Variables " on page 146 .

Length The maximum *text* length is 32000 characters.

Requirement All text strings must be enclosed in quotation marks.

CAUTION **If no text is specified, then the OBTITLE statement deletes all existing titles for the specified output object only.**

RENAME TO Statement

Renames an entry.

Syntax

RENAME *path-1* **TO** *path-2*;

Required Arguments

path-1
specifies the current directory or output object.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

path-2
specifies the new name of the directory or output object.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

REPLAY Statement

Displays one or more entries to the specified open ODS destination(s).

Default: If you omit the LEVELS= option, then all levels of the file are displayed to all open destinations.

Restrictions: User-defined format names must be unique within an ODS DOCUMENT.

When replaying an ODS document, values created by the MVAR statement must be re-created in the same session that is replaying the document.

Tip: There is an initial page break for each destination for each use of the REPLAY statement. There might be subsequent page breaks depending on the destination and the output being generated.

Examples: "Example 2: Managing Entries" on page 157

"Example 5: Output Dynamics to a Data Set, Modify the Dynamics, and Use Them in Subsequent Graphs" on page 172

"Example 6: Opening and Listing ODS Documents Using WHERE Expressions" on page 178

"Example 7: Restructuring Output" on page 183

"Example 8: Using PROC DOCUMENT to Work with Table Templates " on page 188

Syntax

REPLAY *path* <(*where-expression*)> <, *path-2* <(*where-expression-2*)>, …> </ *options*>;

Optional Arguments

path
specifies the location of an entry. An entry can be one or more directories, links, or output objects.

Requirement Separate multiple paths with commas.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

(WHERE=(*where-expression-1<operator >*))
conditionally selects a subset of entries in an ODS document.

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

operand
specifies the constant, SAS function, or variable to be operated on.

An operand is one of the following:

constant
is a fixed value such as a date literal, a value, or a BY variable value.

SAS function
returns a value from a computation or system manipulation.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

subsetting variable
is a special type of WHERE expression operand used by the DOCUMENT procedure to help you find common values in ODS documents.

Here are the subsetting variables:

CDATE
is the creation date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
   \ work.mydoc\monthly;
run;
```

CDATETIME
is the creation datetime of the current entry.

Example The following COPY TO statement copies all entries with a creation datetime of May 1, 2003, at 9:30 to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_cdatetime_ = '01may04:9:30:00'dt))
to \work.mydoc\monthly;
run;
```

CTIME
is the creation time of the current entry.

Example The following DELETE statement deletes all entries with a creation time of 9:25:19 PM:

```
delete ^(where=(_ctime_ = '9:25:19pm't));
run;
```

LABEL
is the label of the current entry.

Example The following LIST statement lists all tables containing the label 'Type III Model' within the GLM procedure:

```
list glm(where=(_type_ = 'table' _label_ ? 'Type III Model'));
run;
```

LABELPATH
is the path to the label of the current entry. Document label paths are formed by concatenating the labels and sequence numbers, and then separating them with the forward slash (*/*) symbol.

Document label paths are similar to the label paths specified by the ODS TRACE statement.

Note In document label paths, the instances of '.' are replaced with '\'.

See “ODS TRACE Statement” on page 70

Example The following LIST statement lists all items containing “Fit Statistics” in the label path.

```
list glm (where=(_labelpath_ ? "Fit Statistics"))/ levels=all;
run;
```

MAX
is the last observation.

Restrictions _MAX_ is used only for output objects.

MAX is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

MDATE
is the modification date of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a modification date of 16JUL2004 to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _mdate_ = '16JUL2004'd)) to
   \work.mydoc\monthly;
run;
```

MDATETIME
is the modification datetime of the current entry.

Example The following REPLAY statement replays all entries with a modification datetime of May 1, 2003, at 9:30:

```
replay ^(where=(_mdatetime_ = '01may04:9:30:00'dt));
run;
```

MIN
is the first observation.

Restrictions _MIN_ is used only for output objects.

MIN is always set to 1

MIN is used only in the REPLAY statement.

Example The following REPLAY statement replays all observations except the first observation:

```
replay class(where=(_obs_ <  _min_));
```

MTIME
is the modification time of the current entry.

Example The following COPY TO statement copies all entries with a modification time of 9:25:19 PM to the Monthly directory of Work.Mydoc:

```
copy ^(where=(_mtime_ = '9:25:19pm't)) to \work.mydoc\monthly;
run;
```

NAME
is the name of the current entry.

Example The following DELETE statement deletes all entries that contain the name “stemleng” within the GLM procedure:

```
delete glm(where=(_name_ ? 'stemleng'));
```

OBS
is the current observation number in an output object.

Restrictions _OBS_ is used only for output objects.

OBS is used only in the REPLAY statement.

Examples The following REPLAY statement replays all but the first ten observations:

```
replay class(where=(_obs_ > 10));
```

The following REPLAY statement replays all observations except the last observation:

```
replay class(where=(_obs_ < _max_));
```

The following REPLAY statement replays the first, third, fifth, seventh, and ninth observations:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-number
is the observation number to be replayed.

Restrictions *observation-number* is used only for output objects.

observation-number is used only in the REPLAY statement.

Example The following REPLAY statement replays the first, third, fifth, seventh, and ninth observation:

```
replay class(where=(_obs_ in (1,3,5,7,9)));
```

observation-variable
is the name of an observation.

Restrictions *observation-variable* is used only for output objects.

observation-variable is used only in the REPLAY statement.

Examples The following REPLAY statement replays all observations where the variable Weight is greater than 100:

```
replay class(where=(weight>100));
```

The following REPLAY statement replays all observations where the variable Sex is equal to ‘F’.

```
replay class(where=(sex='F'));
```

PATH
is the path of the current entry.

Example The following LIST statement lists all entries with a path containing the substring 'Anova' at all levels of the current directory:

```
list ^(where=(_path_ ? 'Anova'));
run;
```

SEQNO
is the sequence number of the current entry.

See "Understanding Sequence Numbers " on page 90

Example The following REPLAY statement replays all entries that have a sequence number of 2 in the GLM procedure:

```
replay glm(where=(_seqno_ = 2));
```

TYPE
is the type of the current entry.

Example The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of July 16, 2004, to the Monthly directory of Work.Mydoc:

```
move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd)) to
     \work.mydoc\monthly;
run;
```

variable-name
is the name of a BY variable.

Example The following MOVE TO statement moves all entries where the value of the variable Gender is 'F' to the Monthly directory of Work.Mydoc:

```
move ^(where=(gender='F')) to \work.mydoc\monthly;
run;
```

operator
compares one variable with a value or another variable.

operator can be AND, OR NOT, OR, AND NOT, or a comparison operator.

Table 6.5 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Restriction For the REPLAY statement, the WHERE= option applies to directories and output objects. For the following statements, the WHERE= option applies to directories only:

- COPY TO
- DELETE

- LIST
- MOVE TO

Requirement Enclose *where-expression* in quotation marks.

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 6: Opening and Listing ODS Documents Using WHERE Expressions" on page 178

Options to Use after a Slash Delimiter

ACTIVEFOOTN
specifies that footnotes that are active in a SAS session override the footnotes that are stored in an ODS document.

Alias ACFOOTN

ACTIVETITLE
specifies that titles that are active in a SAS session override the titles that are stored in an ODS document.

Alias ACTITLE

ANCESTORS=*n* | ALL
specifies the number of parent directories to replay.

n
specifies the number of parent directories to replay. *n* must be a nonnegative integer.

ALL
specifies that all directories are replayed.

Default ALL

DEST= (*ODS-destination(s)*)
specifies one or more ODS destinations to display the output objects.

Requirement When you specify the DEST= option, enclose the ODS destinations in parentheses and separate each destination with a blank space. For example, DEST=(HTML5 RTF).

Tip When you specify only one destination, you do not need to use parentheses. For example, DEST=HTML5

See For information about ODS destinations, see "Understanding ODS Destinations" in *SAS Output Delivery System: User's Guide* .

Example The following REPLAY statement replays two levels of the entry Data to the HTML5 and RTF destinations:

```
replay \Report\GLM#1\Data / levels=2 dest=(html5 rtf);
run;
```

DYNAMDATA=*data-set-name*
specifies the name of the data set that contains the dynamic values.

See For information about how to create a table of dynamics, see "OBDYNAM Statement" on page 130.

LEVELS= ALL | *value*
specifies the number of levels that you want to replay.

ALL
specifies that all levels of the directory are displayed to all open destinations.

value
specifies the numeric value of the path level.

Example The following REPLAY statement replays two levels of the entry Data to the HTML5 and RTF destinations:

```
replay \Report\GLM#1\Data / levels=2 dest=(html5 rtf);
run;
```

Default ALL

Restriction The LEVELS= option is valid only when you specify a directory.

STORE=*template-store*
specifies the template store when replaying output objects from the path. If the replayed path is an output object, the template store applies only to that output object. If the path is a directory, the template store applies to any output objects contained within the directory.

Restriction The STORE= option applies only to output objects with external templates. Output objects with internal templates, such as those generated by PROC PRINT, PROC REPORT, PROC TABULATE, and PROC SGPLOT, are unaffected by STORE=.

Requirement The STORE= option can be used only if the *path* option is also specified in the REPLAY statement.

Tip The STORE= option is useful if you want to use different templates with a single or multiple output objects.

SETLABEL Statement

Assigns a label to the specified path.

Syntax

SETLABEL *path* '*label*';

Required Arguments

'label'
specifies the text of the label. You can customize labels by inserting BY variable values (#BYVAL), BY variable names (#BYVAR), or BY lines (#BYLINE) into labels that are specified in PROC DOCUMENT steps.

Requirement The label must be enclosed in quotation marks.

See For more information, see “Customizing Labels, Titles, and Footnotes with BY Variables ” on page 146.

path
specifies the location of a link, output object, or directory.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

UNHIDE Statement

Enables the output of a hidden entry to be displayed when it is replayed.

Syntax

UNHIDE *path* <,*path-2*, …*path-n*>;

Required Argument

path
specifies the location of a link, output object, or file.

Requirement Separate multiple paths with commas.

Tip You can use the symbol '^' to represent the current directory and the symbol '^^' to represent the parent directory.

Usage: DOCUMENT Procedure

Working with ODS Documents

Creating an ODS Document

An ODS document is a hierarchical file of output objects that is created from a procedure or data query. The ODS document holds these output objects in their original structures, but you can rearrange the hierarchy and structure of these objects. ODS documents are stored in a proprietary format (a document *store*) and can be viewed only with SAS software. To view or modify what is in the document store, use the DOCUMENT procedure.

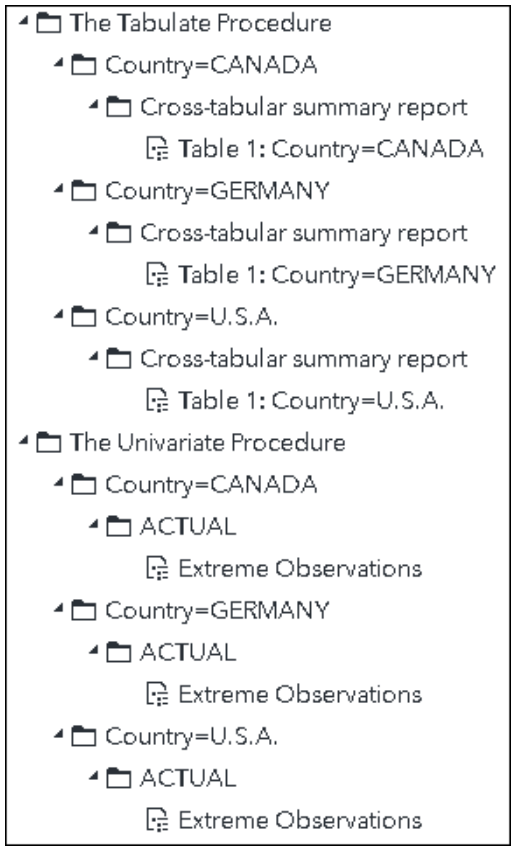
To create an ODS document, you must use the “ODS DOCUMENT Statement”. The following code creates the ODS document Work.Prddoc within a document store:

```
proc sort data=sashelp.prdsale out=prdsale;
   by Country;
run;

ods document name=work.prddoc(write);
proc tabulate data=prdsale;
   by Country;
   var predict;
   class prodtype;
   table prodtype all,
   predict*(min mean max);
run;

ods select ExtremeObs;
proc univariate data=prdsale;
   by Country;
   var actual;
run;
ods document close;
```

Output 6.1 *Work.Prddoc Document*



Items Included in an ODS Document

In an ODS document, each level of the hierarchical file represents a path that refers to the location of a file, link, or output object. An output object can be one of the following:

- table
- graph
- equation
- note
- SAS/GRAPH external graph titles

Items Not Included in an ODS Document

An ODS document does not store the following items:

- GRSEGs (References to GRSEGs, but not GRSEGs themselves, are stored.)
- ODS options
- procedure options

- Report Writing Interface output
- SAS logs
- SAS/GRAPH options
- SAS system options

ODS Document Persistence

An ODS document persists in the SAS system until the document, or the SAS library containing the document, is deleted. An ODS document that was created in the Sasuser library, or in another permanent SAS library, can persist indefinitely. It is considered a permanent archive of SAS procedure output. However, an ODS document that is created in the Work library does not persist longer than the SAS session that created it. For information about SAS libraries, see *SAS Programmer's Guide: Essentials*.

Replaying Graphics

When replaying graphics created by a device driver from the following list, you must also specify a device driver from the list with the DEVICE= option in the GOPTIONS statement:

- ACTIVEX
- JAVA
- JAVAIMG

See "GOPTIONS Statement" in *SAS/GRAPH: Reference* in *SAS/GRAPH: Reference* for more information.

Customizing Labels, Titles, and Footnotes with BY Variables

You can customize labels, titles, and footnotes with these statements by inserting BY variable values (#BYVAL), BY variable names (#BYVAR), or BY lines (#BYLINE) in labels that are specified in the following PROC DOCUMENT statements:

- "OBANOTE Statement" on page 127
- "OBBNOTE Statement" on page 128
- "OBFOOTN Statement" on page 130
- "OBSTITLE Statement" on page 132
- "OBTITLE Statement" on page 134
- "SETLABEL Statement" on page 142

Note: The #BYVAL, #BYVAR, and #BYLINE substitutions show up only for output objects that belong to a BY group. Examples of output objects that do not belong to a BY group are data sets that are imported into a document with the IMPORT TO statement, and notes that are created with the NOTES statement.

To create these substitutions, embed the items in the specified object text string at the position where you want the substitution text to appear. The #BYVAL, #BYVAR, and #BYLINE substitutions have this form:

#BYVAL*n* | #BYVAL(*variable-name*)
substitutes the current value of the specified BY variable for #BYVAL in the text string and displays the value in the label.

Follow these rules when you use #BYVAL in a statement of a PROC DOCUMENT step:

- Specify the variable that is used by #BYVAL in the BY statement.
- Insert #BYVAL in the specified text string at the position where you want the substitution text to appear.
- Follow #BYVAL with a delimiting character, either a space or other non-alphanumeric character (for example, a quotation mark) that ends the text string.
- To immediately follow the #BYVAL substitution with other text and no delimiter, use a trailing dot (as with macro variables).
- Specify the variable with one of the following:

n
specifies which variable in the BY statement #BYVAL should use. The value of *n* indicates the position of the variable in the BY statement.

Example #BYVAL2 specifies the second variable in the BY statement.

variable-name
names the BY variable.

Requirement You must enclose *variable-name* in parentheses.

Tip *variable-name* is not case sensitive.

Example #BYVAL(YEAR) specifies the BY variable, YEAR.

#BYVAR*n* | #BYVAR(*variable-name*)
substitutes the name of the BY variable or label that is associated with the variable (whatever the BY line would normally display) for #BYVAR in the text string and displays the name or label.

Follow these rules when you use #BYVAR in a statement of a PROC DOCUMENT step:

- Specify the variable that is used by #BYVAR in the BY statement.
- Insert #BYVAR in the specified text string at the position where you want the substitution text to appear.
- Follow #BYVAR with a delimiting character, either a space or other non-alphanumeric character (for example, a quotation mark) that ends the text string.

- To immediately follow the #BYVAR substitution with other text and no delimiter, use a trailing dot (as with macro variables).
- Specify the variable with one of the following:

n
specifies the variable in the BY statement that #BYVAR should use. The value of *n* indicates the position of the variable in the BY statement.

Example #BYVAR2 specifies the second variable in the BY statement.

variable-name
names the BY variable.

Requirement You must enclose *variable-name* in parentheses.

Tip *variable-name* is not case sensitive.

Example #BYVAR(SITES) specifies the BY variable SITES.

#BYLINE
substitutes the entire BY line without leading or trailing blanks for #BYLINE in the text string and displays the BY line in the label.

Using WHERE Expressions in PROC DOCUMENT

WHERE expressions in PROC DOCUMENT statements enable you to conditionally perform actions on objects within the document. For example, you can use a WHERE expression with a subsetting variable to help you find common metadata values in ODS documents. You can also use WHERE expressions to only print output objects with specific BY variable values. For an example, see "Example 6: Opening and Listing ODS Documents Using WHERE Expressions" on page 178.

WHERE Expression Syntax

WHERE expressions can be used in the following PROC DOCUMENT statements:

- COPY TO
- DELETE
- LINK
- LIST
- MOVE TO
- REPLAY

For complete documentation about WHERE expressions with any of the preceding statements, see the statement documentation in Chapter 6, "DOCUMENT Procedure," on page 87.

Document paths can be followed by a WHERE clause to further subset the objects matched by a path. The general form of WHERE clause usage is shown below:

document-path(WHERE=(*where-expression*))

(WHERE=(*where-expression-1<operator >*))
conditionally selects a subset of entries in an ODS document.

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

operand
is one of the following:

constant
is a fixed value such as a date literal, a value, or a BY variable value.

SAS function
For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

subsetting variable
is a special type of WHERE expression operand used by the DOCUMENT procedure to help you find common metadata values in ODS documents. Variables available in a WHERE clause are listed in the following table.

Table 6.6 *Table of WHERE Clause Variables*

Name	Description	Example
CDATE	Creation date of the item	The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of 16JUL2004 to the Monthly directory in Work.MyDoc: `move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd))` `  to \ work.mydoc\monthly;` `run;`
CDATETIME	Creation datetime of the item	The following COPY TO statement copies all entries with a creation datetime of May 1, 2003, at 9:30 to the Monthly directory of Work.MyDoc: `copy ^(where=(_cdatetime_ = '01may04:9:30:00'dt))` `  to \work.mydoc\monthly;` `run;`
CTIME	Creation time of the item	The following DELETE statement deletes all entries with a creation time of 9:25:19 PM: `delete ^(where=(_ctime_ = '9:25:19pm't));` `run;`
LABEL	Item label	The following LIST statement lists all tables containing the label 'Type III Model' within the GLM procedure: `list glm(where=(_type_ = 'table' and _label_ ?` `  'Type III Model'));` `run;`

Name	Description	Example
LABELPATH	Labels of all entries in the path	The following LIST statement lists all items containing “Fit Statistics” in the label path: `list gml(where=(_labelpath_ ? "Fit Statistics"))` `  / levels=all;` `run;`
MAX	The last observation	The following REPLAY statement replays all observations except the last observation: `replay class(where=(_obs_ < _max_));`
MDATE	Modification date of the item	The following MOVE TO statement moves all entries of the type 'Graph' with a modification date of 16JUL2004 to the Monthly directory of Work.MyDoc: `move ^(where=(_type_ = 'Graph' and _mdate_ = '16JUL2004'd))` `  to \work.mydoc\monthly;` `run;`
MDATETIME	Modification datetime of the item	The following REPLAY statement replays all entries with a modification datetime of May 1, 2003, at 9:30: `replay ^(where=(_mdatetime_ = '01may04:9:30:00'dt));` `run;`
MIN	The first observation	The following REPLAY statement replays all observations except the first observation: `replay class(where=(_obs_ >  _min_));`
MTIME	Modification time of the item	The following COPY TO statement copies all entries with a modification time of 9:25:19 PM to the Monthly directory of Work.MyDoc: `copy ^(where=(_mtime_ = '9:25:19pm't))` `  to \work.mydoc\monthly;` `run;`
NAME	Item name	The following DELETE statement deletes all entries that contain the name “stemleng” within the GLM procedure: `delete glm(where=(_name_ ? 'stemleng'));`
OBS	The current observation number in an output object	The following REPLAY statement replays all but the first ten observations: `replay class(where=(_obs_ > 10))`
observation-number	The observation number to be replayed	The following REPLAY statement replays the first, third, fifth, seventh, and ninth observations: `replay class(where=(_obs_ in (1,3,5,7,9)));`

Name	Description	Example
observation-variable	The name of an observation	The following REPLAY statement replays all observations where the variable Weight is greater than 100: `replay class(where=(weight>100));`
PATH	Item path	The following LIST statement lists all entries with a path containing the substring 'Anova' at all levels of the current directory: `list ^(where=(_path_ ? 'Anova'));` `run;`
SEQNO	Item sequence number	The following REPLAY statement replays all entries that have a sequence number of 2 in the GLM procedure: `replay glm(where=(_seqno_ = 2));`
TYPE	Item type	The following MOVE TO statement moves all entries of the type 'Graph' with a creation date of July 16, 2004, to the Monthly directory of Work.MyDoc: `move ^(where=(_type_ = 'Graph' and _cdate_ = '16JUL2004'd))` `  to \work.mydoc\monthly;` `run;`
variable	BY variable	The following MOVE TO statement moves all entries where the value of the variable Gender is 'F' to the Monthly directory of Work.MyDoc: `move ^(where=(gender='F')) to \work.mydoc\monthly;` `run;`

operator
compares one variable with a value or another variable. *operator* can be AND, OR NOT, OR, AND NOT, or a comparison operator.

Table 6.7 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to

Symbol	Mnemonic Equivalent	Definition
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Working with Output Objects and Table Templates

PROC DOCUMENT enables you to view information about the output objects that are stored in a document. When a document is created, a reference to the template that output objects use is created. Templates are stored in an ODS document only when you use PROC PRINT, PROC REPORT, PROC TABULATE, or the ODS Graphics procedures. PROC DOCUMENT enables you to view specific information about the output objects' template in your document. You can then use this information to further customize your output by specifying the template that you want to use. For an example, see "Example 8: Using PROC DOCUMENT to Work with Table Templates " on page 188.

Displaying the Template Used by Output Objects

You can use the OBTEMPL statement to display the template of an output object. The OBTEMPL statement has the following form:

OBTEMPL *output-object*;

output-object specifies the pathname of the output object.

By viewing the output object's table template, you can view the name of the table template that is used for your output and the table attributes that are used to customize the output. For graphics, you can view the name of the graphic template that is used for a graphic. It is important to know the name of the default table template so that you can create a customized version. You can use either the TEMPLATE procedure or the ODSTABLE procedure to customize table templates. For more information about the TEMPLATE procedure and the ODSTABLE procedure, see *SAS Output Delivery System: Procedures Guide*.

Examples: DOCUMENT Procedure

Example 1: Navigating the Directory and Listing the Entries

Features:	ODS DOCUMENT statement option NAME= DOC statement option NAME= LIST statement options *entry* LEVELS= DETAILS DIR statement option *path* Procedure output PROC DOCUMENT
ODS destinations:	DOCUMENT , HTML5

Details

This example shows you how to do these tasks:

- name an ODS document
- see what ODS documents exist
- open a document for browsing or editing purposes
- list one or more entries
- change directories

Program

```
data distrdata;
   drop n;
   label Normal_x='Normal Random Variable'
         Exponential_x='Exponential Random Variable';
```

```
   do n=1 to 100;
      Normal_x=10*rannor(53124)+50;
      Exponential_x=ranexp(18746363);
      output;
   end;
run;

ods document name=univ;

title '100 Obs Sampled from a Normal Distribution';
proc univariate data=distrdata noprint;
  var Normal_x;

  histogram Normal_x / normal(noprint) cbarline=grey name='normal';
run;

title '100 Obs Sampled from an Exponential Distribution';

proc univariate data=distrdata noprint;
  var Exponential_x;

  histogram / exp(fill l=3) cfill=yellow midpoints=.05 to 5.55 by .25
              name='exp';
run;

ods document close;
title;

proc document;
   doc;
   doc name=univ;
   list/levels=all;

   dir \Univariate#2\Exponential_x#1\Histogram#1\Exponential#1;
   list;
   list fitquantiles/details;
run;


quit;
```

Program Description

Create the DistrData data set. The DistrData data set contains the statistical information that PROC UNIVARIATE uses to create the histograms.

```
data distrdata;
   drop n;
   label Normal_x='Normal Random Variable'
         Exponential_x='Exponential Random Variable';
   do n=1 to 100;
      Normal_x=10*rannor(53124)+50;
      Exponential_x=ranexp(18746363);
      output;
   end;
run;
```

Create the ODS document Univ and open the DOCUMENT destination. The ODS DOCUMENT statement opens the DOCUMENT destination. The NAME= option assigns the name Univ to the ODS document that contains the information from this PROC UNIVARIATE program. Note that by default Univ is created in the Work library. Assign a libref to create Univ in a permanent library.

```
ods document name=univ;
```

Create a normal distribution histogram. The TITLE statement specifies the title of the normal distribution histogram. The PROC UNIVARIATE step creates a normal distribution histogram from the DistrData data set.

```
title '100 Obs Sampled from a Normal Distribution';
proc univariate data=distrdata noprint;
  var Normal_x;

  histogram Normal_x / normal(noprint) cbarline=grey name='normal';
run;
```

Create an exponential distribution histogram. The TITLE statement specifies the title of the exponential histogram. The PROC UNIVARIATE step creates an exponential distribution histogram from the DistrData data set.

```
title '100 Obs Sampled from an Exponential Distribution';

proc univariate data=distrdata noprint;
  var Exponential_x;

  histogram / exp(fill l=3) cfill=yellow midpoints=.05 to 5.55 by .25
              name='exp';
run;
```

Close the DOCUMENT destination. If the DOCUMENT destination is not closed, no DOCUMENT procedure output can be viewed.

```
ods document close;
title;
```

View the ODS documents, choose an ODS document, and list the entries of the opened ODS document. The DOC statement (with no arguments specified) prints a listing of all of the available documents that are in the SAS system. The DOC statement with the NAME= option specifies the current document, Work.Univ. The LIST statement with the LEVELS=ALL option lists detailed information about all levels of the document Work.Univ.

```
proc document;
   doc;
   doc name=univ;
   list/levels=all;
```

Set the current directory to EXPONENTIAL, list the contents of the EXPONENTIAL directory, select a table, and list the details of the table that you selected. The DIR statement changes the current directory to **\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1**. The path **\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1** was obtained from the listing of the Work.Univ document. The LIST statement (with no arguments) lists the contents of **EXPONENTIAL**. The LIST FITQUANTILES/DETAILS statement specifies that ODS opens the FitQuantiles table and lists its details.

```
   dir \Univariate#2\Exponential_x#1\Histogram#1\Exponential#1;
   list;
   list fitquantiles/details;
```

```
run;
```

Terminate the DOCUMENT procedure. Specify the QUIT statement to terminate the DOCUMENT procedure. If you omit QUIT, then you cannot view DOCUMENT procedure output.

```
quit;
```

Output

Output 6.2 *List of the Entries of the ODS Document Work.Univ and the Properties of Those Entries*

Listing of: \Work.Univ
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\Univariate#1	Dir
2	\Univariate#1\Normal_x#1	Dir
3	\Univariate#1\Normal_x#1\Histogram#1	Dir
4	\Univariate#1\Normal_x#1\Histogram#1\Histogram#1	Graph
5	\Univariate#2	Dir
6	\Univariate#2\Exponential_x#1	Dir
7	\Univariate#2\Exponential_x#1\Histogram#1	Dir
8	\Univariate#2\Exponential_x#1\Histogram#1\Histogram#1	Graph
9	\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1	Dir
10	\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1\ParameterEstimates#1	Table
11	\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1\GoodnessOfFit#1	Table
12	\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1\FitQuantiles#1	Table

Output 6.3 *List of the Entries of the Exponential#1 Entry and the Properties of Those Entries*

Listing of: \Work.Univ\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1
Order by: Insertion
Number of levels: 1

Obs	Path	Type
1	ParameterEstimates#1	Table
2	GoodnessOfFit#1	Table
3	FitQuantiles#1	Table

Output 6.4 *Details of the FitQuantiles#1 Table*

Listing of: \Work.Univ\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1\FitQuantiles#1

Order by: Insertion

Number of levels: 1

Type	Size in Bytes	Created	Modified	Symbolic Link	Template	Label
Table	601	31MAR2020:17:13:23	31MAR2020:17:13:23		base.univariate.FitQuant	Quantiles

Example 2: Managing Entries

Features:

PROC DOCUMENT statement option
NAME=

DIR statement

LIST statement option
LEVELS=

NOTE statement

OBANOTE statement option
SHOW

OBBNOTE statement option
SHOW

OBFOOTN statement option
SHOW

OBPAGE statement

OBSTITLE statement option
SHOW

OBTITLE statement option
SHOW

REPLAY statement

Procedure output
PROC CONTENTS

ODS destinations: DOCUMENT, HTML5

Details

This example shows you how to do these tasks:

- generate PROC CONTENTS output to the DOCUMENT destination
- change the title and footnote of the output
- add object footer and object heading notes to the output
- change the subtitle of the output
- add a note to the document

- add a page break to the output
- specifies that a table containing the output object's headings, titles, and footnotes, be written to the active destinations
- generate a listing of the entries that shows the details for each entry

Program

```
ods document name=class(write);
proc contents data=sashelp.class;
run;
title 'Title Specified by the Global TITLE Statement';
footnote 'Footnote Specified by the Global FOOTNOTE Statement';
ods document close;

proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obtitle Attributes#1 'Title Specified by the OBTITLE Statement';
run;
quit;

proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obbnote Attributes#1 'Object Heading Note Specified by the OBBNOTE
Statement';
run;
quit;

proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obfootn Variables#1 'Change the Global Footnote with the OBFOOTN
Statement';
run;
quit;

proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obanote Attributes#1 'Object Footer Note Specified by the OBANOTE
Statement';
run;
quit;

proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obstitle Attributes#1 'Subtitle Specified by the OBSTITLE
Statement';
run;
quit;

proc document name=class;
   note addnote 'Note added to the document';
 run;
```

```
 quit;

ods html5 file='your_file.html' style=Sapphire;
proc document name=class;
   obpage \Contents#1\DataSet#1\Variables#1;
   replay;
run;
quit;

proc document name=class;
   list/ levels=all details;
   dir \Contents#1\DataSet#1;
   obanote  Attributes#1  show;
   obbnote  Attributes#1   show;
   obfootn  Variables#1  show;
   obstitle Attributes#1   show;
   obtitle  Attributes#1  show;
run;
quit;
```

Program Description

Create the Class document. The NAME= option creates an ODS document named Class. The CONTENTS procedure shows the contents of the SAS data set Sashelp.Class. The entries in the ODS document Class are used in the remainder of this example.

```
ods document name=class(write);
proc contents data=sashelp.class;
run;
title 'Title Specified by the Global TITLE Statement';
footnote 'Footnote Specified by the Global FOOTNOTE Statement';
ods document close;
```

Change the global title. The OBTITLE statement assigns a new title to the Attributes#1 entry. The NAME= option specifies the current ODS document. Note that PROC DOCUMENT is still running after the RUN statement executes. The DIR statement changes the current path to **`\Contents#1\DataSet#1`**. The QUIT statement terminates PROC DOCUMENT.

```
proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obtitle Attributes#1 'Title Specified by the OBTITLE Statement';
run;
quit;
```

Add an object heading note to the output. The OBBNOTE statement assigns an object heading note to the Attributes#1 entry. The NAME= option specifies the current ODS document. The DIR statement changes the current directory to **`\Contents#1\DataSet#1`**. The QUIT statement terminates PROC DOCUMENT.

```
proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obbnote Attributes#1 'Object Heading Note Specified by the OBBNOTE
Statement';
```

```
run;
quit;
```

Change the global footnote. The OBFOOTN statement assigns a new footnote to the Variables#1 entry. The NAME= option specifies the current ODS document. The DIR statement changes the current directory to **\Contents#1\DataSet#1**. The QUIT statement terminates PROC DOCUMENT.

```
proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obfootn Variables#1 'Change the Global Footnote with the OBFOOTN
Statement';
run;
quit;
```

Add an object footer note. The OBANOTE statement assigns an object footer note to the Attributes#1 entry. The NAME= option specifies the current ODS document. The DIR statement changes the current directory to **\Contents#1\DataSet#1**. The QUIT statement terminates PROC DOCUMENT.

```
proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obanote Attributes#1 'Object Footer Note Specified by the OBANOTE
Statement';
run;
quit;
```

Change the subtitle of the output. The OBSTITLE statement changes the subtitle. The subtitle identifies the procedure that produced the output. The NAME= option specifies the current ODS document. The DIR statement changes the current directory to **\Contents#1\DataSet#1**. The QUIT statement terminates PROC DOCUMENT.

```
proc document name=class;
   dir \Contents#1\DataSet#1;
run;
   obstitle Attributes#1 'Subtitle Specified by the OBSTITLE
Statement';
run;
quit;
```

Add a note to the document. The NOTE statement adds a note object named ADDNOTE to the ODS document. The NAME= option specifies the current ODS document. The LIST statement with the LEVELS=ALL option shows a list of entries in the Class document. The QUIT statement terminates PROC DOCUMENT.

```
proc document name=class;
   note addnote 'Note added to the document';
 run;
 quit;
```

Add a page break to the output, create HTML5 output, and replay Variables#1. The ODS HTML5 statement opens the HTML5 destination and creates HTML5 output. The STYLE= option specifies that ODS use the style Sapphire. The OBPAGE statement inserts a page break. The NAME= option specifies the current ODS document. The REPLAY statement replays the Variables#1 object and generates output for all open ODS destinations. The QUIT statement terminates PROC DOCUMENT.

```
ods html5 file='your_file.html' style=Sapphire;
proc document name=class;
   obpage \Contents#1\DataSet#1\Variables#1;
   replay;
run;
quit;
```

Generate a table containing contextual information for each title, footnote, and note. The NAME= option specifies the current ODS document. The LIST statement with the LEVELS=ALL option shows a list of entries in the Class document. The DIR statement changes the current directory to **\Contents#1\DataSet#1**. The SHOW option generates a table containing contextual information for each title, footnote, and note. The QUIT statement terminates PROC DOCUMENT.

```
proc document name=class;
   list/ levels=all details;
   dir \Contents#1\DataSet#1;
   obanote  Attributes#1  show;
   obbnote  Attributes#1   show;
   obfootn  Variables#1  show;
   obstitle Attributes#1   show;
   obtitle  Attributes#1  show;
run;
quit;
```

Output

Output 6.5 *Global Title, Global Footnote, Subtitle, Object Heading Note, Object Footer Note, and Note*

Title Specified by the OBTITLE Statement

Subtitle Specified by the OBSTITLE Statement

Object Heading Note Specified by the OBBNOTE Statement

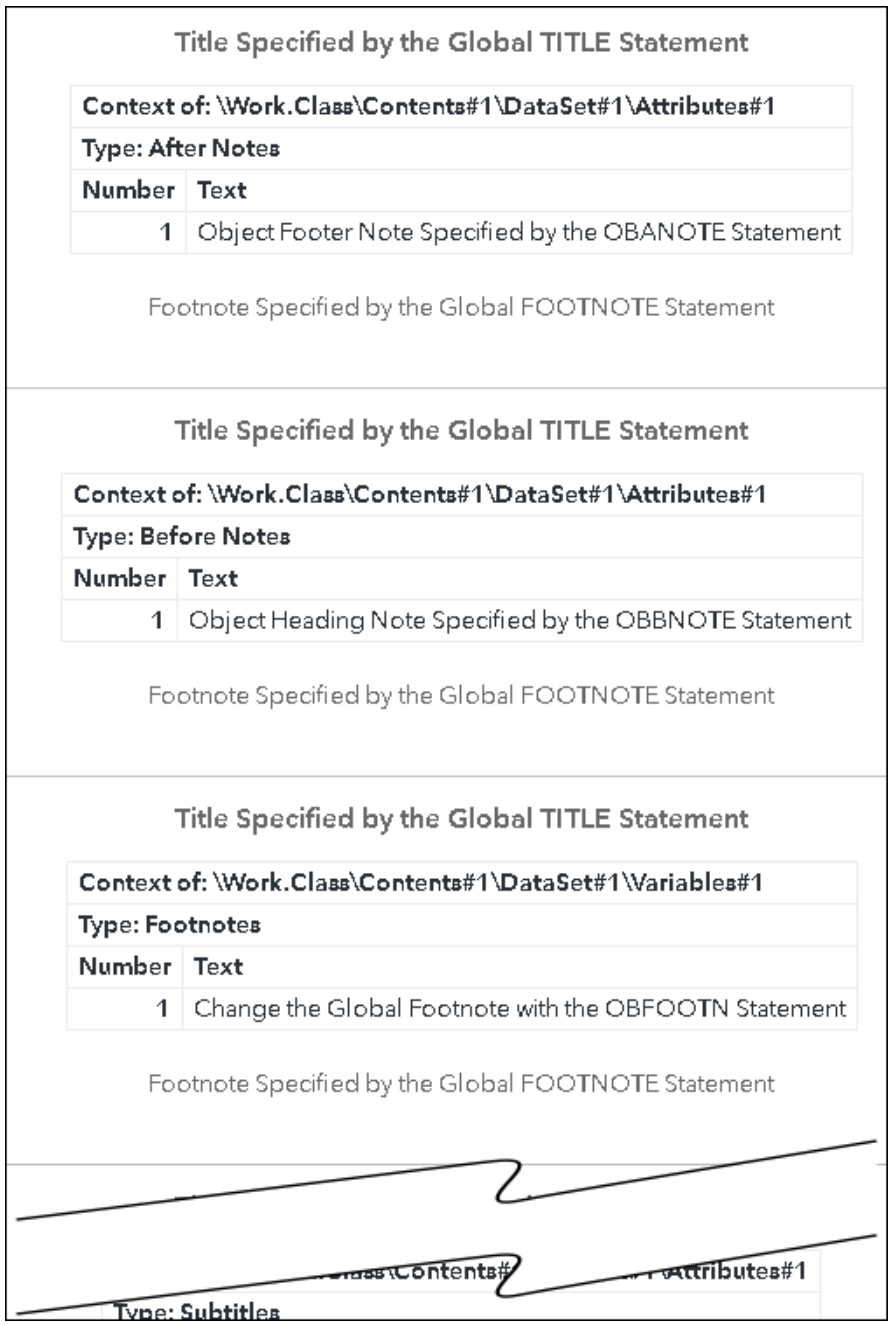
Data Set Name	SASHELP.CLASS	**Observations**	19
Member Type	DATA	**Variables**	5
Engine	V9	**Indexes**	0
Created	10/22/2019 10:38:47	**Observation Length**	40
Last Modified	10/22/2019 10:38:47	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label	Student Data		
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	us-ascii ASCII (ANSI)		

Object Footer Note Specified by the OBANOTE Statement

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	1632
Obs in First Data Page	19
Number of Data Set Repairs	0
Filename	/opt/sas/viya/home/SASFoundation/sashelp/class.sas7bdat
Release Created	V.0305M0
Host Created	Linux
Inode Number	110950457
Access Permission	rw-r--r--
Owner Name	sas
File Size	

Change the Global Footnote with the GLFOOTN Statement

Output 6.6 *Context Tables Generated By Specifying the SHOW Option*



Title Specified by the Global TITLE Statement

Context of: \Work.Class\Contents#1\DataSet#1\Attributes#1	
Type: After Notes	
Number	Text
1	Object Footer Note Specified by the OBANOTE Statement

Footnote Specified by the Global FOOTNOTE Statement

Title Specified by the Global TITLE Statement

Context of: \Work.Class\Contents#1\DataSet#1\Attributes#1	
Type: Before Notes	
Number	Text
1	Object Heading Note Specified by the OBBNOTE Statement

Footnote Specified by the Global FOOTNOTE Statement

Title Specified by the Global TITLE Statement

Context of: \Work.Class\Contents#1\DataSet#1\Variables#1	
Type: Footnotes	
Number	Text
1	Change the Global Footnote with the OBFOOTN Statement

Footnote Specified by the Global FOOTNOTE Statement

Example 3: Listing BY-Group Entries

Features: LIST statement options
- BYGROUPS
- LEVELS
- LIST

PROC DOCUMENT statement option
- NAME=

OBTEMPL statement

Procedure output
PROC DOCUMENT
PROC STANDARD

ODS destinations: DOCUMENT, HTML5

Details

This example shows you how to do these tasks:

- generate PROC STANDARD output to the DOCUMENT destination
- view the table template that describes how to display the PROC STANDARD output
- create an ODS document
- open an ODS document
- list the BY-group entries in an ODS document

Program

```
ods document name=mydocument(write);

data score;
   input Student Section Test1-Test3;
   stest1=test1;
   stest2=test2;
   stest3=test3;
   datalines;
238900545 1 94 91 87
254701167 1 95 96 97
238806445 2 91 86 94
999002527 2 80 76 78
263924860 1 92 40 85
459700886 2 75 76 80
416724915 2 66 69 72
999001230 1 82 84 80
242760674 1 75 76 70
990001252 2 51 66 91
;
run;

proc sort data=score;
   by Section Student;
run;

proc standard mean=80 std=5 out=StndScore print;

   by section student;
   var stest1-stest3;
run;
```

```
ods document close;

proc document name=mydocument;
title "Listing of MyDocument Using the BYGROUPS Option";
run;
   list/ levels=all bygroups;
run;
title "Table Template for the Output Object Standard#1";
   obtempl \Standard#1\ByGroup1#1\Standard#1;
run;

quit;
```

Program Description

Create the ODS document MyDocument and open the DOCUMENT destination. The ODS DOCUMENT statement with the NAME= option specified opens the ODS document MyDocument and provides Write access as well as Read access. Note that by default MyDocument is created in the Work library. Assign a libref to create MyDocument in a permanent library.

```
ods document name=mydocument(write);
```

Create and sort the Score data set. This data set contains test scores for students who took two tests and a final exam. The SORT procedure sorts the data set by the BY variables Section and Student.

```
data score;
   input Student Section Test1-Test3;
   stest1=test1;
   stest2=test2;
   stest3=test3;
   datalines;
238900545 1 94 91 87
254701167 1 95 96 97
238806445 2 91 86 94
999002527 2 80 76 78
263924860 1 92 40 85
459700886 2 75 76 80
416724915 2 66 69 72
999001230 1 82 84 80
242760674 1 75 76 70
990001252 2 51 66 91
;
run;

proc sort data=score;
   by Section Student;
run;
```

Generate the standardized data and create the output data set StndScore. PROC STANDARD uses a mean of 80 and a standard deviation of 5 to standardize the values. OUT= identifies StndScore as the data set to contain the standardized values. The PRINT option prints the statistics.

```
proc standard mean=80 std=5 out=StndScore print;
```

Create the standardized values for each BY group and specify the variables to standardize. The BY statement standardizes the values separately by section number and student ID. The VAR statement specifies the variables to standardize and their order in the output.

```
   by section student;
   var stest1-stest3;
run;
```

Close the DOCUMENT destination. If the DOCUMENT destination is not closed, no DOCUMENT procedure output can be viewed.

```
ods document close;
```

Open the ODS document MyDocument, list the entries, and view the table template that determines how the PROC STANDARD output is displayed. The PROC DOCUMENT statement with the NAME= option specified opens the ODS document Work.MyDocument. The LIST statement with the LEVELS=ALL option lists detailed information about all levels of the document Work.MyDocument. The BYGROUPS option creates columns in the list statement output for BY group information. The names of the columns with the BY group information are the names of the BY variables, Section and Student. The OBTEMPL statement writes the table template that is associated with the output object Standard#1 to the HTML5 destination. The ODS DOCUMENT CLOSE statement closes the DOCUMENT destination. If the DOCUMENT destination is not closed, no DOCUMENT procedure output can be viewed. If you omit LEVELS=ALL, then no entry list is created. This is because ODS cannot find any BY groups at the directory level and only BY groups are listed when the BYGROUPS option is specified.

To see what the output looks like if you omit the BYGROUPS option, see Output 6.17 on page 168.

```
proc document name=mydocument;
title "Listing of MyDocument Using the BYGROUPS Option";
run;
   list/ levels=all bygroups;
run;
title "Table Template for the Output Object Standard#1";
   obtempl \Standard#1\ByGroup1#1\Standard#1;
run;
```

Terminate the DOCUMENT procedure. Specify the QUIT statement to terminate the DOCUMENT procedure. If you omit QUIT, then you cannot view DOCUMENT procedure output.

```
quit;
```

Output

Without the BYGROUPS option specified, there are only three columns for this output: Obs, Path, and Type. All levels and all entries of Work.MyDocument are displayed.

Output 6.7 *Listing of Work.MyDocument without the BYGROUPS Option Specified*

Listing of: \Work.Mydocument
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\Standard#1	Dir
2	\Standard#1\ByGroup1#1	Dir
3	\Standard#1\ByGroup1#1\Standard#1	Table
4	\Standard#1\ByGroup2#1	Dir
5	\Standard#1\ByGroup2#1\Standard#1	Table
6	\Standard#1\ByGroup3#1	Dir
7	\Standard#1\ByGroup3#1\Standard#1	Table
8	\Standard#1\ByGroup4#1	Dir
9	\Standard#1\ByGroup4#1\Standard#1	Table
10	\Standard#1\ByGroup5#1	Dir
11	\Standard#1\ByGroup5#1\Standard#1	Table
12	\Standard#1\ByGroup6#1	Dir
13	\Standard#1\ByGroup6#1\Standard#1	Table
14	\Standard#1\ByGroup7#1	Dir
15	\Standard#1\ByGroup7#1\Standard#1	Table
16	\Standard#1\ByGroup8#1	Dir
17	\Standard#1\ByGroup8#1\Standard#1	Table
18	\Standard#1\ByGroup9#1	Dir
19	\Standard#1\ByGroup9#1\Standard#1	Table
20	\Standard#1\ByGroup10#1	Dir
21	\Standard#1\ByGroup10#1\Standard#1	Table

With the BYGROUPS option specified there are now five columns. The additional columns, named Section and Student, were created by the BYGROUPS option. The BY variable names become the names of the columns. Only the entries containing BY group information are displayed. The entries that are directories are not displayed because they do not contain any actual BY group information.

Output 6.8 *Listing of Work.MyDocument with the BYGROUPS Option Specified*

Listing of MyDocument Using the BYGROUPS Option

Listing of: \Work.Mydocument

Order by: Insertion

Number of levels: All

Obs	Path	Type	Section	Student
1	\Standard#1\ByGroup1#1\Standard#1	Table	1.000000	238900545
2	\Standard#1\ByGroup2#1\Standard#1	Table	1.000000	242760674
3	\Standard#1\ByGroup3#1\Standard#1	Table	1.000000	254701167
4	\Standard#1\ByGroup4#1\Standard#1	Table	1.000000	263924860
5	\Standard#1\ByGroup5#1\Standard#1	Table	1.000000	999001230
6	\Standard#1\ByGroup6#1\Standard#1	Table	2.000000	238806445
7	\Standard#1\ByGroup7#1\Standard#1	Table	2.000000	416724915
8	\Standard#1\ByGroup8#1\Standard#1	Table	2.000000	459700886
9	\Standard#1\ByGroup9#1\Standard#1	Table	2.000000	990001252
10	\Standard#1\ByGroup10#1\Standard#1	Table	2.000000	999002527

Output 6.9 *Table Template for the Output Object Standard#1*

```
proc template;
   define table Base.Standard;
      notes "Table definition for PROC Standard.";
      column name mean std n label;

      define name;
         header = "Name";
         varname = Name;
         style = RowHeader;
      end;

      define mean;
         header = "Mean";
         format = D12.;
         varname = Mean;
      end;

      define std;
         header = "/Standard/Deviation";
         format = D12.;
         varname = stdDev;
      end;

      define n;
         header = "N";
         format = best.;
      end;

      define label;
         header = "Label";
         varname = Label;
      end;
      required_space = 3;
      byline;
      wrap;
   end;
run;
```

Example 4: Create a Table of Dynamic Values

Features:	PROC DOCUMENT statement option NAME= DIR statement DOC statement LIST statement OBDYNAM statement Procedure output PROC DOCUMENT PROC TABULATE
ODS destinations:	DOCUMENT, HTML5

Details

Dynamics are dynamic (run-time) values that an application associates with an output object or data object. Knowledgeable users will recognize a dynamic as what is inserted into a PROC TEMPLATE DYNAMIC reference when a template and data object are paired during rendering. This example demonstrates how to print a table that shows what dynamics are stored in a specific output object.

Program: Create the ODS Document

```
ods document name=tab(write);
proc tabulate data=sashelp.class;
   class sex age;
   var height;
   table sex, age*height*(mean median);
   run;
ods document close;

proc document;
   doc name=tab;
   list / levels=all;
quit;
```

Program Description

Create the ODS document Tab. The ODS DOCUMENT statement opens the DOCUMENT destination. The NAME= option assigns the name Tab to the ODS document that contains the information from the PROC TABULATE program. Note that by default Tab is created in the Work library. Assign a libref to create Tab in a permanent library. Close the DOCUMENT destination. If the DOCUMENT destination is not closed, no DOCUMENT procedure output can be viewed.

```
ods document name=tab(write);
proc tabulate data=sashelp.class;
   class sex age;
   var height;
   table sex, age*height*(mean median);
   run;
ods document close;
```

List the entries of the newly created ODS document Tab. The DOC statement with the NAME= option specifies the current document, Work.Tab. The LIST statement with the LEVELS=ALL option lists detailed information about all levels of the document Work.Tab.

```
proc document;
   doc name=tab;
   list / levels=all;
quit;
```

Output 6.10 *Listing of the Tab Document*

Listing of: \Work.Tab\		
Order by: Insertion		
Number of levels: All		
Obs	Path	Type
1	\Tabulate#1	Dir
2	\Tabulate#1\Report#1	Dir
3	\Tabulate#1\Report#1\Table#1	Table

Program: Create the Table of Dynamic Values

```
proc document name=tab;
   obdynam \Tabulate#1\Report#1\Table#1;
   run;
quit;
```

```
proc document name=tab;
   dir \Tabulate#1\Report#1;
   obdynam Table#1;
quit;
```

Program Description

Copy the path from the listing and paste it into the OBDYNAM statement to display the dynamic variables: The DIR statement changes the current directory to `Tabulate#1\Rreport#1`. The path `Tabulate#1\Report#1` was obtained from the listing of the Work.Tab document Output 6.19 on page 171. The OBDYNAM statement creates a table that displays the dynamics for the Table#1 output object.

```
proc document name=tab;
   obdynam \Tabulate#1\Report#1\Table#1;
   run;
quit;
```

Alternatively, you can set the directory and then specify only the last level of the path. This next step is equivalent to the preceding step:

```
proc document name=tab;
   dir \Tabulate#1\Report#1;
   obdynam Table#1;
quit;
```

Output 6.11 *Table of Dynamics for the Table#1 Output Object*

Dynamics for: \Work.Tab\Tabulate#1\Report#1\Table#1			
Name	**Value**	**Type**	**Namespace**
ORDER	3	Column	Sex
COUNT_MISSING	0	Column	Sex
GROUPBY_RAW	1	Column	Sex
PRELOAD_FORMAT	0	Column	Sex
MULTILABEL_FORMAT	0	Column	Sex
NUM_LEVELS	2	Column	Sex
ORDER	3	Column	Age
COUNT_MISSING	0	Column	Age
GROUPBY_RAW	0	Column	Age
PRELOAD_FORMAT	0	Column	Age
MULTILABEL_FORMAT	0	Column	Age
NUM_LEVELS	6	Column	Age

Example 5: Output Dynamics to a Data Set, Modify the Dynamics, and Use Them in Subsequent Graphs

Features:

PROC DOCUMENT statement option
- NAME=

REPLAY statement option
- DYNAMDATA=

LIST statement

OBDYNAM statement

Procedure output
- PROC DOCUMENT
- PROC REG

ODS destinations: DOCUMENT, OUTPUT, HTML5

Details

The following example creates a data set that contains the dynamic variables for a fit plot graph created by PROC REG. The dynamic variables are then modified and saved in a new data set. The new data set is used to replay the fit plot.

Program: Create the ODS Document

```
ods graphics on;

ods document name=Mydoc (write);
proc reg data=sashelp.class;
   ods select fitplot;
   model weight=height;
quit;
ods document close;

proc document name=Mydoc;
   list / levels=all;
quit;
```

Program Description

Create the ODS document Mydoc. The ODS DOCUMENT statement opens the DOCUMENT destination. The NAME= option assigns the name Mydoc to the ODS document that contains the information from the PROC REG procedure. Close the DOCUMENT destination. If the DOCUMENT destination is not closed, no DOCUMENT procedure output can be viewed.

```
ods graphics on;

ods document name=Mydoc (write);
proc reg data=sashelp.class;
   ods select fitplot;
   model weight=height;
quit;
ods document close;
```

List the entries of the opened ODS document Mydoc. The LIST statement with the LEVELS=ALL option lists detailed information about all levels of the document Work.Mydoc.

```
proc document name=Mydoc;
   list / levels=all;
quit;
```

Output 6.12 *Listing of the ODS Document Mydoc*

Listing of: \Work.Mydoc

Order by: Insertion

Number of levels: All

Obs	Path	Type
1	\Reg#1	Dir
2	\Reg#1\MODEL1#1	Dir
3	\Reg#1\MODEL1#1\ObswiseStats#1	Dir
4	\Reg#1\MODEL1#1\ObswiseStats#1\Weight#1	Dir
5	\Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\FitPlot#1	Graph

Program: Create the SAS Data Set of Dynamic Variables

```
proc document name=Mydoc;
   ods output dynamics=Dynamics;
   obdynam \Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\FitPlot#1;
quit;

proc print noobs data=Dynamics;
run;
```

Program Description

Display the dynamic variables for the fit plot and store them in a SAS data set. Both the data object and the dynamic variables are stored in the ODS document. The OBDYNAM statement displays the dynamics, and the ODS OUTPUT statement stores them in a data set named Dynamics. The path specified in the OBDYNAM statement was copied from the results produced by the LIST statement.

```
proc document name=Mydoc;
   ods output dynamics=Dynamics;
   obdynam \Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\FitPlot#1;
quit;
```

View the dynamic variables. You can use PROC PRINT to print the data set Dynamics, which contains the dynamic variables.

```
proc print noobs data=Dynamics;
run;
```

The Dynamics data set shows that the values of the dynamic variables are captured in two variables, cValue1 and nValue1. When the dynamic variable is numeric, nValue1 captures its numeric value, and cValue1 captures its formatted value. When the dynamic variable is character, nValue1 is missing, and cValue1 captures its formatted value.

Output 6.13 *Output Data Set of the Dynamic Variables for the Fit Plot Graph*

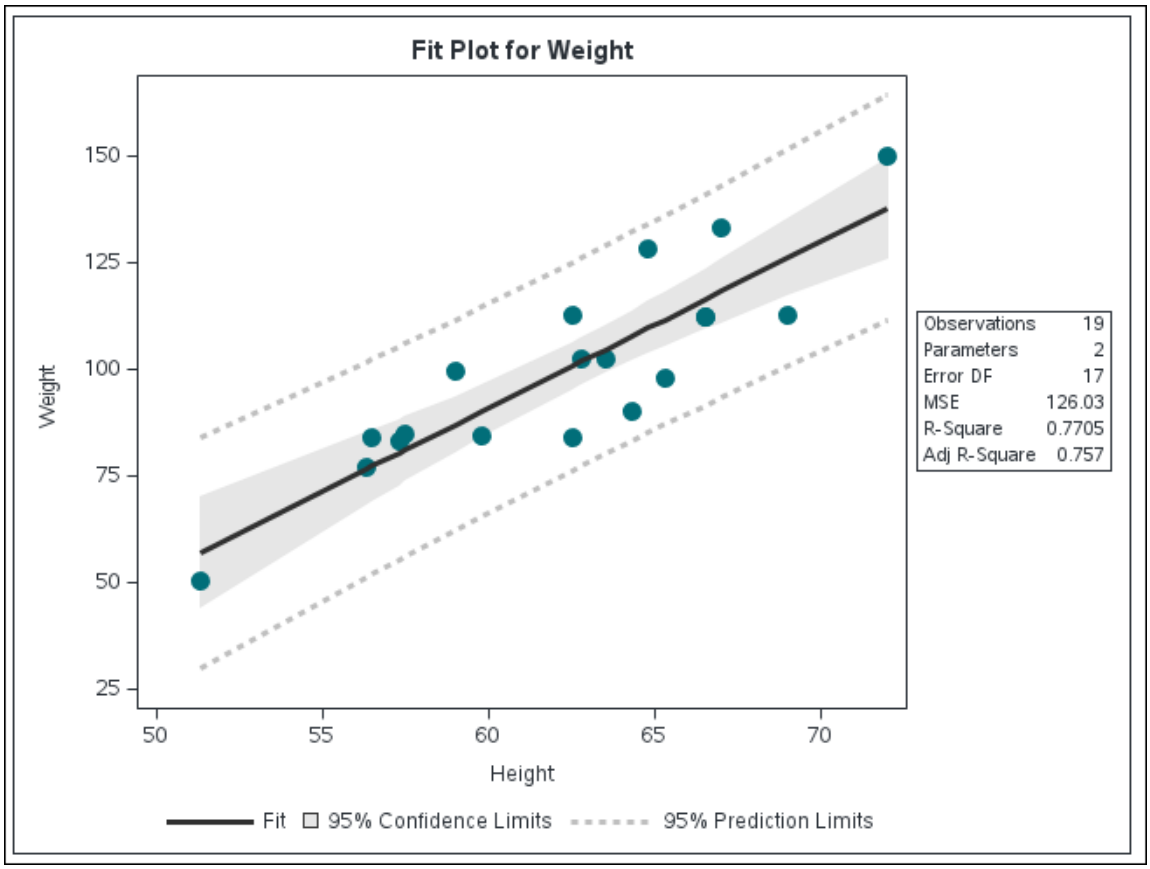
Dynamics for: \Work.Mydoc\Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\FitPlot#1

Name	Value	Type	Namespace
_SHOWCLM	1	Data	
_SHOWCLI	1	Data	
_WEIGHT	0	Data	
_SHOWSTATS	1	Data	
_NSTATSCOLS	2	Data	
_SHOWNOBS	1	Data	
_NOBS	19	Data	
_SHOWTOTFREQ	0	Data	
_TOTFREQ	19	Data	
_SHOWNPARM	1	Data	
_NPARM	2	Data	
_SHOWEDF	1	Data	
_EDF	17	Data	
_SHOWMSE	1	Data	
_MSE	126.02868962	Data	
_SHOWRSQUARE	1	Data	
_RSQUARE	0.7705068427	Data	
_SHOWADJRSQ	1	Data	
_ADJRSQ	0.7570072452	Data	
_SHOWSSE	0	Data	
_SSE	2142.4877235	Data	
_SHOWDEPMEAN	0	Data	
_DEPMEAN	100.02631579	Data	
_SHOWCV	0	Data	
_CV			
	93.78039	Data	

Replay the fit plot using the dynamic variables from the Dynamics data set. The REPLAY statement with the DYNAMDATA= option specified replays the fit plot and uses the dynamic variables from the Dynamics data set. Because no modifications were made, the replayed fit plot is the same as the original replayed fit plot. See Output 6.23 on page 176.

```
proc document name=Mydoc;
   replay \Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\FitPlot#1 /
          dynamdata=Dynamics;
quit;
```

Output 6.14 *Fit Plot Using Original Dynamic Variable Values*



Program: Modify the Values of the Dynamic Variables

```
data Dynamics2;
   set Dynamics;
   if label1 = '_SHOWNPARM'
      then nvalue1 = 0;
   if label1 = '_SHOWAIC'
      then nvalue1 = 1;
   if label1 = '_SHOWSBC'
      then nvalue1 = 1;
   if label1 = '_SHOWDEPMEAN'
      then nvalue1 = 1;
   if label1 in ('_RSQUARE'
                    '_ADJRSQ')
      then nvalue1 =
           round(nvalue1, 0.001);
run;

proc document name=Mydoc;
   replay \Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\FitPlot#1 /
          dynamdata=Dynamics2;
quit;
```

Program Description

Modify the values of the dynamic variables and store them in a data set named Dynamics2. You can modify the values of the dynamic variables and store them in a new data set.

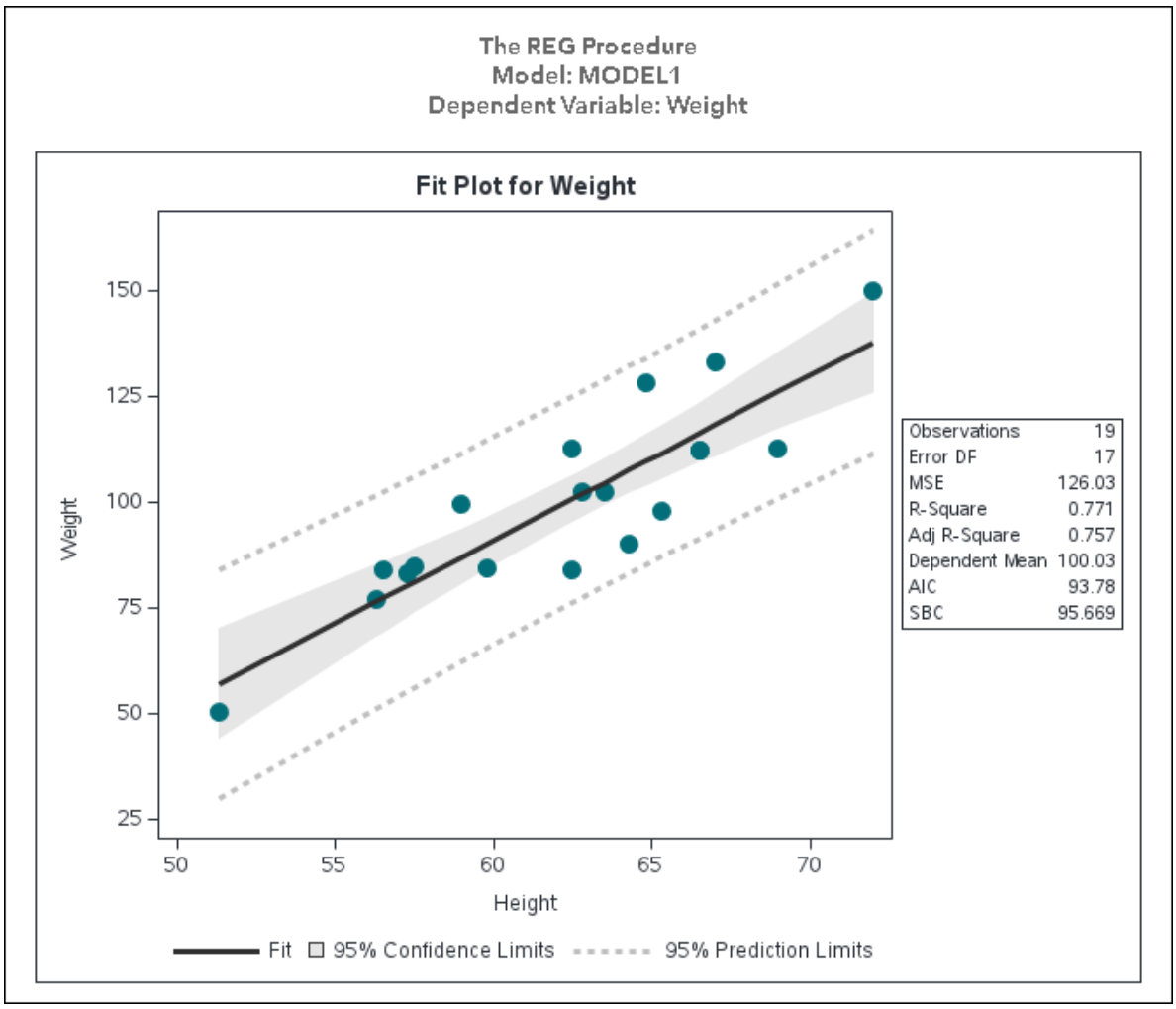
```
data Dynamics2;
   set Dynamics;
   if label1 = '_SHOWNPARM'
      then nvalue1 = 0;
   if label1 = '_SHOWAIC'
      then nvalue1 = 1;
   if label1 = '_SHOWSBC'
      then nvalue1 = 1;
   if label1 = '_SHOWDEPMEAN'
      then nvalue1 = 1;
   if label1 in ('_RSQUARE'
                      '_ADJRSQ')
      then nvalue1 =
           round(nvalue1, 0.001);
run;
```

Replay the graph and explicitly specify that the values of the dynamic variables come from the ODS output data set Dynamics2. By default, the dynamic variables used are the ones that are stored in the ODS document. The DYNAMDATA= option specifies that the dynamic variables come from the data set Dynamics2.

```
proc document name=Mydoc;
   replay \Reg#1\MODEL1#1\ObswiseStats#1\Weight#1\FitPlot#1 /
          dynamdata=Dynamics2;
quit;
```

The following fit plot uses new dynamic variables. Notice that the values in the statistics table have changed.

Output 6.15 *Fit Plot Using Modified Dynamic Variable Values*



Example 6: Opening and Listing ODS Documents Using WHERE Expressions

Features: PROC DOCUMENT statements
- PROC DOCUMENT
- LIST statement
- REPLAY statement

ODS PDF statement

ODS DOCUMENT statement

PROC UNIVARIATE

Data set: DistrData

Note: See "Creating the Univ ODS Document" on page 909 for the SAS code that creates the Univ ODS document.

Details

This example shows you how to do these tasks:

- open an ODS document
- replay a table and send the output to the PDF destination
- list specific entries in an ODS document by using WHERE expressions
- list the details of a specified entry
- replay an ODS document to a PDF file

Program

```
options nodate nonumber;

ods pdf file='file-path/your_file.pdf';

proc document name=univ;

list ^ (where=(_type_ = 'Graph' or _type_ = 'Table') ) / levels=all;
replay univariate#1\Normal_x#1\Histogram#1\Histogram#1;


list \Work.Univ\Univariate#2\Exponential_x#1\Histogram#1\
     Exponential#1\FitQuantiles#1
     / details;
replay \Work.Univ\Univariate#2\Exponential_x#1\Histogram#1\
       Exponential#1\FitQuantiles#1;
run;

quit;
ods pdf close;
```

Program Description

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. The NONUMBER option suppresses the printing of page numbers.

```
options nodate nonumber;
```

Specify that you want to replay the output to a PDF file. The ODS PDF statement opens the PDF file that contains the replayed histogram. The FILE= statement sends all output objects to the external file that you specify. HTML5 output is created by default.

```
ods pdf file='file-path/your_file.pdf';
```

Open the ODS document Work.Univ. The PROC DOCUMENT statement with the NAME= option specified opens the ODS document Work.Univ for updates.

WORK.Univ was created in the example “Navigating the directory and Listing the Entries”.

```
proc document name=univ;
```

List the entries that are associated with the current document and replay a histogram. By using a WHERE expression, the LIST statement lists only entries that are graphs or tables. The LEVELS=ALL option specifies that detailed information about all levels be shown. The ^ symbol represents the current directory. The REPLAY statement replays the Histogram#1 entry to all open ODS destinations.

```
list ^ (where=(_type_ = 'Graph' or _type_ = 'Table') ) / levels=all;
replay univariate#1\Normal_x#1\Histogram#1\Histogram#1;
```

List the details of the FitQuantiles table, and replay the FitQuantiles table. The LIST statement with the DETAILS option specifies the listing of the properties of the entry FitQuantiles table. The REPLAY statement replays FITQUANTILES to open destinations.

```
list \Work.Univ\Univariate#2\Exponential_x#1\Histogram#1\
     Exponential#1\FitQuantiles#1
     / details;
replay \Work.Univ\Univariate#2\Exponential_x#1\Histogram#1\
       Exponential#1\FitQuantiles#1;
run;
```

Terminate the DOCUMENT procedure and close the PDF destination. Specify the QUIT statement to terminate the DOCUMENT procedure. If you omit QUIT, then you cannot view DOCUMENT procedure output. The ODS PDF CLOSE statement closes the PDF destination and all the files that are associated with it. If you do not close the destination, then you cannot view the files.

```
quit;
ods pdf close;
```

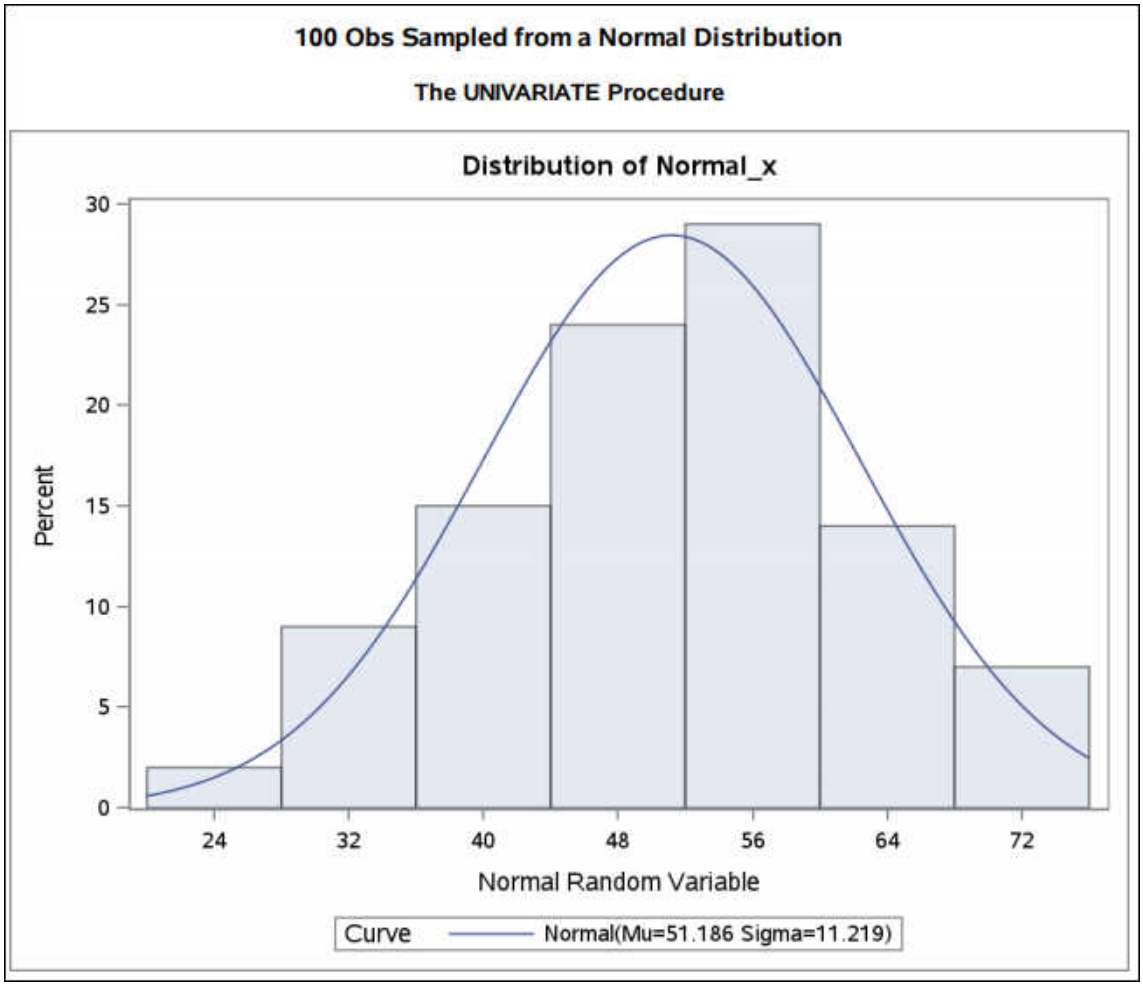
Output

This display is page 1 of the ODS document Work.Univ that was sent to the PDF destination.

Output 6.16 *List of the Graphs and Tables Found in Work.Univ, Viewed in Acrobat Reader*

Listing of: \Work.Univ\(where=((_type_='Graph') or (_type_='Table')))		
Order by: Insertion		
Number of levels: All		
Obs	**Path**	**Type**
1	\Univariate#1\Normal_x#1\Histogram#1\Histogram#1	Graph
2	\Univariate#2\Exponential_x#1\Histogram#1\Histogram#1	Graph
3	\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1\ParameterEstimates#1	Table
4	\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1\GoodnessOfFit#1	Table
5	\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1\FitQuantiles#1	Table
6	\Univariate#3\Normal_x#1\Histogram#1\Histogram#1	Graph
7	\Univariate#4\Exponential_x#1\Histogram#1\Histogram#1	Graph
8	\Univariate#4\Exponential_x#1\Histogram#1\Exponential#1\ParameterEstimates#1	Table
9	\Univariate#4\Exponential_x#1\Histogram#1\Exponential#1\GoodnessOfFit#1	Table
10	\Univariate#4\Exponential_x#1\Histogram#1\Exponential#1\FitQuantiles#1	Table
11	\Univariate#5\Normal_x#1\Histogram#1\Histogram#1	Graph
12	\Univariate#6\Exponential_x#1\Histogram#1\Histogram#1	Graph
13	\Univariate#6\Exponential_x#1\Histogram#1\Exponential#1\ParameterEstimates#1	Table
14	\Univariate#6\Exponential_x#1\Histogram#1\Exponential#1\GoodnessOfFit#1	Table
15	\Univariate#6\Exponential_x#1\Histogram#1\Exponential#1\FitQuantiles#1	Table

Output 6.17 *Replayed Normal Distribution Histogram*



Output 6.18 *Details of the FitQuantiles#1 Table*

Listing of: \Work.Univ\Univariate#2\Exponential_x#1\Histogram#1\Exponential#1\FitQuantiles#1							
Order by: Insertion							
Number of levels: 1							
Type	Size in Bytes	Created	Modified	Symbolic Link	Template	Label	Page Break
Table	601	02APR2020:14:50:27	02APR2020:14:50:27		base.univariate.FitQuant	Quantiles	

Output 6.19 *Replayed FitQuantiles#1 Table*

100 Obs Sampled from an Exponential Distribution

The UNIVARIATE Procedure
Fitted Exponential Distribution for Exponential_x (Exponential Random Variable)

Quantiles for Exponential Distribution		
	Quantile	
Percent	Observed	Estimated
1.0	0.00560	0.00924
5.0	0.05600	0.04717
10.0	0.06979	0.09690
25.0	0.30030	0.26458
50.0	0.62936	0.63749
75.0	1.20484	1.27497
90.0	2.08322	2.11768
95.0	3.00117	2.75517
99.0	5.07829	4.23536

Example 7: Restructuring Output

Features: PROC DOCUMENT statements
- DOC
- DIR
- COPY TO
- MAKE
- PROC DOCUMENT
- REPLAY
- SETLABEL

ODS PDF statement

ODS DOCUMENT statement

ODS SELECT statement

Details

PROC DOCUMENT allows you to create a new, modified ODS document in a SAS program.

The following code creates the ODS document Work.Prddoc, which we are working with. The ODS SELECT statement specifies that only the output ExtremeObs (which is one of the output objects created by PROC UNIVARIATE) is added to the document. All of the output objects created by PROC TABULATE are added to the document.

```
ods html5 close;
proc sort data=sashelp.prdsale out=prdsale;
   by Country;
run;
ods document name=work.prddoc(write);
proc tabulate data=prdsale;
   by Country;
   var predict;
   class prodtype;
   table prodtype all,
   predict*(min mean max);
run;
ods select ExtremeObs;
proc univariate data=prdsale;
   by Country;
   var actual;
run;
ods document close;
```

Program

```
ods html5 close;

proc
document;

   doc
name=Work.Newprd(WRITE);

   dir
\;


   make
USA;

   setlabel USA 'United
States';
   copy \Work.Prddoc\Univariate#1\ByGroup3#1\ACTUAL#1\ExtremeObs#1 TO
USA#1;
   copy \Work.Prddoc\Tabulate#1\ByGroup3#1\Report#1\Table#1 TO
USA#1;
   setlabel USA#1\Table#1 'Predicted Sales';


   make
Germany;
```

```
   setlabel Germany
'Germany';
   copy \Work.Prddoc\Univariate#1\ByGroup2#1\ACTUAL#1\ExtremeObs#1 TO
Germany#1;
   copy \Work.Prddoc\Tabulate#1\ByGroup2#1\Report#1\Table#1 TO
Germany#1;
   setlabel Germany#1\Table#1 'Predicted Sales';


   make
Canada;

   setlabel Canada
'Canada';
   copy \Work.Prddoc\Univariate#1\ByGroup1#1\ACTUAL#1\ExtremeObs#1 TO
Canada#1;
   copy \Work.Prddoc\Tabulate#1\ByGroup1#1\Report#1\Table#1 TO
Canada#1;
   setlabel Canada#1\Table#1 'Predicted
Sales';


quit;
```

```
ods pdf file='file-path/CustomPrd.pdf' contents=yes;

proc document name=work.Newprd;
   replay;
run;
quit;

ods _all_ close;
```

Program Description

Close the HTML5 destination. The HTML5 destination is open by default. Closing the HTML5 destination saves system resources.

```
ods html5 close;
```

Create the ODS document Newprd. The DOC statement with the WRITE option specified creates a new document in the Work directory.

```
proc
document;

   doc
name=Work.Newprd(WRITE);

   dir
\;
```

Create a new folder and copy output objects into it. The MAKE statement creates the folder "USA". The SETLABEL statement specifies the text of the label. The COPY TO statements copy the output objects ExtremeObs#1 and Table#1 to the new folder.

```
   make
USA;

   setlabel USA 'United
States';
   copy \Work.Prddoc\Univariate#1\ByGroup3#1\ACTUAL#1\ExtremeObs#1 TO
USA#1;
   copy \Work.Prddoc\Tabulate#1\ByGroup3#1\Report#1\Table#1 TO
USA#1;
   setlabel USA#1\Table#1 'Predicted Sales';
```

Create a new folder and copy output objects into it. The MAKE statement creates the folder "Germany". The SETLABEL statement specifies the text of the label. The COPY TO statements copy the output objects ExtremeObs#1 and Table#1 to the new folder.

```
   make
Germany;

   setlabel Germany
'Germany';
   copy \Work.Prddoc\Univariate#1\ByGroup2#1\ACTUAL#1\ExtremeObs#1 TO
Germany#1;
   copy \Work.Prddoc\Tabulate#1\ByGroup2#1\Report#1\Table#1 TO
Germany#1;
   setlabel Germany#1\Table#1 'Predicted Sales';
```

Create a new folder and copy output objects into it. The MAKE statement creates the folder "Canada". The SETLABEL statement specifies the text of the label. The COPY TO statements copy the output objects ExtremeObs#1 and Table#1 to the new folder.

```
   make
Canada;

   setlabel Canada
'Canada';
   copy \Work.Prddoc\Univariate#1\ByGroup1#1\ACTUAL#1\ExtremeObs#1 TO
Canada#1;
   copy \Work.Prddoc\Tabulate#1\ByGroup1#1\Report#1\Table#1 TO
Canada#1;
   setlabel Canada#1\Table#1 'Predicted
Sales';
```

End the DOCUMENT procedure.

```
quit;
```

Specify the PDF destination and replay the document. The ODS PDF statement opens the PDF destination and creates the output file. The REPLAY statement in the PROC DOCUMENT step replays the document Newprd to the PDF destination.

```
ods pdf file='file-path/CustomPrd.pdf' contents=yes;

proc document name=work.Newprd;
   replay;
run;
```

```
quit;

ods _all_ close;
```

PDF Output

Output 6.20 *Original PDF Table of Contents*

Table of Contents

The Tabulate Procedure . . . 1
Country=CANADA . . . 1
Cross-tabular summary report . . . 1
Table 1 . . . 1
Country=GERMANY . . . 2
Cross-tabular summary report . . . 2
Table 1 . . . 2
Country=U.S.A. . . . 3
Cross-tabular summary report . . . 3
Table 1 . . . 3
The Univariate Procedure . . . 4
Country=CANADA . . . 4
ACTUAL . . . 4
Extreme Observations . . . 4
Country=GERMANY . . . 5
ACTUAL . . . 5
Extreme Observations . . . 5
Country=U.S.A. . . . 6
ACTUAL . . . 6
Extreme Observations . . . 6

Output 6.21 *PDF Table of Contents After the Output Has Been Modified*

Table of Contents

United States . . . 1
Extreme Observations . . . 1
Predicted Sales . . . 2
Germany . . . 3
Extreme Observations . . . 3
Predicted Sales . . . 4
Canada . . . 5
Extreme Observations . . . 5
Predicted Sales . . . 6

Example 8: Using PROC DOCUMENT to Work with Table Templates

Features:

PROC DOCUMENT statements
- DIR
- LIST statement
- MAKE
- OBTEMPL
- PROC DOCUMENT
- REPLAY
- SETLABEL

ODS DOCUMENT statement

PROC ODSTABLE

PROC SQL

PROC TEMPLATE statements
- DELETE
- CELLSTYLE AS

Details

In this example, the PROC SQL steps create two tables. The following output shows the default output for the PROC SQL steps:

Output 6.22 *Default PROC SQL Output*

Cars with City MPG Greater Than 40

Make	Model	MPG (City)	MPG (Highway)
Honda	Civic Hybrid 4dr manual (gas/electric)	46	51
Honda	Insight 2dr (gas/electric)	60	66
Honda	Civic HX 2dr	36	44
Toyota	Prius 4dr (gas/electric)	59	51
Toyota	Echo 2dr manual	35	43
Toyota	Echo 4dr	35	43
Volkswagen	Jetta GLS TDI 4dr	38	46

Output 6.23 *Default PROC SQL Output*

Cars with City MPG Greater Than 25

Make	Model	MPG (City)	MPG (Highway)
Chevrolet	Aveo 4dr	28	34
Chevrolet	Aveo LS 4dr hatch	28	34
Chevrolet	Cavalier 2dr	26	37
Chevrolet	Cavalier 4dr	26	37
Chevrolet	Cavalier LS 2dr	26	37
Dodge	Neon SE 4dr	29	36
Dodge	Neon SXT 4dr	29	36
Ford	Focus ZX3 2dr hatch	26	33
Ford	Focus LX 4dr	27	36
Ford	Focus SE 4dr	26	33
Ford	Focus ZX5 5dr	26	33
Ford	Focus ZTW	26	33
Honda	Civic Hybrid 4dr manual (gas/electric)	46	51
Honda	Insight 2dr (gas/electric)	60	66
Honda	Civic DX 2dr	32	38
Honda	Civic HX 2dr	36	
Honda			
			34
	Accord EX 2dr	26	34

Suppose that you want to modify the look of these tables. You can use PROC ODSTABLE to create customized table templates. You can then use PROC DOCUMENT to apply the customized table templates to your output objects. You can even create two different table templates with PROC ODSTABLE and apply each one to a different output object with PROC DOCUMENT.

Program: Create the ODS Document and List the Output Objects

```
ods document name=SqlTempl(write);
proc sql;
title "Cars with City MPG Greater Than 25";
   select Make, Model, MPG_City, MPG_Highway
      from sashelp.cars
      where MPG_City >25;
run;
quit;
proc sql;
title "Cars with City MPG Greater Than 40";
```

```
   select Make, Model, MPG_City, MPG_Highway
      from sashelp.cars
      where MPG_Highway >40;
run;
quit;
ods document close;

title;
proc document name=SqlTempl;
   list / levels=all;
run;
quit;
```

Program Description

In order to create a customized version of the output object's table template with PROC DOCUMENT, you need to know the table template name. It is also helpful to see the SAS code that creates the default table template, so that you can customize the attributes.

Create the ODS document and the procedure output. The ODS DOCUMENT statement creates the document named SqlTempl. The SqlTempl document contains references to the PROC SQL output objects that the program creates.

```
ods document name=SqlTempl(write);
proc sql;
title "Cars with City MPG Greater Than 25";
   select Make, Model, MPG_City, MPG_Highway
      from sashelp.cars
      where MPG_City >25;
run;
quit;
proc sql;
title "Cars with City MPG Greater Than 40";
   select Make, Model, MPG_City, MPG_Highway
      from sashelp.cars
      where MPG_Highway >40;
run;
quit;
ods document close;
```

List the output objects. The PROC DOCUMENT procedure step that includes the LIST statement prints a listing of the paths for all of the output objects.

```
title;
proc document name=SqlTempl;
   list / levels=all;
run;
quit;
```

Output 6.24 *Listing of the SqlTempl Document.*

Listing of: \Work.Sqltempl\		
Order by: Insertion		
Number of levels: All		
Obs	Path	Type
1	\SQL#1	Dir
2	\SQL#1\SQL_Results#1	Table
3	\SQL#2	Dir
4	\SQL#2\SQL_Results#1	Table

Program: View the Table Templates

View the table template associated with the two tables. By using the preceding LIST statement, we know that the paths to the output objects that are tables are `\SQL#1\SQL_Results#1` and `\SQL#2\SQL_Results#1`. The OBTEMPL statement specifies that PROC DOCUMENT prints the contents of the output object's table template to the open destination, which is the HTML5 destination in this example.

```
proc document name=SqlTempl;
   obtempl \SQL#1\SQL_Results#1;
   obtempl \SQL#2\SQL_Results#1;
run;
quit;
```

The name of the table template that is used for the PROC SQL output is Base.SQL. Both output objects use this table template because it is the default table template for all PROC SQL output. The entire code for the table template is written to the Results window.

Output 6.25 *SQL Table Template Program Code*

```
proc template;
   define table Base.SQL;
      notes "PROC SQL Output";
      dynamic wmax flowmin flowmax ds;
      column obsno character flowcharacter numeric;

      define obsno;

         define header t;
            text _label_;
            def_split;
         end;
         header = t;
         space = 2;
         format = 6.0;
         just = r;
         style = RowHeader;
         id;
         generic;
      end;
```

Program: Create Two Custom Table Templates

```
proc odstable name=Base.SQL store=work.mystore;
   cellstyle
         _DATANAME_ = "MPG_City" and  _val_ gt (40) as
{background=orange},
          mod(_row_,2) as {Background=Honeydew};
run;

proc odstable name=Base.SQL;
   cellstyle _DATANAME_ = "MPG_Highway" and _val_ gt (50) as
{background=orange},
          mod(_row_,2) as {Background=Honeydew};
run;
```

Program Description

The following program creates two table templates, both named Base.SQL. The name Base.SQL is used because that is the table template name that PROC DOCUMENT is looking for. However, one instance of Base.SQL is stored in the default template store Sasuser.Templat, and the other is stored in the template store Work.Mystore. You can use the STORE= option in the REPLAY statement to replay two output objects and use a different table template for each one, even though the template name is the same.

Create a custom table template in a new item store. PROC ODSTABLE creates a custom table template. You must give the new template the same name as the

default table template, Base.SQL. However, this template is now applied to every table created by SAS that uses the table name Base.SQL, unless it is overridden by another template created by PROC TEMPLATE or PROC ODSTABLE, removed with the DELETE statement, or manually removed from the item store. The STORE= option creates the item store Work.MyStore that contains the new table template.

```
proc odstable name=Base.SQL store=work.mystore;
   cellstyle
         _DATANAME_ = "MPG_City" and  _val_ gt (40) as
{background=orange},
          mod(_row_,2) as {Background=Honeydew};
run;
```

Create a second, different, custom table template in the Sasuser item store. Use PROC ODSTABLE to create a custom table template with the same name as the first table template. You must give the new template the same name as the default table template, Base.SQL. This template is applied to every table created by SAS that uses the table name Base.SQL, unless it is overridden by another template created by PROC TEMPLATE or PROC ODSTABLE, removed with the DELETE statement, or manually removed from the item store. However, because the STORE= option is not specified in the PROC ODSTABLE statement, the template is stored in the default location, which is Sasuser.Templat. Because these two new templates are stored in different locations, you can apply a different template to each output object.

```
proc odstable name=Base.SQL;
   cellstyle _DATANAME_ = "MPG_Highway" and _val_ gt (50) as
{background=orange},
          mod(_row_,2) as {Background=Honeydew};
run;
```

Program: Replay the Output Objects

Replay the output objects. The REPLAY statement replays the output objects SQL_Results#1 and SQL_Results#2. The STORE= option specifies that the procedure use the template from the template store Mystore. If the STORE= option is not specified, then the Sasuser.Templat item store is used by default.

```
proc document name=sqlTempl;
   replay \SQL#2\SQL_Results#1, \SQL#1\SQL_Results#1(store=mystore);
run;
quit;
```

Output 6.26 *Output Using the Table Template from Work.Mystore*

Cars with City MPG Greater Than 25

Make	Model	MPG (City)	MPG (Highway)
Chevrolet	Aveo 4dr	28.000000	34.000000
Chevrolet	Aveo LS 4dr hatch	28.000000	34.000000
Chevrolet	Cavalier 2dr	26.000000	37.000000
Chevrolet	Cavalier 4dr	26.000000	37.000000
Chevrolet	Cavalier LS 2dr	26.000000	37.000000
Dodge	Neon SE 4dr	29.000000	36.000000
Dodge	Neon SXT 4dr	29.000000	36.000000
Ford	Focus ZX3 2dr hatch	26.000000	33.000000
Ford	Focus LX 4dr	27.000000	36.000000
Ford	Focus SE 4dr	26.000000	33.000000
Ford	Focus ZX5 5dr	26.000000	33.000000
Ford	Focus ZTW	26.000000	33.000000
Honda	Civic Hybrid 4dr manual (gas/electric)	46.000000	51.000000
Honda	Insight 2dr (gas/electric)	60.000000	66.000000
Honda	Civic DX 2dr	32.000000	38.000000
Honda	Civic HX 2dr	36.000000	
Honda			
	Accord EX 2dr	26.000000	34.000000

Output 6.27 *Output Using the Table Template from Sasuser.Templat*

Cars with City MPG Greater Than 40

Make	Model	MPG (City)	MPG (Highway)
Honda	Civic Hybrid 4dr manual (gas/electric)	46.000000	51.000000
Honda	Insight 2dr (gas/electric)	60.000000	66.000000
Honda	Civic HX 2dr	36.000000	44.000000
Toyota	Prius 4dr (gas/electric)	59.000000	51.000000
Toyota	Echo 2dr manual	35.000000	43.000000
Toyota	Echo 4dr	35.000000	43.000000
Volkswagen	Jetta GLS TDI 4dr	38.000000	46.000000

Delete the table templates. The DELETE statements delete the template Base.SQL from the Sasuser.Templat template store and the template Base.SQL from the Work.Mystore template store. Because Base.SQL is the default table template for all PROC SQL output, if you do not delete the custom templates, all of your PROC SQL output will have the customized template applied to it.

```
proc template;
   delete Base.SQL;
```

```
   delete Base.SQL / store=mystore;
run;
```

PART 4

The ODSLIST Procedure

Chapter 7
ODSLIST Procedure **199**

Chapter 7

ODSLIST Procedure

Overview: ODSLIST Procedure ***199***
What Does the ODSLIST Procedure Do? 200

Syntax: ODSLIST Procedure ***200***
PROC ODSLIST Statement 201
BY Statement 202
END Statement 206
CELLSTYLE AS Statement 206
DYNAMIC Statement 209
ITEM Statement 210
LIST Statement 212
MVAR Statement 214
NMVAR Statement 215
P Statement 216
TRANSLATE INTO Statement 218

Usage: ODSLIST Procedure ***220***
Using the ODSLIST Procedure 220
Changing the Bullet Type for Lists 221

Examples: ODSLIST Procedure ***223***
Example 1: Using Item Statements 223
Example 2: Creating Nested Lists 226
Example 3: Customizing Lists 228
Example 4: Comparing Lists Created with PROC ODSLIST and PROC ODSTEXT 233

Overview: ODSLIST Procedure

What Does the ODSLIST Procedure Do?

The ODSLIST procedure is used to create bulleted lists. With PROC ODSLIST, you can create text templates for lists that can be customized and nested an infinite number of times. You can use style attributes and formats to customize your content and WHERE expressions to specify list item content. With PROC ODSLIST, you can use the DATA= option to bind your data to a template without using a DATA step.

PROC ODSLIST can be used with any output destination. However, PROC ODSLIST is essential for creating content for the ODS destination for PowerPoint and e-books. See ODS EPUB Statement and "ODS POWERPOINT Statement" in *SAS Output Delivery System: User's Guide* for more information about creating output for the ODS destination for PowerPoint and e-book output.

Syntax: ODSLIST Procedure

PROC ODSLIST <CONTENTS=*text-string*><DATA=*SAS-data-set*>
<NAME=*template-name*>
<PAGEBREAK=NO | YES> <PRINT> <STORE=*template-store*>;
BY <DESCENDING> *variable-1*
<… <DESCENDING> *variable-n*>
<NOTSORTED>;
CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*] ><, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*]>>;
DYNAMIC *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;
MVAR *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;
NMVAR *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;
ITEM <*expression*> / <FORMAT=*format-name*> <STYLE=*style-override*>
<VALUE=*integer-value*>;
P *expression* / <FORMAT=*format-name*> <STYLE=*style-override*>;
LIST / <START=*integer-value*><STYLE=*style-override*> ;

```
    ITEM <expression> / <FORMAT=format-name> <STYLE=style-override>
      <VALUE=integer-value>;
  END;
END;
TRANSLATE expression-1 INTO expression-2 < , expression-n INTO expression-
m;>
```

PROC ODSLIST Statement

Creates an item list.

Syntax

PROC ODSLIST <CONTENTS=*text-string*><DATA=*SAS-data-set*>
<NAME=*template-name*>
<PAGEBREAK=NO | YES><PRINT> <STORE=*template-store*>;

Optional Arguments

CONTENTS="*text-string*"
specifies the title for the table of contents. The CONTENTS= option overrides the generated table of contents text.

Tip *text-string* is the text that can be seen in the PDF table of contents.

DATA=*SAS-data-set*
specifies a SAS data set.

Tip No output is produced if you specify the NAME= option without either the DATA= option or the PRINT option. Without the DATA= or PRINT options, ODS creates the template but does not render the output.

NAME=*template-name*
specifies that the content of the procedure should be saved as a template with the specified name.

Tip No output is produced if you specify the NAME= option without either the DATA= option or the PRINT option. Without the DATA= or PRINT options, ODS creates the template but does not render the output.

PAGEBREAK=NO | YES
specifies whether the procedure should generate a page break.

NO
specifies that no page break is generated.

Alias OFF

YES
specifies that a page break is generated.

Alias ON

Default NO

PRINT
specifies that the template is printed as well as stored.

Restriction The PRINT option is needed only if the NAME= option is specified without the DATA= option.

Tip No output is produced if you specify the NAME= option without either the DATA= option or the PRINT option. Without the DATA= or PRINT options, ODS creates the template but does not render the output.

STORE=*template-store*
specifies the template store where the compiled template is placed. If you do not specify a name or template store, ODS stores the template in the first writable template store in the ODS path.

If you do specify a name and template store, but do not have a libref specified, then the template is stored in the Work directory. For example, the statement `proc odslist name=mylist store=mystore;` results in a template store named Work.mystore.

Restriction The STORE= option can only be specified if the NAME= option is specified.

BY Statement

Orders the output according to the BY groups.

Note: BY-Group processing is included in release 2020.1.2 and later.

Syntax

BY <DESCENDING> *variable-1*
<… <DESCENDING> *variable-n*>
<NOTSORTED>;

Required Argument

variable
specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING
specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED
specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see *SAS DATA Step Statements: Reference*.
- For more information about PROC SORT, see “SORT Procedure” in *Base SAS Procedures Guide*.
- For more information about BY-group processing in CAS, see .“Grouping and Ordering” in *SAS Cloud Analytic Services: DATA Step Programming*

- For more information about creating indexes, see "INDEX CREATE Statement" in *Base SAS Procedures Guide*.

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

1. The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
2. The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
3. The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted

lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, ‘ABCD’ in the primary data set matches ‘ABCD ’ in the secondary data set. If the secondary data set contained ‘ABCDEF’, then they would not match.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT
ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

"BY Statement" in *SAS DATA Step Statements: Reference*

END Statement

Ends item and list blocks.

Syntax

END;

CELLSTYLE AS Statement

For tables, sets the style element of the cells in the table or column according to the values of the variables. For text, sets the style attributes of the list items or paragraphs. Use this statement to set the presentation characteristics (such as foreground color and font face) of individual cells or text.

Restriction: The CELLSTYLE AS statement can be used only within a column template, an ODS list, an ODS textblock, or a table template.

Syntax

CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*]> <, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*] >>;

Required Arguments

expression

is an expression that is evaluated for each list item, paragraph, or table cell.

If *expression* resolves to TRUE (a nonzero value), the style element that is specified is used for the current cell. If *expression* is FALSE, the next expression in the statement is evaluated. Thus, you can string multiple expressions together to format cells conditionally.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

An operator is a symbol that requests a string, a comparison, logical operation, or arithmetic calculation. An operand is one of the following:

constant
is a fixed value such as the name of a column or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in-variable
is a special type of WHERE expression operand that helps you find common values in table or column templates.

Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

DATANAME
is a data column name.

DATATYPE
is the data type of the column variable. The data type is either numeric ("num") or character ("char").

Example The following CELLSTYLE AS statement specifies that numeric column variables have a red font color and character column variables have a blue font color:

```
cellstyle   _datatype_ = "num" as {color=red},
            _datatype_ = "char" as {color=blue};
```

LABEL
is a column label.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

ROW
is a row number. Row numbering begins with 1.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

STYLE
is a style element name.

See For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

Example “Example 6: Creating Top-Level Templates” on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example “Example 6: Creating Top-Level Templates” on page 577

comparison-operator
compares a variable with a value or with another variable.

The following table lists the comparison operators:

Table 7.1 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one or more from a list of values

Tip Using an expression of 1 as the last expression in the CELLSTYLE AS statement sets the style element for any cells that did not meet an earlier condition. For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer’s Guide: Essentials*.

Example “Example 5: Setting the Style Element for a Specific Column, Row, and Cell” on page 570

style-attribute-specification
describes a style attribute to set.

Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

For information about the style attributes that you can set in a table template, see “About Style Attributes” on page 465.

Optional Argument

style-element-name
is the name of a style element that is part of a style that is registered with the Output Delivery System.

SAS provides some styles. You can create customized styles and style elements with PROC TEMPLATE by using the “DEFINE STYLE Statement” on page 454. For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

The following style elements are frequently used with the CELLSTYLE AS statement:

- Data
- DataFixed
- DataEmpty
- DataEmphasis
- DataEmphasisFixed
- DataStrong
- DataStrongFixed
- ListItem
- ListItem2
- Paragraph

The style element provides the basis for displaying the cell. Additional style attributes modify the display.

Default Data

See Chapter 14, “TEMPLATE Procedure,” on page 433

For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

DYNAMIC Statement

Defines a symbol that references a value that the data component supplies from the procedure or DATA step.

Restriction: The DYNAMIC statement can be used only in the template of an ODS textblock, ODS list, table, column, header, footer, or statistical graph template. A dynamic variable that is defined in a template is available to that template and to all the templates that it contains.

Syntax

DYNAMIC *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;

Required Argument

variable-name
names a variable that the data component supplies. ODS resolves the value of the variable when it binds the template and the data component.

Tip Dynamic variables are most useful to the authors of SAS procedures and to DATA step programmers.

Optional Arguments

value
sets the value of the variable.

text
is text that is placed in the template to explain the dynamic variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

ITEM Statement

Specifies an item to be added to the item list. An ITEM statement that does not specify an expression begins an item block.

Restriction: ITEM statements that do not specify an *expression* must end with an END statement.

Example: "Example 4: Comparing Lists Created with PROC ODSLIST and PROC ODSTEXT" on page 233

Syntax

ITEM <*expression*> / <FORMAT=*format-name*> <STYLE=*style-override*> <VALUE=*integer-value*>;

Optional Argument

expression
is an expression that specifies the content of an item.

expression has this form:

expression-1 < *expression-n*>

expression
is an arithmetic or logical sequence of operators and operands. An operator is a symbol that requests a logical operation, a string, or an arithmetic calculation.

An operand is one of the following:

constant
is a fixed value, such as the name of a column, a text string, or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

Restriction ITEM statements that do not specify an *expression* must end with an END statement.

Options to Use after a Slash Delimiter

FORMAT=*format-name*
specifies a default format for the value in each table cell or list item. You can use any SAS or user-defined format.

Example
```
proc odslist data=sashelp.class;
   item age / format=2.;
run;
```

STYLE=<*style-element-name* >[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]
specifies the style element to use for the specified item.

style-element-name
is the name of a style element that is part of a style template that is registered with the Output Delivery System. SAS provides some style templates. You can create your own style templates with PROC TEMPLATE.

See Chapter 14, “TEMPLATE Procedure,” on page 433 for information about PROC TEMPLATE and the default style templates.

For a list of style elements, see Chapter 20, “Style Elements,” on page 777.

style-attribute-name
specifies the attribute to change.

See For information about style attributes and their values, see Chapter 21, “Style Attributes,” on page 807.

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values.

See For information about style attributes and their values, see Chapter 21, “Style Attributes,” on page 807.

Note You can use braces ({ and }) instead of square brackets ([and]).

Example The following statement specifies that the text color for the item is red:

```
item 'first item' / style=[color=red];
```

VALUE=*integer-value*

starts a numbered list from the specified *integer-value*. If another VALUE= option is specified in subsequent statements, numbering begins again at that point.

Restriction When VALUE= is specified, you must also specify a numbering list type with the LISTSTYLETPYE= style attribute. For example, LISTSTYLETYPE= “decimal_leading_zero” or LISTSTYLETYPE=“decimal”.

Tip You can change the bullet type of a list item by specifying the “LISTSTYLETYPE=bullet-type” style attribute with the STYLE= option.

Examples The following example creates a list with three items that are numbered 7, 10, and 4.

```
proc odslist;
   item;
     p 'This is a numbered list';
     list / style={liststyletype="decimal_leading_zero"};
       item "Seven" / value=7;
       item "Ten" / value=10;
       item "Four" / value=4;
     end;
   end;
run;
```

The following example creates a list that begins numbering items with the number 7.

```
proc odslist;
   item;
     p 'This is a numbered list';
     list / style={liststyletype="decimal_leading_zero"};
       item "Seven" / value=7;
       item "Eight";
     end;
   end;
run;
```

Example “Example 2: Creating Nested Lists” on page 226

LIST Statement

Creates a list.

Restrictions: The LIST statement must be used within an item block.

LIST statement begins with a LIST statement block, and must end with an END statement.

Example: “Example 4: Comparing Lists Created with PROC ODSLIST and PROC ODSTEXT” on page 233

Syntax

LIST / <START=*integer-value*><STYLE=*style-override*> ;

Options to Use after a Slash Delimiter

START=*integer-value*
starts a numbered list from the specified *integer-value*. If another START= option is specified in subsequent statements, the numbering begins again at that point.

Restrictions When START= is specified, you must also specify a numbering list type with the LISTSTYLETPYE= style attribute. For example, LISTSTYLETYPE= “decimal_leading_zero” or LISTSTYLETYPE=“decimal” are numbering list types.

The START= option is valid for the PDF, POWERPOINT, HTML, and RTF destinations.

Tip You can change the bullet type of a list item by specifying the “LISTSTYLETYPE=bullet-type” style attribute with the STYLE= option.

Examples The following example creates a list that begins numbering items with the number seven. For HTML5 output, if you want to start a numbered list from a specific number with the START= option, then specify LISTSTYLETYPE= on the LIST statement, as this example does.

```
proc odslist;
   item;
     p 'This is a numbered list';
     list / style={liststyletype="decimal_leading_zero"} start=7;
      item "Seven";
      item "Eight";
     end;
   end;
run;
```

The following example creates a list that assigns a specific number to each item.

```
proc odslist;
   item;
     p 'This is a numbered list';
     list / style={liststyletype="decimal_leading_zero"};
      item "Two" / value=2;
      item "Four" / value=4;
      item "Six" / value=6
     end;
   end;
run;
```

STYLE=<*style-element-name*> [*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]
specifies the style element to use for the specified item in the list.

style-element-name
is the name of a style element that is part of a style template that is registered with the Output Delivery System. SAS provides some style templates. You can create your own style templates with PROC TEMPLATE.

See Chapter 14, “TEMPLATE Procedure,” on page 433 for information about creating style templates with PROC TEMPLATE.

For a list of style elements, see Chapter 20, “Style Elements,” on page 777.

style-attribute-name
specifies the attribute to change.

See For information about style attributes and their values, see Chapter 21, “Style Attributes,” on page 807.

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values.

See For information about style attributes and their values, see Chapter 21, “Style Attributes,” on page 807.

Note You can use braces ({ and }) instead of square brackets ([and]).

Example The following statement specifies that the text color for the item is red:

```
item 'first item' / style=[color=red];
```

MVAR Statement

Defines a symbol that references a macro variable. ODS uses the value of the variable as a string. References to the macro variable are resolved when ODS binds the template and the data component to produce an output object.

Restriction: When replaying an ODS document with PROC DOCUMENT, values created by the MVAR statement must be re-created in the same session that is replaying the document.

Tip: You can use the MVAR statement in the template of an ODS list, ODS textblock, table, column, header, or footer. A macro variable that is defined in a template is available to that template and to all the templates that it contains.

See: “Example 3: Creating a New Table Template ” on page 557 and “Example 1: Creating a Stand-Alone Style” on page 466

Syntax

MVAR *variable-name-1* <='*value-1*' <'*text-1*'>>

< *variable-name-2* <='*value-2*'> <'*text-2*'…>>;

Required Argument

variable-name
names a macro variable to reference in the template. ODS uses the value of the macro variable as a string. ODS does not resolve the value of the macro variable until it binds the template and the data component.

Tip Declare macro variables this way in a template. For example, to use the automatic macro variable SYSDATE9 in a template, declare it in an MVAR statement and reference it as SYSDATE9, without an ampersand, in the PROC TEMPLATE or PROC ODSTABLE step. If you use the ampersand, the macro variable resolves when the template is compiled instead of when ODS binds the template to the data component.

Optional Arguments

value
sets the default variable value.

text
is text that is placed in the template to explain the macro variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

NMVAR Statement

Defines a symbol that references a macro variable. ODS converts the variable's value to a number (stored as a double) before using it. References to the macro variable are resolved when ODS binds the template and the data component to produce an output object.

Restriction: The NMVAR statement can be used only in the template of an ODS list, ODS textblock, table, column, header, or footer. A macro variable that is defined in a template is available to that template and to all the templates that it contains.

See: "Example 4: Setting the Style Element for Cells Based on Their Values" on page 565

Syntax

NMVAR *variable-name-1* <='*value–1*' <'*text-1*'>>
< *variable-name-2* <='*value-2*'> <'*text-2*'…>>;

Required Argument

variable-name
names a macro variable to reference in the template. ODS converts the variable's value to a number (stored as a double) before using it. ODS does not resolve the macro variable until it binds the template and the data component.

Tip Declare macro variables this way in a template. For example, to use a macro variable as a number, declare it in an NMVAR statement and reference it without an ampersand. If you use the ampersand, the macro variable resolves when the template is compiled instead of when ODS binds the template to the data component.

Optional Arguments

value
sets the value of the variable.

text
is text that is placed in the template to explain the macro variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

P Statement

Specifies a paragraph. Multiple P statements are allowed within an item block.

Restriction: In the ODSLIST Procedure, the P statement can be specified only within an ITEM block. In the ODSTEXT procedure, the P statement can be used at the top level of PROC ODSTEXT as well as within list items.

Syntax

P *expression* / <FORMAT=*format-name*> <STYLE=*style-override*> ;

Required Argument

expression
is an expression that specifies the content of an item or paragraph.

expression has this form:

expression-1 < *expression-n*>

expression
is an arithmetic or logical sequence of operators and operands. An operator is a symbol that requests a logical operation or an arithmetic calculation.

An operand is one of the following:

constant
is a fixed value, such as the name of a column, text string, or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

Examples The following code creates a paragraph and an item list with PROC ODSTEXT:

The following code creates an item list with PROC ODSLIST:

```
proc odslist data=sashelp.class;
   item;
     p "My name is " || name;
     list;
       item;
         p "My age is " || put(age, 2.);
         p "My weight is " || put(weight, 3.);
       end;
     end;
   end;
run;
```

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

Restriction ITEM statements that do not specify an *expression* must end with an END statement.

Options to Use after a Slash Delimiter

FORMAT=*format-name*
specifies a default format for the value in each paragraph. You can use any SAS or user-defined format.

STYLE=<*style-element-name* >[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]
specifies the style element to use for the items in the list or paragraph.

style-element-name
is the name of a style element that is part of a style template that is registered with the Output Delivery System. SAS provides some style templates. You can create your own style templates with PROC TEMPLATE.

See Chapter 14, “TEMPLATE Procedure,” on page 433 for information about PROC TEMPLATE and the default style templates.

For a list of style elements, see Chapter 20, “Style Elements,” on page 777.

style-attribute-name
specifies the attribute to change.

See For information about style attributes and their values, see Chapter 21, “Style Attributes,” on page 807.

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values.

See For information about style attributes and their values, see Chapter 21, "Style Attributes," on page 807.

Note You can use braces ({ and }) instead of square brackets ([and]).

Examples The following statement specifies that the text color of the paragraph is red:

```
p 'text block' / style=[color=red];
```

The following statement specifies that the text color for the item is red:

```
item 'first item' / style=[color=red];
```

TRANSLATE INTO Statement

Translates the specified numeric values to other values.

Restrictions: The TRANSLATE INTO statement can be used only in a column template, an ODS list, an ODS textblock, or a table template.

The TRANSLATE INTO statement in a table template applies only to numeric variables. To translate the values of a character variable, use TRANSLATE INTO in the template of that column.

Syntax

TRANSLATE *expression-1* **INTO** *expression-2* <, *expression-n* INTO *expression-m*>;

Required Argument

expression
is an expression that is evaluated for each list item, paragraph, table, or column cell that contains a numeric variable.

If *expression-1* resolves to TRUE (a nonzero value), the translation that is specified is used for the current cell. If *expression-1* is FALSE (zero), the next expression in the statement is evaluated. Thus, you can string multiple expressions together to format cells conditionally.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation.

An operand is one of the following:

constant
is a fixed value such as the name of a column or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in-variable
is a special type of WHERE expression operand that helps you find common values in table or column templates.

Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

DATANAME
is a data column name.

DATATYPE
is the data type of the column variable. The data type is either numeric ("num") or character ("char").

LABEL
is a column label.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

ROW
is a row number. Row numbering begins with 1.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

STYLE
is a style element name.

Example "Example 6: Creating Top-Level Templates" on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example "Example 6: Creating Top-Level Templates" on page 577

comparison-operator
compares a variable with a value or with another variable.

The following table lists the comparison operators:

Table 7.2 Comparison Operators

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one or more from a list of values

Restriction You cannot reference the values of other columns in *expression-1*.

Tip Using an expression of 1 as the last expression in the TRANSLATE–INTO statement specifies a translation for any cells that did not meet an earlier condition.

See You can use any expression that can be used in the WHERE= data set option. For information about expressions that you can use in the WHERE data set option, see the section on WHERE-Expression Processing in *SAS Programmer's Guide: Essentials*.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

Usage: ODSLIST Procedure

Using the ODSLIST Procedure

PROC ODSLIST uses many of the same statements as PROC TEMPLATE to create and customize your list templates. The CELLSTYLE AS, DYNAMIC, MVAR, NMVAR, and TRANSLATE INTO statements are all valid statements within PROC ODSLIST that can be used to customize your list templates.

Changing the Bullet Type for Lists

By default, lists created by the ODSLIST procedure are displayed as an unordered list with a disc as the bullet type. In markup family output, you can change the bullet type for lists by using the following two steps:

1 Specify the LISTSTYLETYPE= style attribute with the desired bullet type in a style override. The following shows the general syntax for specifying the LISTSTYLETYPE= attribute::

   STYLE={ LISTSTYLETYPE="*bullet-type*" <*more-style-attributes*>}

2 Specify the style override in an ITEM or LIST statement. The following shows the general syntax for specifying a style override in the LIST or ITEM statements:

   LIST / STYLE={ LISTSTYLETYPE="*bullet-type*" <*more-style-attributes*>}

   ITEM / STYLE={ LISTSTYLETYPE="*bullet-type*" <*more-style-attributes*>}

Note:

For HTML5 output, if you want to start a numbered list from a specific number with the START= option, then specify LISTSTYLETYPE= in the LIST statement.

IMPORTANT

The LISTSTYLETYPE= style attribute is valid in the Markup family, Excel, and PowerPoint destinations. The valid bullet-type values differ for each destination. For more information about valid bullet types, see the LISTSTYLETYPE= Style Attribute on page 861.

When you specify LISTSTYLETYPE= in the LIST statement, the bullets for all of the following items are changed. If you specify LISTSTYLETYPE= in an ITEM statement, only the bullet for that ITEM statement is changed.

The following example creates a numbered list. Because the style override is specified in the LIST statement, all of the following items have the same numbering scheme.

```
title;
proc odslist;
   item / style={liststyletype="none" fontsize=12pt};
   p 'Follow these steps to change the bullet type for lists:'
                                    / style={fontsize=12pt};
   list / style={liststyletype="decimal" fontsize=12pt};
      item "Decide what bullet type to display.";
      item "Add a '/' after the LIST statement (if not already
present).";
      item "Specify the LISTSTYLETYPE= attribute with the STYLE=
option.";
   end;
end;
run;
```

Output 7.1 *Changing the Bullet Type for All Items*

Follow these steps to change the bullet type for lists:

1. Decide what bullet type to display.
2. Add a '/' after the LIST statement (if not already present).
3. Specify the LISTSTYLETYPE= attribute with the STYLE= option.

You can also change the bullet type for each individual item by specifying a style override in each ITEM statement.

```
title 'You Can Change the Bullet Type for Each Individual Item' ;

proc odslist;
   item "List items can be a square" / style={liststyletype="square"
fontsize=12pt};
   item "Or a circle" / style={liststyletype="circle" fontsize=12pt};
   item "Or a disc" / style={liststyletype="disc" fontsize=12pt};
   item "Or items can have no bullet" / style={liststyletype="none"
fontsize=12pt};
end;
run;
```

Output 7.2 *Changing the Bullet Type for Individual Items*

You Can Change the Bullet Type for Each Individual Item

- List items can be a square
  - Or a circle
    - Or a disc

      Or items can have no bullet

The following are browser-supported bullet types. However, many browsers support other common types and the support is browser dependent. If an unsupported bullet type is specified, the bullet's display is browser dependent.

Table 7.3 *Valid Bullet Types*

Bullet Type	Description
	Ordered Lists
decimal	The list items are ordered with numbers (default).
upper_latin	The list items are ordered with uppercase letters.
lower_latin	The list items are ordered with lowercase letters.
upper_roman	The list items are ordered with uppercase roman numerals.
lower_roman	The list items are ordered with lowercase roman numerals.
	Unordered Lists

Bullet Type	Description
disc	The list items are marked with filled circles.
circle	The list items are marked with circles.
square	The list items are marked with squares.
none	The list items are not marked.

Examples: ODSLIST Procedure

Example 1: Using Item Statements

Features:

CELLSTYLE AS statement

PROC ODSLIST statement
- NAME= option
- PRINT option
- STORE= option

ITEM statement
- STYLE= option

ODS POWERPOINT statement

OPTIONS statement

TITLE statement

Details

The following program creates a list for a Microsoft PowerPoint slide. The CELLSTYLE AS statement formats the contents of each item.

Program

```
ods html close;
options nodate;
title 'Using PROC ODSLIST ITEM Statements';
```

```
ods powerpoint file="file-path/DefaultStyle.ppt";
ods powerpoint(2) file="file-path/PowerpointdarkStyle.ppt"
style=powerpointdark;


proc odslist name=Slides store=work.Myexampleslides print;
   cellstyle 1 as {fontsize=1cm color=purple fontweight=bold};
   item 'Fraud';
   item 'Customer Intelligence';
   item 'Social Media';
   item 'Data Mining';
   item 'High-Performance Computing';
   item 'Risk';
   item 'Data Management';
run;


ods _all_ close;
```

Program Description

Close the HTML destination and specify the SAS system options. The HTML destination is open by default in SAS Windowing environment. If you are not creating HTML output, then close the destination to save system resources.

```
ods html close;
options nodate;
title 'Using PROC ODSLIST ITEM Statements';
```

Open two instances of the ODS destination for PowerPoint. The ODS POWERPOINT statement creates output formatted for the ODS destination for PowerPoint.

```
ods powerpoint file="file-path/DefaultStyle.ppt";
ods powerpoint(2) file="file-path/PowerpointdarkStyle.ppt"
style=powerpointdark;
```

Begin the ODSLIST procedure and create a template store and name for the template. The NAME= option gives the template a name and the STORE= option specifies the template store where the template is stored. If the specified template store does not exist, then ODS creates it. Because there is no DATA= option specified, you must use the PRINT option to render the output. Otherwise, the template is created but no output is rendered.

```
proc odslist name=Slides store=work.Myexampleslides print;
```

Create content for the ODS destination for PowerPoint. The ITEM statements create each item in the list. The CELLSTYLE AS statement applies the style attributes to each item.

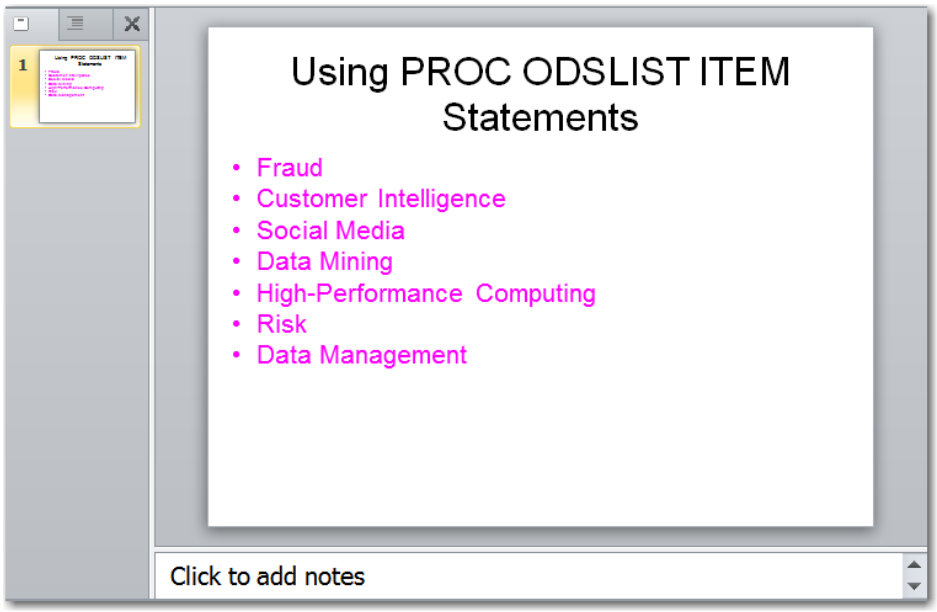
```
   cellstyle 1 as {fontsize=1cm color=purple fontweight=bold};
   item 'Fraud';
   item 'Customer Intelligence';
   item 'Social Media';
```

```
   item 'Data Mining';
   item 'High-Performance Computing';
   item 'Risk';
   item 'Data Management';
run;
```

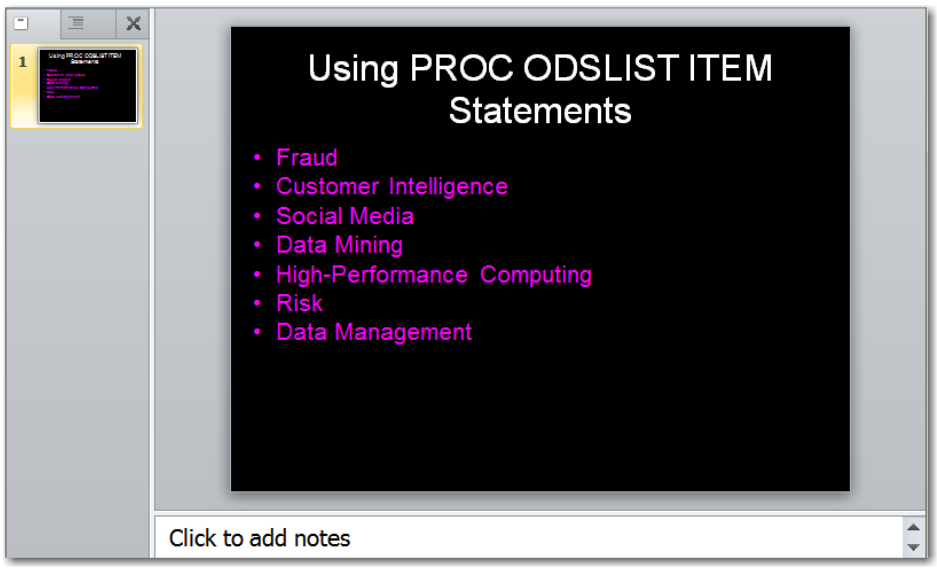
Close the open destinations.

```
ods _all_ close;
```

Output 7.3 *Slide with Default Style*



Output 7.4 *Slide with PowerPointDark Style*



Example 2: Creating Nested Lists

Features:

ITEM statement
- STYLE= option
- VALUE option

LIST statement

P statement

PROC ODSLIST statement
- NAME= option
- PRINT option
- STORE= option

ODS POWERPOINT statement

OPTIONS statement

TITLE statement

Details

PROC ODSLIST enables you to easily nest lists and add text. The following example creates a nested list inside the first item of the list. You can use the P statement to add text that is not in a list.

Program

```
ods html close;
options nodate;
title 'Creating Nested Lists';

ods powerpoint file="file-path/NestedList.ppt" style=powerpointdark;

proc odslist name=nested store=work.Myexampleslides print;


   item;
      p ' Fraud' ;
      list;
         item 'Consumer Fraud';
         item 'Business Fraud';
      end;
   end;

   item ' Customer Intelligence';
   item ' Social Media' ;
   item ' Data Mining';
   item ' High-Performance Computing';
   item ' Risk';
   item ' Data Management';
```

```
run;
ods _all_ close;
```

Program Description

Close the HTML destination and set the SAS system options. The HTML destination is open by default in SAS Windowing environment. If you are not creating HTML output, then close the destination to save system resources.

```
ods html close;
options nodate;
title 'Creating Nested Lists';
```

Open the ODS destination for PowerPoint. The ODS POWERPOINT statement creates output formatted for the ODS destination for PowerPoint.

```
ods powerpoint file="file-path/NestedList.ppt" style=powerpointdark;
```

Begin the ODSLIST procedure and create a template store and name for the template. The NAME= option gives the template a name and the STORE= option specifies the template store to put the template in. If the specified template store does not exist, then ODS creates it. Because there is no DATA= option specified, you must use the PRINT option to render the output. Otherwise, the template is created, but no output is rendered.

```
proc odslist name=nested store=work.Myexampleslides print;
```

Create a nested list. The first ITEM statement creates the list item that contains the nested list. The P statement specifies the paragraph that contains the nested list. The LIST statement begins the list, and the ITEM statements create the list items.

```
   item;
      p ' Fraud' ;
      list;
         item 'Consumer Fraud';
         item 'Business Fraud';
      end;
   end;
```

Create the remaining list items. The ITEM statements that are not nested create the remaining list items.

```
   item ' Customer Intelligence';
   item ' Social Media' ;
   item ' Data Mining';
   item ' High-Performance Computing';
   item ' Risk';
   item ' Data Management';
run;
```

Close the open destinations.

```
ods _all_ close;
```

Output 7.5 *Nesting Lists*



Example 3: Customizing Lists

Features:

LIST statement

P statement

PROC ODSLIST statement
- NAME= option
- PRINT option
- STORE= option

ITEM statement
- STYLE= option

ODS ESCAPECHAR statement

FOOTNOTE statement

ODS POWERPOINT statement

OPTIONS statement

TITLE statement

Details

You can customize any list created with PROC ODSLIST. You can use the STYLE= option on the ITEM, P, or LIST statement. You can specify customizations for the entire list with the STYLE= option in the LIST statement. You can also make specific changes to each item or text with the STYLE= option on the ITEM or P statement.

Program

```
ods html close;
options nodate;

ods escapechar = "^";
title 'Customizing Lists';
footnote  "Marketing Data ^{nbspace 50}  www.sas.com ";


ods powerpoint file="file-path/a.ppt" style=powerpointdark;

proc odslist name=nested store=work.Myexampleslides print;

item;
      p ' Topics For This Week';
      list / style={liststyletype=decimal fontweight=bold fontsize=1cm
color=blue};

      item / style=[fontweight=bold fontsize=1cm color=red] value=1;
         p ' Fraud' ;
         list / style={liststyletype=decimal color=purple};
            item 'Consumer Fraud ^{super January-June}' / value=1;
            item 'Business Fraud';
         end;
      end;

      item ' Customer Intelligence' ;
      item ' Social Media';
      item ' Data Mining';
      item ' High-Performance Computing';
      item ' Risk ^{style [font_style=italic color=purple]
                  ^{sub (* Allen and sons)}}';
      item ' Data Management';
   end;
end;
run;


title;
proc odslist;
   item;
     p ' Topics For Today';
     list / style={liststyletype=decimal };
        item ' Customer Intelligence' / style=[fontweight=bold
fontsize=1cm
                                       color=purple] value=1;
        item ' Social Media' / style=[fontweight=bold fontsize=1cm ];
        item ' Data Mining' / style=[fontweight=bold fontsize=1cm ];
     end;
   end;
run;



ods _all_ close;
```

Program Description

Close the HTML destination and specify the SAS system options. The HTML destination is open by default in SAS Windowing environment. If you are not creating HTML output, then close the destination to save system resources.

```
ods html close;
options nodate;
```

Specify titles and footnotes and define a representative character to be used in output strings.

```
ods escapechar = "^";
title 'Customizing Lists';
footnote  "Marketing Data ^{nbspace 50}  www.sas.com ";
```

Open the ODS destination for PowerPoint. The ODS POWERPOINT statement creates output formatted for the ODS destination for PowerPoint.

```
ods powerpoint file="file-path/a.ppt" style=powerpointdark;
```

Begin the ODSLIST procedure and create a template store and name for the template. The NAME= option gives the template a name and the STORE= option specifies the template store to put the template in. If the specified template store does not exist, then ODS creates it. Because there is no DATA= option specified, you must use the PRINT option to render the output. Otherwise, the template is created but no output is rendered.

```
proc odslist name=nested store=work.Myexampleslides print;
```

Begin creating content for your output. The ITEM statement without an expression specified begins an ITEM block. The P statement specifies explanatory text for the list. The LIST statement begins the list. LIST statements must be specified within an ITEM block. The STYLE= option that is specified in the LIST statement applies to all of the list items within the list.

```
item;
      p ' Topics For This Week';
      list / style={liststyletype=decimal fontweight=bold fontsize=1cm
color=blue};
```

Create a nested list. The LIST statement within the P statement creates a nested list for the first item on the list. The STYLE= option in the LIST statement specifies style attributes to format the text for all of the items in the list. The END statements end the LIST and ITEM blocks. A STYLE= option specified in an ITEM statement overrides the STYLE= option specified in the LIST statement. The color “blue” is applied to all of the items in overall list except the first item, which has a STYLE= option specified. The color “red” is applied to the first item only.

```
      item / style=[fontweight=bold fontsize=1cm color=red] value=1;
         p ' Fraud' ;
         list / style={liststyletype=decimal color=purple};
            item 'Consumer Fraud ^{super January-June}' / value=1;
            item 'Business Fraud';
         end;
      end;
```

Create the remaining list items. The ITEM statements that are not nested create the remaining list items. The customization for each item comes from the STYLE=

option specified on the first LIST statement. The STYLE= option on the “Risk” item specifies style attributes to format the text for that item only.

```
        item ' Customer Intelligence' ;
        item ' Social Media';
        item ' Data Mining';
        item ' High-Performance Computing';
        item ' Risk ^{style [font_style=italic color=purple]
                    ^{sub (* Allen and sons)}}';
        item ' Data Management';
     end;
  end;
  run;
```

Create a second slide. The second ODSLIST procedure block creates a list on a second slide. The STYLE= option specifies style attributes to format the text.

```
  title;
  proc odslist;
     item;
       p ' Topics For Today';
       list / style={liststyletype=decimal };
          item ' Customer Intelligence' / style=[fontweight=bold
  fontsize=1cm
                                          color=purple] value=1;
          item ' Social Media' / style=[fontweight=bold fontsize=1cm ];
          item ' Data Mining' / style=[fontweight=bold fontsize=1cm ];
       end;
     end;
  run;
```

Close the open destinations.

```
  ods _all_ close;
```

Slide 1

Output 7.6 *Slide 1*



Output 7.7 *Slide 2*



Example 4: Comparing Lists Created with PROC ODSLIST and PROC ODSTEXT

Features:

ODSLIST Procedure
- CELLSTYLE AS statement
- END statement
- ITEM statement
- LIST statement
- P statement:
- PROC ODSLIST statement
- TRANSLATE INTO statement

ODSTEXT Procedure
- CELLSTYLE AS statement
- END statement
- ITEM statement
- LIST statement
- P statement
- PROC ODSTEXT statement
- TRANSLATE INTO statement

OPTIONS statement

TITLE statement

Details

The following examples create similar output using PROC ODSLIST and PROC ODSTEXT. Both procedures can be used to create lists and paragraphs. However, there are differences in how the syntax of the two procedures can be specified. The following table shows how the P, ITEM, and LIST statements are specified in each procedure.

PROC ODSTEXT	PROC ODSLIST
The P statement can be used at the top level of PROC ODSTEXT as well as within list items.	The P statement can be specified only within an ITEM block.
The ITEM statement must be used within a LIST block.	The ITEM statement does not have to be specified within a LIST block.
The LIST statement does not have to be specified within an item block.	The LIST statement must be used within an item block.

There are also slight differences in the appearance of the output created by the two procedures. Examine the output created by the two programs in this example. Notice that the spacing and bullet points are slightly different.

Program: Using PROC ODSLIST

```
options nodate nonumber obs=7;
title "Students' Names, Ages, and Weights";


proc odslist data=sashelp.class;
   item;
      p 'List of Students'' Names, Ages, and Weights' /
style=systemtitle;
      p 'Weights of children younger than fourteen are hidden for
legal reasons'
        / style={fontstyle=italic};

      list;
         cellstyle age<=13 as datastrong{background=green},
                   age>=14 as {background=lightpurple};

         item 'Student name is ' || name;
         item 'Age: ' || put(age, 2.);

         item;
            translate age<=13 into 'Weight: N/A';
            p 'Weight: ' || put(weight, 3.);
         end;

      end;
   end;
run;
```

Program Description

Specify the SAS system options and title.

```
options nodate nonumber obs=7;
title "Students' Names, Ages, and Weights";
```

Begin the ODSLIST procedure, begin an ITEM block, and specify explanatory text. The PROC ODSLIST statement specifies the input data set and begins the procedure. The ITEM statement with no options specified begins an ITEM block. The P statements specify the explanatory text. In the ODSLIST Procedure, the P statement must be specified within an ITEM block.

```
proc odslist data=sashelp.class;
   item;
      p 'List of Students'' Names, Ages, and Weights' /
style=systemtitle;
      p 'Weights of children younger than fourteen are hidden for
legal reasons'
        / style={fontstyle=italic};
```

Begin the list and conditionally set the style attributes. The LIST statement with no options specified begins a list block. The LIST statement must be used within an item block. The CELLSTYLE AS statement sets the style attributes of the list items based on the value of the variable Age. If the value of the variable Age is less than

or equal to 13, the background color is green. If the value of the variable Age is greater than or equal to 14, the background color is light purple.

```
        list;
           cellstyle age<=13 as datastrong{background=green},
                     age>=14 as {background=lightpurple};
```

Specify the items. The ITEM statements specify the content of each item.

```
           item 'Student name is ' || name;
           item 'Age: ' || put(age, 2.);
```

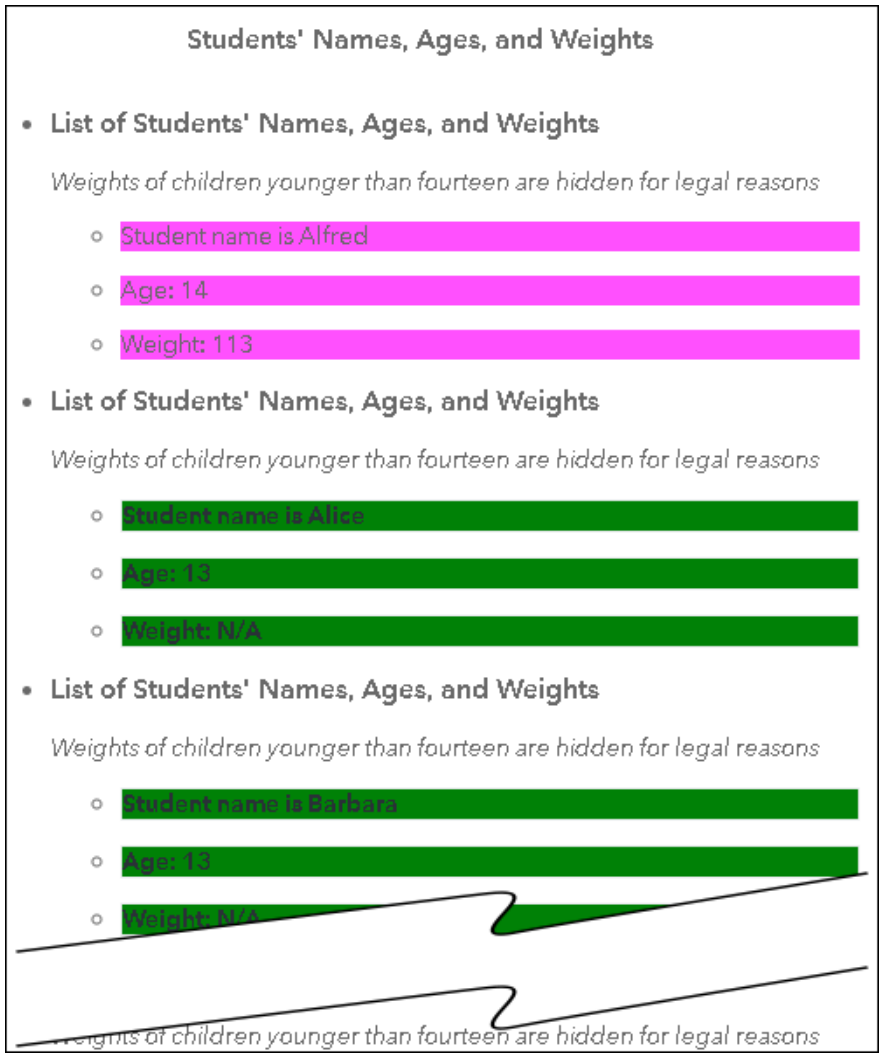
Conditionally transform the value for the AGE variable. The TRANSLATE INTO statement translates the value of Age based on the conditions specified. For values of the variable AGE that are equal to or less than thirteen, "N/A" is written to the output.

```
           item;
              translate age<=13 into 'Weight: N/A';
              p 'Weight: ' || put(weight, 3.);
           end;
```

End the LIST and ITEM blocks. The LIST block began with the LIST statement. An END statement ends the LIST block. Both ITEM blocks began with an ITEM statement with no expression specified. An END statement ends each ITEM block.

```
        end;
     end;
  run;
```

Output 7.8 *Output Created with the PROC ODSLIST*



Program: PROC ODSTEXT

```
options nodate nonumber obs=7;
title "Students' Names, Ages, and Weights";

proc odstext data=sashelp.class;
   p 'List of Students'' Names, Ages, and Weights' / style=systemtitle;
   p 'Weights of children younger than fourteen are hidden for legal
reasons'
     / style={fontstyle=italic};

   list;
      cellstyle age<=13 as datastrong{background=green},
                age>=14 as {background=lightpurple};

      item 'Student name is ' || name;
      item 'Age: ' || put(age, 2.);

      item;
         translate age<=13 into 'Weight: N/A';
```

```
         p 'Weight: ' || put(weight, 3.);
      end;

   end;
run;
```

Program Description

Specify the SAS system options and title.

```
options nodate nonumber obs=7;
title "Students' Names, Ages, and Weights";
```

Begin the ODSTEXT procedure and specify explanatory text for the list. The PROC ODSLIST statement specifies the input data set and begins the procedure. The P statements specify the text that precedes the list. In the ODSTEXT procedure, the P statement does not have to be specified within an ITEM block.

```
proc odstext data=sashelp.class;
   p 'List of Students'' Names, Ages, and Weights' / style=systemtitle;
   p 'Weights of children younger than fourteen are hidden for legal
reasons'
     / style={fontstyle=italic};
```

Begin the list and conditionally set the style attributes. The LIST statement with no options specified begins a list block. In the ODSTEXT procedure, the LIST statement does not have to be specified within an item block. The CELLSTYLE AS statement sets the style attributes of the list items based on the value of the variable Age. If the value of the variable Age is less than or equal to 13, the background color is green. If the value of the variable Age is greater than or equal to 14, the background color is light purple.

```
      list;
         cellstyle age<=13 as datastrong{background=green},
                   age>=14 as {background=lightpurple};
```

Specify the items. The ITEM statements specify the content of each item.

```
         item 'Student name is ' || name;
         item 'Age: ' || put(age, 2.);
```

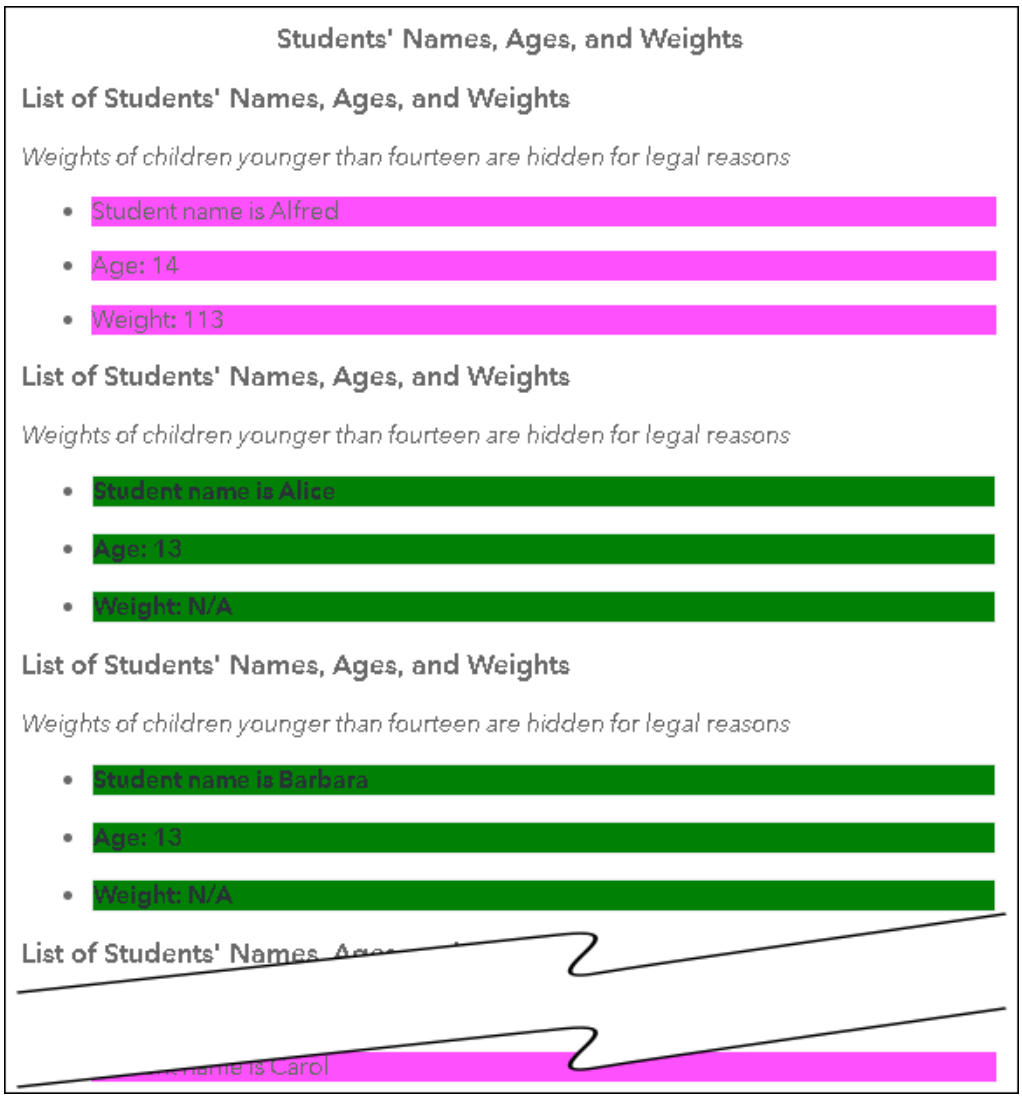
Conditionally transform the value for the AGE variable. The TRANSLATE INTO statement translates the value of Age based on the conditions specified. For values of the variable AGE that are equal to or less than thirteen, "N/A" is written to the output. The TRANSLATE INTO statement must be specified within the block of the variable you want to be translated. Otherwise, the variable is translated and the results apply to all variables.

```
         item;
            translate age<=13 into 'Weight: N/A';
            p 'Weight: ' || put(weight, 3.);
         end;
```

End the LIST block. The LIST block began with the LIST statement. An END statement ends the LIST block.

```
      end;
run;
```

Output 7.9 *Output Created with the PROC ODSTEXT*



PART 5

The ODSTABLE Procedure

Chapter 8
ODSTABLE Procedure **241**

Chapter 8

ODSTABLE Procedure

Overview: ODSTABLE Procedure ***241***
What Does the ODSTABLE Procedure Do? 242

Concepts: ODSTABLE Procedure ***242***
Comparing PROC TEMPLATE and PROC ODSTABLE 242

Syntax: ODSTABLE Procedure ***244***
PROC ODSTABLE Statement 244
BY Statement 246
CELLSTYLE AS Statement 250
COLUMN Statement 253
COMPUTE AS Statement 254
DEFINE Statement 256
DYNAMIC Statement 258
END Statement 258
FOOTER Statement 259
HEADER Statement 260
MVAR Statement 261
NMVAR Statement 262
NOTES Statement 262
TRANSLATE INTO Statement 263

Usage: ODSTABLE Procedure ***265***
Viewing the Contents of a Table Template 265
Values in Table Columns and How They Are Justified 266
Formatting Values in Table Columns 267
Stacking Values for Two or More Variables 268

Examples: ODSTABLE Procedure ***269***
Example 1: Customizing Columns 269
Example 2: Defining Variables with the COLUMN Statement 271
Example 3: Creating and Storing a Customized Table Template 273
Example 4: Using PROC TEMPLATE and PROC ODSTABLE 275

Overview: ODSTABLE Procedure

What Does the ODSTABLE Procedure Do?

By default, ODS output is formatted according to the various definitions or templates that the procedure or DATA step specify. Table templates describe how tables should be constructed. This includes the content and placement of headers and footers, the content and placement of columns, and style overrides. All SAS procedures, except PROC PRINT, PROC REPORT, and PROC TABULATE, use table templates to describe how their tables should look. This means that you can change the structure of tables that are generated by SAS procedures by using table templates.

You can create your own new tabular output templates by using the ODSTABLE procedure. The Output Delivery System then uses these templates to produce customized tabular output. With the ODSTABLE procedure, you can create table templates and bind them with the input data set in one statement. You can also name your templates and store them in a template store.

Concepts: ODSTABLE Procedure

Comparing PROC TEMPLATE and PROC ODSTABLE

The ODSTABLE procedure is a simpler way of producing the same output that you would expect to get from using the DEFINE TABLE statement in PROC TEMPLATE. You can use the same statements in an ODSTABLE block as you would use in a DEFINE TABLE block in PROC TEMPLATE. For example, the table templates created in both of the following programs are equivalent. The statements between the DEFINE TABLE statement and the END statement in the PROC TEMPLATE step are identical to the statements in the PROC ODSTABLE step.

Example Code 8.1 *Table Template Created with PROC TEMPLATE*

```
proc template;
   define table Base.Summary;
      notes "Summary table for MEANS and SUMMARY";
      dynamic clmpct one_var_name one_var_label one_var;
```

```
      column class nobs id type ways (varname) (label) (min)
         (max) (range)
         (n         ) (nmiss) (sumwgt) (sum) (mean) (uss)
         (css) (var) (stddev) (cv)
         (        stderr) (t) (probt) (lclm) (uclm) (skew)
         (kurt) (median) (mode) (q1)
         (q3) (qrange) (p1) (p5) (p10) (p25) (p50) (p75)
         (p90) (p95) (p99);

      define nobs;
        style={color=orange backgroundcolor=white};

      end;

   end;
run;
```

Example Code 8.2 *Table Template Created with PROC ODSTABLE*

```
proc odstable name=Base.Summary;
      notes "Summary table for MEANS and SUMMARY";
      dynamic clmpct one_var_name one_var_label one_var;
      column class nobs id type ways (varname) (label) (min)
         (max) (range)
         (n         ) (nmiss) (sumwgt) (sum) (mean) (uss)
         (css) (var) (stddev) (cv)
         (        stderr) (t) (probt) (lclm) (uclm) (skew)
         (kurt) (median) (mode) (q1)
         (q3) (qrange) (p1) (p5) (p10) (p25) (p50) (p75)
          (p90) (p95) (p99);

      define nobs;
        style={color=orange backgroundcolor=white};

      end;
run;
```

Table 8.1 *Differences between PROC TEMPLATE and PROC ODSTABLE*

PROC TEMPLATE with DEFINE TABLE Statement	PROC ODSTABLE
Creates and modifies table templates.	Creates and modifies table templates.
Creates and modifies column, header, and footer templates in a single procedure step.	Improves the readability of programs by creating one template type in one procedure step.
Must use a DATA _NULL_ step to bind the data to the table template.	Creates data-dependent table templates without using the DATA _NULL_ step.
Does not require an input data set.	No output is produced if you specify the NAME= option without the DATA= option. Without the DATA= option, ODS creates the template but does not render the output.

Syntax: ODSTABLE Procedure

PROC ODSTABLE DATA= *data-set-name* | NAME=*template-name*
BY <DESCENDING> *variable-1*
<… <DESCENDING> *variable-n*>
<NOTSORTED>;
<PAGEBREAK= NO | YES ><STORE=*template-store*>;
<*table-attribute-1* < *table-attribute-2*>…>;
CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*] ><, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*]>>;
COLUMN *column(s)*;
DEFINE *template-type template-name* </ *options*>;
statements-and-attributes
END;
DYNAMIC *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;
FOOTER *footer-name(s)*;
HEADER *header-name(s)*;
MVAR *variable-1* <=*default-variable-1*><'*text-1*'>
<… *variable-n* <=*default-variable-n*><'*text-n*'>>;
NMVAR *variable-1* <=*default-variable-1*><'*text-1*'>
<… *variable-n* <=*default-variable-n*><'*text-n*'>>;
NOTES "*text*";
TRANSLATE *expression-1* **INTO** *expression-2* < , *expression-n* INTO *expression-m*;>

PROC ODSTABLE Statement

Begins a table template.

Requirement: You must specify the DATA= option or the NAME= option.

Syntax

PROC ODSTABLE DATA=*data-set-name* | **NAME**=*template-name* <CONTENTS= "*text-string*">
<PAGEBREAK= NO | YES ><STORE=*template-store*>;
<*table-attribute-1* < *table-attribute-2*>…>;

Required Arguments

DATA=*data-set-name*
specifies the input data.

Examples “Example 1: Customizing Columns” on page 269

“Example 3: Creating and Storing a Customized Table Template” on page 273

NAME= *template-name*
specifies the name of the template. Specifying a template name with the NAME= option is the same as specifying a template name with the DEFINE TABLE statement in PROC TEMPLATE.

For example, the following two code blocks are equivalent:

```
proc odstable name=mytable;
run;
```

```
proc template;
   define table mytable;
   end;
run;
```

Examples “Example 3: Creating and Storing a Customized Table Template” on page 273

“Example 4: Using PROC TEMPLATE and PROC ODSTABLE ” on page 275

Optional Arguments

CONTENTS="*text-string*"
specifies the title for the table of contents. The CONTENTS= option overrides the generated table of contents text.

PAGEBREAK= NO | YES
specifies whether the procedure should generate a page break.

NO
specifies that no page break is generated.

Alias OFF

YES
specifies that a page break is generated.

Alias ON

Default YES

STORE=*template-store*
specifies the template store where the compiled template is placed. If you do not specify a name or template store, ODS stores the template in the first writable template store in the ODS path. By default, this template store is Workr.Templat.

If you do specify a name and template store, but do not have a libref specified, then the template is stored in the Work directory. For example, the statement

`proc odstable name=mylist store=mystore;` results in a template store named Work.mystore.

Using the STORE= option is the same as using the STORE= option in the DEFINE statement in PROC TEMPLATE. For example, the following two code blocks are equivalent:

```
proc odstable name=mytable store=work.mystore(update);
run;
```

```
proc template;
    define table mytable / store=work.mystore(update);
    end;
run;
```

Restriction The STORE= option can only be used if the NAME= option is specified.

Examples "Example 3: Creating and Storing a Customized Table Template" on page 273

"Example 4: Using PROC TEMPLATE and PROC ODSTABLE " on page 275

BY Statement

Orders the output according to the BY groups.

Note: BY-Group processing is included in release 2020.1.2 and later.

Syntax

BY <DESCENDING> *variable-1*
<… <DESCENDING> *variable-n*>
<NOTSORTED>;

Required Argument

variable
specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING
specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED
specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see *SAS DATA Step Statements: Reference*.
- For more information about PROC SORT, see “SORT Procedure” in *Base SAS Procedures Guide*.
- For more information about BY-group processing in CAS, see .“Grouping and Ordering” in *SAS Cloud Analytic Services: DATA Step Programming*
- For more information about creating indexes, see “INDEX CREATE Statement” in *Base SAS Procedures Guide*.

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

1. The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
2. The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
3. The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, ‘ABCD’ in the primary data set matches ‘ABCD ’ in the secondary data set. If the secondary data set contained ‘ABCDEF’, then they would not match.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT
ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

"BY Statement" in *SAS DATA Step Statements: Reference*

CELLSTYLE AS Statement

For tables, sets the style element of the cells in the table or column according to the values of the variables. For text, sets the style attributes of the list items or paragraphs. Use this statement to set the presentation characteristics (such as foreground color and font face) of individual cells or text.

Restriction: The CELLSTYLE AS statement can be used only within a column template, an ODS list, an ODS textblock, or a table template.

Syntax

CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*]> <, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*] >>;

Required Arguments

expression
is an expression that is evaluated for each list item, paragraph, or table cell.

If *expression* resolves to TRUE (a nonzero value), the style element that is specified is used for the current cell. If *expression* is FALSE, the next expression in the statement is evaluated. Thus, you can string multiple expressions together to format cells conditionally.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

An operator is a symbol that requests a string, a comparison, logical operation, or arithmetic calculation. An operand is one of the following:

constant
is a fixed value such as the name of a column or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in-variable
is a special type of WHERE expression operand that helps you find common values in table or column templates.

Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example “Example 5: Setting the Style Element for a Specific Column, Row, and Cell” on page 570

DATANAME
is a data column name.

DATATYPE
is the data type of the column variable. The data type is either numeric ("num") or character ("char").

Example The following CELLSTYLE AS statement specifies that numeric column variables have a red font color and character column variables have a blue font color:

```
cellstyle   _datatype_ = "num" as {color=red},
            _datatype_ = "char" as {color=blue};
```

LABEL
is a column label.

Example “Example 5: Setting the Style Element for a Specific Column, Row, and Cell” on page 570

ROW
is a row number. Row numbering begins with 1.

Example “Example 5: Setting the Style Element for a Specific Column, Row, and Cell” on page 570

STYLE
is a style element name.

See For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

Example “Example 6: Creating Top-Level Templates” on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example “Example 6: Creating Top-Level Templates” on page 577

comparison-operator
compares a variable with a value or with another variable.

The following table lists the comparison operators:

Table 8.2 Comparison Operators

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one or more from a list of values

Tip Using an expression of 1 as the last expression in the CELLSTYLE AS statement sets the style element for any cells that did not meet an earlier condition. For a table of style element names, see Chapter 20, "Style Elements," on page 777.

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

style-attribute-specification
describes a style attribute to set.

Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

For information about the style attributes that you can set in a table template, see "About Style Attributes" on page 465.

Optional Argument

style-element-name
is the name of a style element that is part of a style that is registered with the Output Delivery System.

SAS provides some styles. You can create customized styles and style elements with PROC TEMPLATE by using the "DEFINE STYLE Statement" on page 454. For a table of style element names, see Chapter 20, "Style Elements," on page 777.

The following style elements are frequently used with the CELLSTYLE AS statement:

- Data
- DataFixed
- DataEmpty
- DataEmphasis
- DataEmphasisFixed
- DataStrong
- DataStrongFixed
- ListItem
- ListItem2
- Paragraph

The style element provides the basis for displaying the cell. Additional style attributes modify the display.

Default Data

See Chapter 14, “TEMPLATE Procedure,” on page 433

For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

COLUMN Statement

Declares a symbol as a column in the table and specifies the order of the columns.

Restriction: The COLUMN statement can be used only within a table template.

Examples: “Example 3: Creating a New Table Template ” on page 557

“Example 2: Defining Variables with the COLUMN Statement” on page 271

Syntax

COLUMN *column(s)*;

Required Argument

column

is one or more columns. If the column is defined outside the current table template, reference it by its path in the template store. Columns in the template are laid out from left to right in the same order that they are specified in the COLUMN statement.

Defaults If you omit a COLUMN statement, ODS makes a column for each column template (DEFINE COLUMN statement), and places the columns in the same order that the column templates have in the table template.

If you use a COLUMN statement but omit a DEFINE COLUMN statement for any of the columns, ODS uses a default column template that is based on the type of data in the column.

Interaction If you specify the column attribute PRINT=OFF, then the value of a column is turned off if the column is part of a stacked column. If all columns in a stacked column have PRINT=OFF set, then the entire column is removed from the table.

Tip Use a list of variable names, such as DAY1–DAY10, to specify multiple variables.

See “Stacking Values for Two or More Variables ” on page 549

COMPUTE AS Statement

Computes values for a column that is not in the data component, or modifies the values of a column that is in the data component.

Restriction: The COMPUTE AS statement can be used only within a column template.

Example: “Example 2: Defining Variables with the COLUMN Statement” on page 271

Syntax

COMPUTE AS *expression*;

Required Argument

expression
is an expression that assigns a value to each table cell in the column.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical sequence of operators and operands. An operator is a symbol that requests a comparison, a logical operation, or an arithmetic calculation.

An operand is one of the following:

constant
is a fixed value, such as the name of a column, or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

To reference another column in a COMPUTE AS statement, use the name of the column. In addition, if the column has values in the data component, you can reference the column itself in the expression.

For example, this DEFINE COLUMN block defines a column that contains the square root of the value in the column called Source:

```
define column sqroot;
   compute as sqrt(source);
   header="Square Root";
   format=6.4;
end;
```

function
specifies a SAS function. For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in variable
is a special type of WHERE expression operand that helps you find common values in column templates.

Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

DATANAME
is a data-column name.

LABEL
is a column label.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

ROW
is a row number. Row numbering begins with 1.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

STYLE
is a style-element name.

Example "Example 6: Creating Top-Level Templates" on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example "Example 6: Creating Top-Level Templates" on page 577

comparison-operator
compares a variable with a value or another variable.

Table 8.3 Comparison Operators

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Tip The COMPUTE AS statement can alter values in an output object. None of the templates that SAS provides modifies any values. To determine whether a template was provided by SAS, use the "ODS VERIFY Statement" in *SAS Output Delivery System: User's Guide*. If the template is not from SAS, the ODS VERIFY statement returns a warning when it runs the SAS program that uses the template. If you receive such a warning, use the SOURCE statement to look at the template and determine whether the COMPUTE AS statement alters values. (See "SOURCE Statement" on page 353.)

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

DEFINE Statement

Creates a template inside of a table template.

Restriction: The DEFINE statement can be used only inside a table template.

See: DEFINE COLUMN Statement on page 530
DEFINE FOOTER Statement on page 531
DEFINE HEADER Statement on page 532

Example: "Example 2: Defining Variables with the COLUMN Statement" on page 271

Syntax

DEFINE <COLUMN | FOOTER | HEADER> *template-name* < / NOLIST>;
statements-and-attributes;
END;

Required Argument

template-name
specifies the name of the new template.

Restriction *template-name* must be a single-level name.

Tip To reference the template that you are creating from another template, create it outside of the table template.

Optional Arguments

COLUMN
specifies to create a column template. For statements and attributes that can be used within a column template, see “DEFINE COLUMN Statement” on page 530.

FOOTER
specifies to create a footer template. For statements and attributes that can be used within a footer template, see “DEFINE FOOTER Statement” on page 531.

HEADER
specifies to create a header template. For statements and attributes that can be used within a header template, see “DEFINE HEADER Statement” on page 532.

Options to Use after a Slash Delimiter

NOLIST
preserves the *template-type* when inheriting it from another table template.

Tip If you specify an existing *template-name* without using the NOLIST option, then the template is overwritten.

Details

The template type, HEADER, FOOTER, or COLUMN, is optional only if you specify an existing template name. For example, if a column named Make exists in the template, the following DEFINE statement is valid:

```
define make;
```

For a complete example, see “Example 2: Defining Variables with the COLUMN Statement” on page 271.

DYNAMIC Statement

Defines a symbol that references a value that the data component supplies from the procedure or DATA step.

Restriction: The DYNAMIC statement can be used only in the template of an ODS textblock, ODS list, table, column, header, footer, or statistical graph template. A dynamic variable that is defined in a template is available to that template and to all the templates that it contains.

Syntax

DYNAMIC *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;

Required Argument

variable-name
names a variable that the data component supplies. ODS resolves the value of the variable when it binds the template and the data component.

Tip Dynamic variables are most useful to the authors of SAS procedures and to DATA step programmers.

Optional Arguments

value
sets the value of the variable.

text
is text that is placed in the template to explain the dynamic variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

END Statement

Ends the table template, header template, column template, or footer template.

Syntax

END;

FOOTER Statement

Declares a symbol as a footer in the table and specifies the order of the footers.

Example: "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

FOOTER *footer-specification(s)*;

Required Argument

footer-specification
is one or more footers. If the footer is defined outside the current table template, reference it by its path in the template store. Footers in the template are laid out from top to bottom in the same order that they are specified in the FOOTER statement.

Each *footer-specification* is one of the following:

"*string*"
specifies the text to use for the footer. If you specify a string, you do not need to specify a DEFINE FOOTER statement. However, you cannot specify any footer attributes except for a split character. If the SPLIT= attribute is not in effect and if the first character of the footer that you specify is neither a blank character nor an alphanumeric character, PROC TEMPLATE and PROC ODSTABLE treat it as the split character.

See "SPLIT= "*character*" | *variable*;" on page 611

footer-path
is the path of the footer template to use. A footer-path consists of one or more names, separated by periods. Each name represents a directory in a template store, which is a type of SAS file.

LABEL
uses the label of the output object as the footer. Each SAS procedure specifies a label for each output object that it creates. The DATA step uses the value of the OBJECTLABEL= option as the label of the output object. If OBJECTLABEL= is not specified, it uses the text of the first TITLE statement as the label.

Default If you omit a FOOTER statement, ODS makes a footer for each footer template (DEFINE FOOTER statement), and places the footers in the same order that the footer templates have in the table template.

HEADER Statement

Declares a symbol as a header in the table and specifies the order of the headers.

Syntax

HEADER *header-specification(s)*;

Required Argument

header-specification
is one or more headers. If the header is defined outside the current table template, reference it by its path in the template store. Headers in the template are laid out from top to bottom in the same order that they are specified in the HEADER statement.

Each *header-specification* is one of the following:

"*string*"
specifies the text to use for the header. If you specify a string, you do not need to use a DEFINE HEADER statement. However, you cannot specify any header attributes except for a split character. If the SPLIT= header attribute is not in effect and if the first character of the header that you specify is neither a blank character nor an alphanumeric character, PROC TEMPLATE and PROC ODSTABLE treat it as the split character.

See “SPLIT= "*character*" | *variable*;” on page 611

header-path
is the path of the header template to use. A header-path consists of one or more names, separated by periods. Each name represents a directory in a template store. (A template store is a type of SAS file.)

LABEL
uses the label of the output object as the header. Each SAS procedure specifies a label for each output object that it creates. The DATA step uses the value of the OBJECTLABEL= option as the label of the output object. If OBJECTLABEL= is not specified, it uses the text of the first TITLE statement as the label.

Default If you omit a HEADER statement, then ODS makes a header for each header template (DEFINE HEADER statement), and places the headers in the same order that the header templates have in the table template.

Example “Example 3: Creating a New Table Template ” on page 557

MVAR Statement

Defines a symbol that references a macro variable. ODS uses the value of the variable as a string. References to the macro variable are resolved when ODS binds the template and the data component to produce an output object.

Restriction: When replaying an ODS document with PROC DOCUMENT, values created by the MVAR statement must be re-created in the same session that is replaying the document.

Tip: You can use the MVAR statement in the template of an ODS list, ODS textblock, table, column, header, or footer. A macro variable that is defined in a template is available to that template and to all the templates that it contains.

See: "Example 3: Creating a New Table Template " on page 557 and "Example 1: Creating a Stand-Alone Style" on page 466

Syntax

MVAR *variable-name-1* <='*value-1*' <'*text-1*'>>
< *variable-name-2* <='*value-2*'> <'*text-2*'…>>;

Required Argument

variable-name
names a macro variable to reference in the template. ODS uses the value of the macro variable as a string. ODS does not resolve the value of the macro variable until it binds the template and the data component.

Tip Declare macro variables this way in a template. For example, to use the automatic macro variable SYSDATE9 in a template, declare it in an MVAR statement and reference it as SYSDATE9, without an ampersand, in the PROC TEMPLATE or PROC ODSTABLE step. If you use the ampersand, the macro variable resolves when the template is compiled instead of when ODS binds the template to the data component.

Optional Arguments

value
sets the default variable value.

text
is text that is placed in the template to explain the macro variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

NMVAR Statement

Defines a symbol that references a macro variable. ODS converts the variable's value to a number (stored as a double) before using it. References to the macro variable are resolved when ODS binds the template and the data component to produce an output object.

Restriction: The NMVAR statement can be used only in the template of an ODS list, ODS textblock, table, column, header, or footer. A macro variable that is defined in a template is available to that template and to all the templates that it contains.

See: "Example 4: Setting the Style Element for Cells Based on Their Values" on page 565

Syntax

NMVAR *variable-name-1* <='*value–1*' <'*text-1*'>>
< *variable-name-2* <='*value-2*'> <'*text-2*'…>>;

Required Argument

variable-name
names a macro variable to reference in the template. ODS converts the variable's value to a number (stored as a double) before using it. ODS does not resolve the macro variable until it binds the template and the data component.

Tip Declare macro variables this way in a template. For example, to use a macro variable as a number, declare it in an NMVAR statement and reference it without an ampersand. If you use the ampersand, the macro variable resolves when the template is compiled instead of when ODS binds the template to the data component.

Optional Arguments

value
sets the value of the variable.

text
is text that is placed in the template to explain the macro variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

NOTES Statement

Provides information about the table, header, column, or footer.

Restriction: The NOTES statement can be used only in the template of a table, column, header, or footer.

Tip: The NOTES statement becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

See: "Example 4: Setting the Style Element for Cells Based on Their Values" on page 565

Example: "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

NOTES '*text*';

Required Argument

text
provides information about the table.

TRANSLATE INTO Statement

Translates the specified numeric values to other values.

Restrictions: The TRANSLATE INTO statement can be used only in a column template, an ODS list, an ODS textblock, or a table template.

The TRANSLATE INTO statement in a table template applies only to numeric variables. To translate the values of a character variable, use TRANSLATE INTO in the template of that column.

Syntax

TRANSLATE *expression-1* **INTO** *expression-2* <, *expression-n* INTO *expression-m*>;

Required Argument

expression
is an expression that is evaluated for each list item, paragraph, table, or column cell that contains a numeric variable.

If *expression-1* resolves to TRUE (a nonzero value), the translation that is specified is used for the current cell. If *expression-1* is FALSE (zero), the next expression in the statement is evaluated. Thus, you can string multiple expressions together to format cells conditionally.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation.

An operand is one of the following:

constant
is a fixed value such as the name of a column or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in-variable
is a special type of WHERE expression operand that helps you find common values in table or column templates.

Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

DATANAME
is a data column name.

DATATYPE
is the data type of the column variable. The data type is either numeric ("num") or character ("char").

LABEL
is a column label.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

ROW
is a row number. Row numbering begins with 1.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

STYLE
is a style element name.

Example "Example 6: Creating Top-Level Templates" on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example "Example 6: Creating Top-Level Templates" on page 577

comparison-operator
compares a variable with a value or with another variable.

The following table lists the comparison operators:

Table 8.4 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one or more from a list of values

Restriction You cannot reference the values of other columns in *expression-1*.

Tip Using an expression of 1 as the last expression in the TRANSLATE–INTO statement specifies a translation for any cells that did not meet an earlier condition.

See You can use any expression that can be used in the WHERE= data set option. For information about expressions that you can use in the WHERE data set option, see the section on WHERE-Expression Processing in *SAS Programmer's Guide: Essentials*.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

Usage: ODSTABLE Procedure

Viewing the Contents of a Table Template

To view the contents of a table template use TEMPLATE procedure.

The SOURCE statement writes the source code for the specified template to the SAS log.

For example, to view the source code for all the objects in Base SAS, submit this code.

```
proc template;
   source base;
run;
```

Values in Table Columns and How They Are Justified

The process of justifying the values in columns in LISTING output is determined by the format of the variable and the values of two attributes: JUST= and JUSTIFY=. It is a three-step process:

1. ODS puts the value into the format for the column. Character variables are left-justified within their format fields; numeric variables are right-justified.
2. ODS justifies the entire format field within the column width according to the value of the JUST= attribute for the column, or, if that attribute is not set, JUST= for the table. For example, if you right-justify the column, the format field is placed as far to the right as possible. However, the placement of the individual numbers and characters within the field does not change. Thus, decimal points remain aligned. If the column and the format field have the same width, then JUST= has no apparent effect because the format field occupies the entire column.
3. If you specify JUSTIFY=ON for the column or the table, ODS justifies the values within the column without regard to the format field. By default, JUSTIFY=OFF.

For example, consider this set of values:

```
123.45
234.5
.
987.654
```

If the values are formatted with a 6.2 format and displayed in a column with a width of 6, they appear this way, regardless of the value of JUST= (asterisks indicate the width of the column):

```
******
123.45
234.50
   .
987.65
```

If the width of the column increases to 8, then the value of JUST= does affect the placement of the values, because the format field has room to move within the column. Notice that the decimal points remain aligned but that the numbers shift in relation to the column width.

```
just=left                just=center                just=right
```

```
********                ********                ********
123.45                   123.45                   123.45
234.50                   234.50                   234.50
   .                        .                        .
987.65                   987.65                   987.65
```

Now, if you add JUSTIFY=ON, then the values are formatted within the column without regard to the format width. The results are as follows:

```
justify=on              justify=on              justify=on
just=left               just=center             just=right

********                ********                ********
123.45                   123.45                   123.45
234.50                   234.50                   234.50
.                          .                        .
987.65                   987.65                   987.65
```

All destinations except LISTING justify the values in columns as if JUSTIFY=ON.

Formatting Values in Table Columns

The process of formatting the values in columns in LISTING output is determined by the format of the variable and the values of three options: FORMAT=, FORMAT_WIDTH=, and FORMAT_NDEC=. It is a four-step process:

1. If you omit a FORMAT= option, then the format that the data component provides is used. If the data component does not provide a format, then ODS uses one of the following:
   - best8. for integers
   - D12.3 for doubles
   - the length of the variable for character variables
2. If a format width is specified in the FORMAT= option, then takes precedence over the FORMAT_WIDTH= and FORMAT_NDEC= options.
3. If you specify a decimal width with the FORMAT= and FORMAT_NDEC= options, then the format that is specified with the FORMAT= option is used.
4. If you specify a format width with the FORMAT= and FORMAT_WIDTH= options, then the format that is specified with FORMAT= option is used.

The formatting attributes of a column are determined by the data component or the column template. This table summarizes the behavior of the column formatting attributes based on which attributes the column template provides.

Table 8.5 *Summary of Column Formatting Attributes*

Specifications Provided by the Column Template	Result
Nothing	Format name, width, and number of decimal places are determined by the data component.

Specifications Provided by the Column Template	Result
Format name	Format name and width are determined by the column template; number of decimal places is determined by the data component.
Format name and width	Format name and width are determined by the column template.
Format name, width, and number of decimal places	All three are determined by the column template.
Width	No name is specified; width is determined by the column template; number of decimal places is determined by the data component.
Number of decimal places	No name is specified; width is determined by the data component; number of decimal places is determined by the column template.

Stacking Values for Two or More Variables

To stack values for two or more variables in the same column, put parentheses around the stacked variables. In such a case, the column header for the first column inside the parentheses becomes the header for the column that contains all the variables inside parentheses. For example, this COLUMN statement produces a template with the following characteristics:

- The value of NAME is in the first column by itself.
- The values of CITY and STATE appear in the second column with CITY above STATE. The header for this column is the header that is associated with CITY.
- The values HOMEPHONE and WORKPHONE appear in the third column with HOMEPHONE above WORKPHONE. The header for this column is the header that is associated with HOMEPHONE.

```
column name (city state) (homephone workphone);
```

Use the asterisk (*) in the COLUMN statement to change the layout of stacking variables. An asterisk between groups of variables in parentheses stacks the first item in the first set of parentheses above the first item in the next set of parentheses, and so on, until the last group of parentheses is reached. Then, the second item in the first group is stacked above the second item in the second group, and so on. For example, this COLUMN statement produces a report with the following characteristics:

- The value of NAME is in the first column by itself.
- The values of CITY and HOMEPHONE appear in the second column with CITY above HOMEPHONE. The header for this column is the header that is associated with CITY.

- The values STATE and WORKPHONE appear in the third column with STATE above WORKPHONE. The header for this column is the header that is associated with STATE.

```
column name (city state) * (homephone workphone);
```

Examples: ODSTABLE Procedure

Example 1: Customizing Columns

Features:

COLUMN statement

DATA= option

DEFINE statement

PROC ODSTABLE statement

Table attributes

- STYLE
- HEADER
- FORMAT

OPTIONS statement

TITLE statement

Program

```
options nodate obs=15;
title "Customizing Columns";

proc odstable data=sashelp.class;
   column age sex height weight;

   define age;
      style={fontsize=10pt just=l borderrightstyle=dashed
background=yellow};
      header='Age of Student';
      format=3.;
   end;
   define sex;
      style={fontsize=10pt just=l borderrightstyle=dashed};
      header='Gender';
   end;
   define height;
      style={fontsize=10pt just=l foreground=blue
borderrightstyle=dashed};
      header='Height';
```

```
   end;
run;
```

Program Description

Specify the SAS system options and titles.

```
options nodate obs=15;
title "Customizing Columns";
```

Begin the ODSTABLE procedure and specify the columns. The PROC ODSTABLE statement begins the procedure. The DATA= option specifies the input data set. The COLUMN statement specifies the columns.

```
proc odstable data=sashelp.class;
   column age sex height weight;
```

Customize the columns. The DEFINE statements modify the specified column. The HEADER table attribute specifies the text for the column heading. The STYLE table argument specifies font size, font color, justification, and line style.

```
   define age;
      style={fontsize=10pt just=l borderrightstyle=dashed
background=yellow};
      header='Age of Student';
      format=3.;
   end;
   define sex;
      style={fontsize=10pt just=l borderrightstyle=dashed};
      header='Gender';
   end;
   define height;
      style={fontsize=10pt just=l foreground=blue
borderrightstyle=dashed};
      header='Height';
   end;
run;
```

The following output uses the default HTMLBlue style, with customized columns.

Output 8.1 *Customizing Columns*

Customizing Columns

Age of Student	Gender	Height	Weight
14	M	69	112.5
13	F	56.5	84
13	F	65.3	98
14	F	62.8	102.5
14	M	63.5	102.5
12	M	57.3	83
12	F	59.8	84.5
15	F	62.5	112.5
13	M	62.5	84
12	M	59	99.5
11	F	51.3	50.5
14	F	64.3	90
12	F	56.3	77
15	F	66.5	112
16	M	72	150

Example 2: Defining Variables with the COLUMN Statement

Features: COLUMN statement
DEFINE statement
COMPUTE AS statement

Details

The COLUMN statement indicates which data columns are available for use anywhere in the template. A variable must be specified in the COLUMN statement for its value to be available. If you do not want a specific variable and column to appear in the output, you can suppress the variable by using the PRINT=OFF column attribute.

This example calculates a three percent holdback based on the invoice price and manufacturers suggested retail price.

Program

```
title "Dealer Profit Assuming Three Percent Holdback";
proc odstable data=sashelp.cars(obs=15);

   column Make Model Msrp Holdback;

   define holdback;
      compute as (msrp * .03) ;
      label="3% Dealer Holdback";
      format=dollar10.2;
   end;

   define msrp;
      print=off;
   end;
run;
```

Program Description

Specify the title and begin the ODSTABLE procedure.

```
title "Dealer Profit Assuming Three Percent Holdback";
proc odstable data=sashelp.cars(obs=15);
```

Specify the data columns to include in the table. The column statement specifies the columns that are available to be used in the template. Make, Model, and Msrp are in the data set. The Holdback column values will be calculated.

```
   column Make Model Msrp Holdback;
```

Create the Holdback data values. The define statement creates a column named Holdback. The COMPUTE AS statement specifies the calculation to be used. The LABEL and FORMAT column attributes specify the label and format that the values use.

```
   define holdback;
      compute as (msrp * .03) ;
      label="3% Dealer Holdback";
      format=dollar10.2;
   end;
```

Suppress the printing of the Msrp column. You must include the Msrp variable in the COLUMN statement so that the values are available for calculating the Holdback values. To suppress the printing of the Msrp column, use the DEFINE statement and the PRINT=OFF column attribute.

```
   define msrp;
      print=off;
   end;
run;
```

Output 8.2 *Output Showing Selected Column Variables*

Dealer Profit Assuming Three Percent Holdback

Make	Model	3% Dealer Holdback
Acura	MDX	$1,108.35
Acura	RSX Type S 2dr	$714.60
Acura	TSX 4dr	$809.70
Acura	TL 4dr	$995.85
Acura	3.5 RL 4dr	$1,312.65
Acura	3.5 RL w/Navigation 4dr	$1,383.00
Acura	NSX coupe 2dr manual S	$2,692.95
Audi	A4 1.8T 4dr	$778.20
Audi	A41.8T convertible 2dr	$1,078.20
Audi	A4 3.0 4dr	$955.20
Audi	A4 3.0 Quattro 4dr manual	$1,002.90
Audi	A4 3.0 Quattro 4dr auto	$1,034.40
Audi	A6 3.0 4dr	$1,099.20
Audi	A6 3.0 Quattro 4dr	$1,189.20
Audi	A4 3.0 convertible 2dr	$1,274.70

Example 3: Creating and Storing a Customized Table Template

Features:

PROC ODSTABLE statement

DATA= option

NAME= option

STORE= option

COLUMN statement

DEFINE statement

Table attributes

- STYLE
- HEADER
- FORMAT

OPTIONS statement

TITLE statement

Details

This example creates a table with customized columns. With the ODSTABLE procedure, you can give the table template a name and store it in the template store of your choice.

Program

```
libname mylib 'library-specification';
title;

proc odstable data=sashelp.class name=odstableExample
store=mylib.MyExampleTemplates;

   column age sex height weight;

   define age;
      style={fontsize=10pt just=l borderrightstyle=dashed
background=yellow};
      header='Age of Student';
      format=3.;
   end;
   define sex;
      style={fontsize=10pt just=l borderrightstyle=dashed};
      header='Gender';
   end;
   define height;
      style={fontsize=10pt just=l foreground=blue
borderrightstyle=dashed};
      header='Height';
   end;
run;
```

Program Description

Specify a libref.

```
libname mylib 'library-specification';
title;
```

Specify the name of the new table template and the template store. The DATA= option specifies the input data set. The NAME= option specifies the name of the table template. The STORE= option specifies the template store to put the template in.

```
proc odstable data=sashelp.class name=odstableExample
store=mylib.MyExampleTemplates;
```

Specify the columns. The COLUMN statement specifies the columns.

```
   column age sex height weight;
```

Customize the columns. The DEFINE statements modify the specified column. The HEADER table attribute specifies the text for the column heading. The STYLE table argument specifies font size, font color, justification, and line style.

```
   define age;
      style={fontsize=10pt just=l borderrightstyle=dashed
background=yellow};
      header='Age of Student';
      format=3.;
   end;
   define sex;
      style={fontsize=10pt just=l borderrightstyle=dashed};
      header='Gender';
   end;
   define height;
      style={fontsize=10pt just=l foreground=blue
borderrightstyle=dashed};
      header='Height';
   end;
run;
```

Example 4: Using PROC TEMPLATE and PROC ODSTABLE

Features:

PROC ODSTABLE statement:
- STORE= option
- NAME= option

CELLSTYLE-AS statement

PROC TEMPLATE
- DEFINE TABLE statement

ODS PDF statement

OPTIONS statement

PROC SQL

TITLE statement

Details

The following programs create the same output using two different methods. Using the PROC ODSTABLE statement is the same as using the combination of PROC TEMPLATE and DEFINE TABLE statements.

This example also creates a top-level table template. Top level templates are applied globally to all of your tabular output. For more information about top-level table templates, see “Base.Template.Table” on page 534.

Program: Creating a Table with PROC ODSTABLE

```
libname mylib 'library-path';

ods path reset;
ods path show;
ods html close;


options nodate;
ods pdf file="mylib.ProcOdstableTable.pdf";
title "Using PROC ODSTABLE";

proc odstable name=Base.Template.Table;

   define header myheader1;
      text "Use the CELLSTYLE-AS Statement to Customize Output";
      style={color=red};
   end;
   define header myheader2;
      text "Use PROC ODSTABLE to Create a Table Template";
      style={color=red};
   end;

   define footer myfooter;
      text "This output is formated with a top-level template.";
      style={color=blue};
   end;

   cellstyle _row_ in (13) as {BackgroundColor=Palegreen },
             _row_ in (12) as {BorderBottomStyle=Solid },
             _row_ in (11) as {BackgroundColor=Limegreen },
              mod(_row_,2) as {Background=Honeydew},
             _col_ = 1 as {Width=1.2in BorderLeftColor=Black},
             _col_ = 2 as {Width=2in},
             _col_ = 3 as {Width=.6in},
             _col_ = 4 as {Width=.5in},
             _col_ = 5 as {Width=.9in};
run;

proc sql;
   select * from sashelp.class;
run;
quit;

proc odstable name=Base.Template.Table store=mylib.mystore;

   cellstyle _row_ in (16) as {BackgroundColor=Palegreen },
      _row_ in (15) as {BorderBottomStyle=Solid },
      _row_ in (14) as {BackgroundColor=Limegreen },
       mod(_row_,2) as {Background=Honeydew},
      _col_ = 1 as {Width=1.2in BorderLeftColor=Black},
      _col_ = 2 as {Width=2in},
      _col_ = 3 as {Width=.6in},
      _col_ = 4 as {Width=.5in},
      _col_ = 5 as {Width=.9in};
run;
```

```
ods path (prepend) mylib.Mystore;
ods path show;

proc sql;
   select * from sashelp.class;
run;
quit;

ods pdf close;
ods html;

proc template;
   delete base.template.table;
   delete base.template.table / store=mylib.mystore;
run;

ods path reset;
```

Program Description

Specify a libref.

```
libname mylib 'library-path';
```

Set ODS path to default settings and display the current ODS path.

```
ods path reset;
ods path show;
ods html close;
```

Set the SAS system options, specify the PDF destination, and specify a title.

```
options nodate;
ods pdf file="mylib.ProcOdstableTable.pdf";
title "Using PROC ODSTABLE";
```

Create the table template Base.Template.Table. The PROC ODSTABLE statement creates the top-level template Base.Template.Table. This template is applied to every table created by SAS, unless it is overridden by another template created by PROC TEMPLATE or PROC ODSTABLE, removed with the DELETE statement, or manually removed from the item store. Because there is no STORE= option specified in the PROC ODSTABLE statement, the template is stored in the Sork.Templat template store.

```
proc odstable name=Base.Template.Table;
```

Create two headers and a footer. The DEFINE statement with “header” template type specified, creates a header. The DEFINE statement with “footer” template type specified, creates a footer. DEFINE statement blocks must end with an END statement.

```
define header myheader1;
   text "Use the CELLSTYLE-AS Statement to Customize Output";
   style={color=red};
end;
define header myheader2;
   text "Use PROC ODSTABLE to Create a Table Template";
   style={color=red};
end;
```

```
   define footer myfooter;
      text "This output is formated with a top-level template.";
      style={color=blue};
   end;
```

Format cells. The CELLSTYLE-AS statement specifies the style element and style attributes to use for cells in each of the rows in a table, which creates the alternating row colors in the output.

```
   cellstyle _row_ in (13) as {BackgroundColor=Palegreen },
             _row_ in (12) as {BorderBottomStyle=Solid },
             _row_ in (11) as {BackgroundColor=Limegreen },
              mod(_row_,2) as {Background=Honeydew},
             _col_ = 1 as {Width=1.2in BorderLeftColor=Black},
             _col_ = 2 as {Width=2in},
             _col_ = 3 as {Width=.6in},
             _col_ = 4 as {Width=.5in},
             _col_ = 5 as {Width=.9in};
run;
```

Select columns. The SQL procedure selects all columns from the Sashelp.Class data set.

```
proc sql;
   select * from sashelp.class;
run;
quit;
```

Create a second table template named Base.Template.Table. The PROC ODSTABLE statement creates the top-level template Base.Template.Table. This template is applied to every table created by SAS, unless it is overridden by another template created by PROC TEMPLATE or PROC ODSTABLE, removed with the DELETE statement, or manually removed from the item store. Because the STORE= option is specified in the PROC ODSTABLE statement, the template is placed in the template store Work.Mystore.

```
proc odstable name=Base.Template.Table store=mylib.mystore;
```

Format cells. The CELLSTYLE-AS statement specifies the style element and style attributes to use for cells in each of the rows in a table, which creates the alternating row colors in the output.

```
   cellstyle _row_ in (16) as {BackgroundColor=Palegreen },
      _row_ in (15) as {BorderBottomStyle=Solid },
      _row_ in (14) as {BackgroundColor=Limegreen },
       mod(_row_,2) as {Background=Honeydew},
      _col_ = 1 as {Width=1.2in BorderLeftColor=Black},
      _col_ = 2 as {Width=2in},
      _col_ = 3 as {Width=.6in},
      _col_ = 4 as {Width=.5in},
      _col_ = 5 as {Width=.9in};
run;
```

Add Work.Mystore to the beginning of the ODS search order path. The ODS PATH statement with PREPEND specified adds Work.Mystore to the beginning of the ODS search order path. This enables the template Base.Template.Table from the Work.Mystore item store to be used. The ODS PATH statement displays the modified ODS path.

```
ods path (prepend) mylib.Mystore;
```

```
ods path show;
```

Select columns, close the PDF destination, and open the HTML destination. The SQL procedure selects all columns from the Sashelp.Class data set. The PDF statement closes the PDF destination. The ODS HTML statement opens the HTML statement.

```
proc sql;
   select * from sashelp.class;
run;
quit;

ods pdf close;
ods html;
```

Delete the top-level templates. The DELETE statement deletes the top-level templates. If you do not delete them, they are applied to all of your tabular output until you do delete them.

```
proc template;
   delete base.template.table;
   delete base.template.table / store=mylib.mystore;
run;
```

Set the ODS path to default settings.

```
ods path reset;
```

The following output was created with PROC ODSTABLE. The headers and footers are created by the DEFINE statements.

Output 8.3 *Output Created with PROC ODSTABLE, with Headers and Footers*

Using PROC ODSTABLE

Use the CELLSTYLE-AS Statement to Customize Output

Use PROC ODSTABLE to Create a Table Template

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

This output is formated with a master template.

The following output was created with PROC ODSTABLE. There are no headers and footers for this output, because the DEFINE statement was not used in the second PROC ODSTABLE block..

Output 8.4 *Output Created with PROC ODSTABLE, No Headers or Footers*

Using PROC ODSTABLE

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

Program: Creating a Table with PROC TEMPLATE and the DEFINE TABLE Statement

```
ods path reset;
ods path show;
ods html close;


options nodate;
ods pdf file="mylib.ProctemplTable.pdf";
title "Using PROC TEMPLATE and the DEFINE TABLE Statement";

proc template;
   define table Base.Template.Table;

      define header myheader1;
         text "Use the CELLSTYLE-AS Statement to Customize Output";
         style={color=red};
      end;
```

```
      define header myheader2;
         text "Use PROC TEMPLATE to Create a Table Template";
         style={color=red};
      end;
      define footer myfooter;
         text "This output is formated with a top-level template.";
         style={color=blue};
      end;

      cellstyle _row_ in (13) as {BackgroundColor=Palegreen },
                _row_ in (12) as {BorderBottomStyle=Solid },
                _row_ in (11) as {BackgroundColor=Limegreen },
                 mod(_row_,2) as {Background=Honeydew},
                _col_ = 1 as {Width=1.2in BorderLeftColor=Black},
                _col_ = 2 as {Width=2in},
                _col_ = 3 as {Width=.6in},
                _col_ = 4 as {Width=.5in},
                _col_ = 5 as {Width=.9in};
   end;
run;

proc sql;
   select * from sashelp.class;
run;
quit;

proc template;
   define table Base.Template.Table / store=mylib.Mystore;

      cellstyle _row_ in (16) as {BackgroundColor=Palegreen },
                _row_ in (15) as {BorderBottomStyle=Solid },
                _row_ in (14) as {BackgroundColor=Limegreen },
                 mod(_row_,2) as {Background=Honeydew},
                _col_ = 1 as {Width=1.2in BorderLeftColor=Black},
                _col_ = 2 as {Width=2in},
                _col_ = 3 as {Width=.6in},
                _col_ = 4 as {Width=.5in},
                _col_ = 5 as {Width=.9in};
   end;
run;

ods path (prepend) mylib.Mystore;
ods path show;

proc sql;
   select * from sashelp.class;
run;
quit;
ods pdf close;
ods html;

proc template;
   delete base.template.table;
   delete base.template.table / store=mylib.mystore;
run;
```

Program Description

Set ODS path to default settings and display the current ODS path.

```
ods path reset;
ods path show;
ods html close;
```

Set the SAS system options, specify the PDF destination, and specify a title.

```
options nodate;
ods pdf file="mylib.ProctemplTable.pdf";
title "Using PROC TEMPLATE and the DEFINE TABLE Statement";
```

Create the table template Base.Template.Table. The DEFINE TABLE statement in PROC TEMPLATE creates the top-level template Base.Template.Table. This template is applied to every table created by SAS, unless it is overridden by another template created by PROC TEMPLATE or PROC ODSTABLE, removed with the DELETE statement, or manually removed from the item store. Because there is no STORE= option specified in the DEFINE TABLE statement, the template is stored in the Work.Templat template store.

```
proc template;
   define table Base.Template.Table;
```

Create two headers and a footer. The DEFINE statement with “header” template type specified, creates a header. The DEFINE statement with “footer” template type specified, creates a footer. DEFINE statement blocks must end with an END statement.

```
      define header myheader1;
         text "Use the CELLSTYLE-AS Statement to Customize Output";
         style={color=red};
      end;
      define header myheader2;
         text "Use PROC TEMPLATE to Create a Table Template";
         style={color=red};
      end;
      define footer myfooter;
         text "This output is formated with a top-level template.";
         style={color=blue};
      end;
```

Format cells. The CELLSTYLE-AS statement specifies the style element and style attributes to use for cells in each of the rows in a table, which creates the alternating row colors in the output.

```
      cellstyle _row_ in (13) as {BackgroundColor=Palegreen },
                _row_ in (12) as {BorderBottomStyle=Solid },
                _row_ in (11) as {BackgroundColor=Limegreen },
                 mod(_row_,2) as {Background=Honeydew},
                _col_ = 1 as {Width=1.2in BorderLeftColor=Black},
                _col_ = 2 as {Width=2in},
                _col_ = 3 as {Width=.6in},
                _col_ = 4 as {Width=.5in},
                _col_ = 5 as {Width=.9in};
   end;
run;
```

Select columns. The SQL procedure selects all columns from the Sashelp.Class data set.

```
proc sql;
   select * from sashelp.class;
run;
quit;
```

Create a second table template named Base.Template.Table. The DEFINE TABLE statement in PROC TEMPLATE creates the top-level template Base.Template.Table. This template is applied to every table created by SAS, unless it is overridden by another template created by PROC TEMPLATE or PROC ODSTABLE, removed with the DELETE statement, or manually removed from the item store. Because the STORE= option is specified in the DEFINE TABLE statement, the template is placed in the template store Work.Mystore.

```
proc template;
   define table Base.Template.Table / store=mylib.Mystore;
```

Format cells. The CELLSTYLE-AS statement specifies the style element and style attributes to use for cells in each of the rows in a table, which creates the alternating row colors in the output.

```
      cellstyle _row_ in (16) as {BackgroundColor=Palegreen },
                _row_ in (15) as {BorderBottomStyle=Solid },
                _row_ in (14) as {BackgroundColor=Limegreen },
                 mod(_row_,2) as {Background=Honeydew},
                _col_ = 1 as {Width=1.2in BorderLeftColor=Black},
                _col_ = 2 as {Width=2in},
                _col_ = 3 as {Width=.6in},
                _col_ = 4 as {Width=.5in},
                _col_ = 5 as {Width=.9in};
   end;
run;
```

Add Work.Mystore to the beginning of the ODS search order path. The ODS PATH statement with PREPEND specified adds Work.Mystore to the beginning of the ODS search order path. This enables the template Base.Template.Table from the Work.Mystore item store to be used. The ODS PATH statement displays the modified ODS path.

```
ods path (prepend) mylib.Mystore;
ods path show;
```

Select columns, close the PDF destination, and open the HTML destination. The SQL procedure selects all columns from the Sashelp.Class data set. The ODS PDF statement closes the PDF destination, and the ODS HTML statement opens the HTML destination.

```
proc sql;
   select * from sashelp.class;
run;
quit;
ods pdf close;
ods html;
```

Delete the templates. The DELETE statements delete the template Base.Template.Table from the Work.Templat template store and the Base.Template.Table from the Work.Mystore template store

```
proc template;
   delete base.template.table;
```

```
   delete base.template.table / store=mylib.mystore;
run;
```

The following output was created with PROC TEMPLATE and the DEFINE TABLE statement. The headers and footers are created by the DEFINE statements.

Output 8.5 *Output Created with PROC TEMPLATE, with Headers and Footers*

Using PROC TEMPLATE and the DEFINE TABLE Statement

Use the CELLSTYLE-AS Statement to Customize Output

Use PROC TEMPLATE to Create a Table Template

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

This output is formated with a master template.

The following output was created with PROC TEMPLATE and the DEFINE TABLE statement. There are no headers and footers for this output, because the DEFINE statement was not used in the second PROC ODSTABLE block..

Output 8.6 *Output Created with PROC TEMPLATE, No Headers or Footers*

Using PROC TEMPLATE and the DEFINE TABLE Statement

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84
John	M	12	59	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90
Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112

PART 6

The ODSTEXT Procedure

Chapter 9
ODSTEXT Procedure **289**

Chapter 9

ODSTEXT Procedure

Overview: ODSTEXT Procedure ***289***
What Does the ODSTEXT Procedure Do? 290

Syntax: ODSTEXT Procedure ***290***
PROC ODSTEXT Statement 291
BY Statement 292
CELLSTYLE AS Statement 296
DYNAMIC Statement 299
END Statement 300
HEADING Statement 300
ITEM Statement 302
LIST Statement 304
MVAR Statement 306
NMVAR Statement 307
P Statement 308
TRANSLATE INTO Statement 309

Usage: ODSTEXT Procedure ***312***
Using the ODSTEXT Procedure 312
Changing the Bullet Type for Lists 312

Examples: ODSTEXT Procedure ***315***
Example 1: Creating Content for the ODS Destination for PowerPoint 315
Example 2: Comparing Lists Created with PROC ODSLIST and PROC ODSTEXT 317

Overview: ODSTEXT Procedure

What Does the ODSTEXT Procedure Do?

The ODSTEXT procedure is used to create text block templates. These text block templates create lists and paragraphs for your output. You can use style attributes and formats to customize your content, and WHERE expressions to select your content. With PROC ODSTEXT, you can use the DATA= option to bind your data to the template without using a DATA step.

PROC ODSTEXT can be used with any output destination. However, the procedure is essential for creating content for the ODS destination for PowerPoint and e-books. See "ODS EPUB Statement" in *SAS Output Delivery System: User's Guide* and "ODS POWERPOINT Statement" in *SAS Output Delivery System: User's Guide* for more information about creating output for the ODS destination for PowerPoint and e-book output.

Syntax: ODSTEXT Procedure

PROC ODSTEXT <CONTENTS=*text-string*><DATA=*SAS-data-set*>
<NAME=*template-name*>
<PAGEBREAK=NO | YES> <PRINT><=*template-store*>;
STORE
BY <DESCENDING> *variable-1*
<… <DESCENDING> *variable-n*>
<NOTSORTED>;
CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*] >
<, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*]>>;
DYNAMIC *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;
HEADING<*n*> <"*heading-text*"> / <STYLE=*style-override*>
MVAR *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;
NMVAR *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;
LIST / <START=*integer-value*><STYLE=*style-override*> ;
ITEM <*expression*> / <FORMAT=*format-name*>

<STYLE=*style-override*>
<VALUE=*integer-value*>;

END;

P *expression*/ <FORMAT=*format-name*> <STYLE=*style-override*> ;

TRANSLATE *expression-1* **INTO** *expression-2* < , *expression-n* INTO *expression-m*;>

PROC ODSTEXT Statement

Creates a text block.

Syntax

PROC ODSTEXT <CONTENTS=*text-string*><DATA=*SAS-data-set*> <NAME=*template-name*>
<PAGEBREAK=NO | YES> <PRINT><STORE=*template-store*>;

Optional Arguments

CONTENTS="*text-string*"
specifies the title for the table of contents. The CONTENTS= option overrides the generated table of contents text.

Tip *text-string* is the text that can be seen in the PDF table of contents and in the Results Window folder descriptions.

DATA=*SAS-data-set*
specifies a SAS data set.

Tip No output is produced if you specify the NAME= option without either the DATA= option or the PRINT option. Without the DATA= or PRINT options, ODS creates the template but does not render the output.

NAME=<*template-name*>
specifies that the content of the procedure should be saved as a template with the specified name.

Tip No output is produced if you specify the NAME= option without either the DATA= option or the PRINT option. Without the DATA= or PRINT options, ODS creates the template but does not render the output.

PAGEBREAK= NO | YES
specifies whether the procedure should generate a page break.

NO
specifies that no page break is generated.

Alias OFF

YES
specifies that a page break is generated.

Alias ON

Default NO

PRINT
specifies that the template is printed as well as stored.

Restriction The PRINT option is needed only if the NAME= option is specified without the DATA= option.

Tip No output is produced if you specify the NAME= option without either the DATA= option or the PRINT option. Without the DATA= or PRINT options, ODS creates the template but does not render the output.

STORE=*template-store*
specifies the template store where the compiled template is placed. If you do not specify a name or template store, ODS stores the template in the first writable template store in the ODS path. By default, this template store is Sasuser.Templat.

If you do specify a name and template store, but no libref is specified, then the template is stored in the Work directory. For example, the statement `proc odstext name=mylist store=mystore;` results in a template store named Work.Mystore.

Restriction The STORE= option can only be specified if the NAME= option is specified.

BY Statement

Orders the output according to the BY groups.

Note: BY-Group processing is included in release 2020.1.2 and later.

Syntax

BY <DESCENDING> *variable-1*
<… <DESCENDING> *variable-n*>
<NOTSORTED>;

Required Argument

variable
specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then either the observations in the data set must be sorted by all

the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Optional Arguments

DESCENDING
specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED
specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way (for example, chronological order).

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

You cannot use the NOTSORTED option in a PROC SORT step.

Note: You cannot use the GROUPFORMAT option, which is available in the BY statement in a DATA step, in a BY statement in any PROC step.

Details

BY-Group Processing

Procedures create output for each BY group. For example, the elementary statistics procedures and the scoring procedures perform separate analyses for each BY group. The reporting and ODS procedures produce a report for each BY group.

Note: All Base SAS procedures except PROC PRINT process BY groups independently. PROC PRINT can report the number of observations in each BY group as well as the number of observations in all BY groups. Similarly, PROC PRINT can sum numeric variables in each BY group and across all BY groups.

You can use only one BY statement in each PROC step. When you use a BY statement, the procedure expects an input data set that is sorted by the order of the BY variables or one that has an appropriate index. If your input data set does not meet these criteria, then an error occurs. Either sort it with the SORT procedure or create an appropriate index on the BY variables.

Depending on the order of your data, you might need to use the NOTSORTED or DESCENDING option in the BY statement in the PROC step.

- For more information about the BY statement, see *SAS DATA Step Statements: Reference*.

- For more information about PROC SORT, see "SORT Procedure" in *Base SAS Procedures Guide*.
- For more information about BY-group processing in CAS, see ."Grouping and Ordering" in *SAS Cloud Analytic Services: DATA Step Programming*
- For more information about creating indexes, see "INDEX CREATE Statement" in *Base SAS Procedures Guide*.

Formatting BY-Variable Values

When a procedure is submitted with a BY statement, the following actions are taken with respect to processing of BY groups:

1. The procedure determines whether the data is sorted by the internal (unformatted) values of the BY variable(s).
2. The procedure determines whether a format has been applied to the BY variable(s). If the BY variable is numeric and has no user-applied format, then the BEST12. format is applied for the purpose of BY-group processing.
3. The procedure continues adding observations to the current BY group until both the internal and formatted values of the BY variable or variables change.

This process can have unexpected results if, for example, nonconsecutive internal BY values share the same formatted value. In this case, the formatted value is represented in different BY groups. Alternatively, if different consecutive internal BY values share the same formatted value, then these observations are grouped into the same BY group.

BY Variables That Have Different Lengths in Two Data Sets

When a procedure has a BY statement and two input data sets, one of the input data sets is called the primary data set and the other is called the secondary data set. The primary data set is usually, but not always, the DATA= data set. A BY statement always applies to the primary data set. The variables in the BY statement must appear in the primary data set.

Each procedure determines whether a BY statement applies to a secondary data set, and performs one of the following actions:

- The procedure might always apply the BY statement to the secondary data set. In this case, one or more variables in the BY statement must appear in the secondary data set.
- The procedure might never apply the BY statement to the secondary data set. In this case, the variables in the BY statement are not required in the secondary data set.
- The procedure might check whether the BY variables are in the secondary data set. If none of the BY variables are in the secondary data set, then BY processing does not occur for the secondary data set. If one or more of the BY variables are in the secondary data set, and they match the BY variables in the primary data set, then BY processing is done for the secondary data set. If some but not all BY variables are in the secondary data set, then the procedure might issue an error message and quit. Or, it might take some other action described in the documentation for that particular procedure.

If the BY statement is applied to the secondary data set, then each BY variable that exists on both the data sets must have the same type, character or numeric, in both data sets. The BY variables are required to have either the same formatted value or the same unformatted value. Formatted values match only if both the formatted lengths and the formatted values are the same. Unformatted values are not required to have the same length in order to match. The unformatted character values match if the unformatted values are the same after stripping the trailing blanks. The unformatted doubles match if they have the same value.

A secondary data set does not need to have all of the BY variables that are in the primary data set. A procedure can define a subset of the BY variables for the secondary data set. For example, if the primary data set has the BY variables A,B,C,D, then the procedure can define the following BY variables on the secondary data set:

- A
- A,B
- A,B,C
- A,B,C,D

If both the primary and secondary data sets have the same number of BY variables, and all the BY variables have the same byte lengths and format lengths, then either the unformatted values or the formatted values in the BY buffer (for all of the BY variables) have to match. If they do not match, then each variable is compared. The formatted values of each variable are compared first. The formatted lengths have to match, and the formatted values have to match. If the formatted lengths and values do not match, then the unformatted values are compared even if the byte lengths are different.

If corresponding character variable lengths differ, then the longer character variable can contain only trailing blanks for the extra characters. If the lengths of the character variables are different, then the values match as long as they are the same after stripping the trailing blanks. For example, ‘ABCD’ in the primary data set matches ‘ABCD ’ in the secondary data set. If the secondary data set contained ‘ABCDEF’, then they would not match.

Base SAS Procedures That Support the BY Statement

COMPARE	SORT (required)
CORR	STANDARD
FREQ	SUMMARY
MEANS	TABULATE
PRINT	TRANSPOSE
RANK	UNIVARIATE
REPORT (nonwindowing environment only)	ODSTEXT

ODSLIST	ODSTABLE

Note: In the SORT procedure, the BY statement specifies how to sort the data. In the other procedures, the BY statement specifies how the data is currently sorted.

See Also

"BY Statement" in *SAS DATA Step Statements: Reference*

CELLSTYLE AS Statement

For tables, sets the style element of the cells in the table or column according to the values of the variables. For text, sets the style attributes of the list items or paragraphs. Use this statement to set the presentation characteristics (such as foreground color and font face) of individual cells or text.

Restriction: The CELLSTYLE AS statement can be used only within a column template, an ODS list, an ODS textblock, or a table template.

Syntax

CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*]> <, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*] >>;

Required Arguments

expression
is an expression that is evaluated for each list item, paragraph, or table cell.

If *expression* resolves to TRUE (a nonzero value), the style element that is specified is used for the current cell. If *expression* is FALSE, the next expression in the statement is evaluated. Thus, you can string multiple expressions together to format cells conditionally.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

An operator is a symbol that requests a string, a comparison, logical operation, or arithmetic calculation. An operand is one of the following:

constant
is a fixed value such as the name of a column or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in-variable
is a special type of WHERE expression operand that helps you find common values in table or column templates.

Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

DATANAME
is a data column name.

DATATYPE
is the data type of the column variable. The data type is either numeric ("num") or character ("char").

Example The following CELLSTYLE AS statement specifies that numeric column variables have a red font color and character column variables have a blue font color:

```
cellstyle   _datatype_ = "num" as {color=red},
            _datatype_ = "char" as {color=blue};
```

LABEL
is a column label.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

ROW
is a row number. Row numbering begins with 1.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

STYLE
is a style element name.

See For a table of style element names, see Chapter 20, "Style Elements," on page 777.

Example "Example 6: Creating Top-Level Templates" on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example “Example 6: Creating Top-Level Templates” on page 577

comparison-operator
compares a variable with a value or with another variable.

The following table lists the comparison operators:

Table 9.1 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one or more from a list of values

Tip Using an expression of 1 as the last expression in the CELLSTYLE AS statement sets the style element for any cells that did not meet an earlier condition. For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer’s Guide: Essentials*.

Example “Example 5: Setting the Style Element for a Specific Column, Row, and Cell” on page 570

style-attribute-specification
describes a style attribute to set.

Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

For information about the style attributes that you can set in a table template, see “About Style Attributes” on page 465.

Optional Argument

style-element-name
is the name of a style element that is part of a style that is registered with the Output Delivery System.

SAS provides some styles. You can create customized styles and style elements with PROC TEMPLATE by using the "DEFINE STYLE Statement" on page 454. For a table of style element names, see Chapter 20, "Style Elements," on page 777.

The following style elements are frequently used with the CELLSTYLE AS statement:

- Data
- DataFixed
- DataEmpty
- DataEmphasis
- DataEmphasisFixed
- DataStrong
- DataStrongFixed
- ListItem
- ListItem2
- Paragraph

The style element provides the basis for displaying the cell. Additional style attributes modify the display.

Default Data

See Chapter 14, "TEMPLATE Procedure," on page 433

For a table of style element names, see Chapter 20, "Style Elements," on page 777.

DYNAMIC Statement

Defines a symbol that references a value that the data component supplies from the procedure or DATA step.

Restriction: The DYNAMIC statement can be used only in the template of an ODS textblock, ODS list, table, column, header, footer, or statistical graph template. A dynamic variable that is defined in a template is available to that template and to all the templates that it contains.

Syntax

DYNAMIC *variable-name-1 <=value–1<'text-1'>>*
< variable-name-2 <=value-2><'text-2'…>>;

Required Argument

variable-name
names a variable that the data component supplies. ODS resolves the value of the variable when it binds the template and the data component.

Tip Dynamic variables are most useful to the authors of SAS procedures and to DATA step programmers.

Optional Arguments

value
sets the value of the variable.

text
is text that is placed in the template to explain the dynamic variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

END Statement

Ends item and list blocks.

Syntax

END;

HEADING Statement

Creates H# tags for the ODSTEXT output.

Alias: H

Accessibility note: Adding headings to your output helps make the output accessible to screen reader users and to people with certain visual impairments. For an example, see *Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics*.

Syntax

HEADING <*n*> “*heading-text*” / <STYLE=*style-override*>;

Required Argument

heading-text
specifies the heading text. You can use in-line formatting and style overrides to customize your heading text. This option helps to make reports accessible for people with a wide range of abilities.

See For information about in-line formatting, see "ODS ESCAPECHAR Statement" in *SAS Output Delivery System: User's Guide*.

For information about style overrides, see "STYLE= <*style-element-name*> [*style-attribute-name=style-attribute-value* <… *style-attribute-name=style-attribute-value*>]" on page 301.

Optional Argument

n
specifies the heading level.

Range 1–6

Example The following statement creates a second-level heading:

```
h2 "My Heading 2";
```

Options to Use after a Slash Delimiter

STYLE= <*style-element-name*> [*style-attribute-name=style-attribute-value* <… *style-attribute-name=style-attribute-value*>]
specifies the style element to use for the heading text.

style-element-name
is the name of a style element that is part of a style template that is registered with the Output Delivery System. SAS provides some style templates. You can create your own style templates with PROC TEMPLATE.

See Chapter 14, "TEMPLATE Procedure," on page 433 for information about PROC TEMPLATE and the default style templates.

For a list of style elements, see Chapter 20, "Style Elements," on page 777.

style-attribute-name
specifies the attribute to change.

See For information about style attributes and their values, see Chapter 21, "Style Attributes," on page 807.

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values.

See For information about style attributes and their values, see Chapter 21, "Style Attributes," on page 807.

Note You can use braces ({ and }) instead of square brackets ([and]).

Example For example, the following statement specifies that the text color for the third heading is red:

```
H3 'Third Heading' / style=[color=red];
```

ITEM Statement

Specifies an item to be added to the item list. An ITEM statement that does not specify an expression begins an item block.

Restrictions: The ITEM statement must be used within a LIST block.

ITEM statements that do not specify an *expression* must end with an END statement.

Example: "Example 2: Comparing Lists Created with PROC ODSLIST and PROC ODSTEXT" on page 317

Syntax

ITEM <*expression*> / <FORMAT=*format-name*> <STYLE=*style-override*> <VALUE=*integer-value*>;

Optional Argument

expression
is an expression that specifies the content of an item.

expression has this form:

expression-1 < *expression-n*>

expression
is an arithmetic or logical sequence of operators and operands. An operator is a symbol that requests a logical operation, a string, or an arithmetic calculation.

An operand is one of the following:

constant
is a fixed value, such as the name of a column, a text string, or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

Restriction ITEM statements that do not specify an *expression* must end with an END statement.

Options to Use after a Slash Delimiter

FORMAT=*format-name*
specifies a default format for the value in each table cell or list item. You can use any SAS or user-defined format.

Example
```
proc odstext data=sashelp.class;
   list;
      item age / format=2.;
   end;
run;
```

STYLE=<*style-element-name* >[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]
specifies the style element to use for the specified item.

style-element-name
is the name of a style element that is part of a style template that is registered with the Output Delivery System. SAS provides some style templates. You can create your own style templates with PROC TEMPLATE.

See Chapter 14, “TEMPLATE Procedure,” on page 433 for information about PROC TEMPLATE and the default style templates.

For a list of style elements, see Chapter 20, “Style Elements,” on page 777.

style-attribute-name
specifies the attribute to change.

See For information about style attributes and their values, see Chapter 21, “Style Attributes,” on page 807.

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values.

See For information about style attributes and their values, see Chapter 21, “Style Attributes,” on page 807.

Note You can use braces ({ and }) instead of square brackets ([and]).

Example The following statement specifies that the text color for the item is red:
```
item 'first item' / style=[color=red];
```

VALUE=*integer-value*
starts a numbered list from the specified *integer-value*. If another VALUE= option is specified in subsequent statements, numbering begins again at that point.

Restriction When VALUE= is specified, you must also specify a numbering list type with the LISTSTYLETPYE= style attribute. For example, LISTSTYLETYPE= “decimal_leading_zero” or LISTSTYLETYPE=“decimal”.

Tip You can change the bullet type of a list item by specifying the “LISTSTYLETYPE=bullet-type” style attribute with the STYLE= option.

Examples The following example creates a list with three items that are numbered 7, 10, and 4.

```
proc odstext;
   p 'This is a numbered list';
   list / style={liststyletype="decimal_leading_zero"};
      item "Seven" / value=7;
      item "Ten" / value=10;
      item "Four" / value=4;
   end;
run;
```

The following example creates a list that begins numbering items with the number 7.

```
proc odstext;
   p 'This is a numbered list';
   list / style={liststyletype="decimal_leading_zero"};
      item "Seven" / value=7;
      item "Eight";
   end;
run;
```

Example “Example 2: Creating Nested Lists” on page 226

LIST Statement

Creates a list.

Restriction: LIST statements begin a LIST statement block, and must end with an END statement.

Example: “Example 2: Comparing Lists Created with PROC ODSLIST and PROC ODSTEXT” on page 317

Syntax

LIST / <START=*integer-value*><STYLE=*style-override*> ;

Optional Arguments

START=*integer-value*
starts a numbered list from the specified *integer-value*. If another START= option is specified in subsequent statements, the numbering begins again at that point.

Restrictions When START= is specified, you must also specify a numbering list type with the LISTSTYLETPYE= style attribute. For example, LISTSTYLETYPE= "decimal_leading_zero" or LISTSTYLETYPE="decimal" are numbering list types.

The START= option is valid for the PDF, POWERPOINT, HTML, and RTF destinations.

Tip You can change the bullet type of a list item by specifying the "LISTSTYLETYPE=bullet-type" style attribute with the STYLE= option.

Examples The following example creates a list that begins numbering items with the number 7. For HTML5 output, if you want to start a numbered list from a specific number with the START= option, then specify LISTSTYLETYPE= in the LIST statement, as this example does.

```
proc odstext;
   p 'This is a numbered list';
   list / style={liststyletype="decimal_leading_zero"} start=7;
      item "Seven";
      item "Eight";
   end;
run;
```

The following example creates a list that assigns a specific number to each item.

```
proc odstext;
   p 'This is a numbered list';
   list / style={liststyletype="decimal_leading_zero"};
      item "Two" / value=2;
      item "Four" / value=4;
      item "Six" / value=6;
   end;
run;
```

STYLE=<*style-element-name* >[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]

specifies the style element to use for the items in the list.

For example, the following statement specifies that the text color for item is red:

```
item 'first item' / style=[color=red];
```

Note: You can use braces ({ and }) instead of square brackets ([and]).

style-element-name

is the name of a style element that is part of a style template that is registered with the Output Delivery System. SAS provides some style templates. You can create your own style templates with PROC TEMPLATE.

See Chapter 14, "TEMPLATE Procedure," on page 433 for information about PROC TEMPLATE and the default style templates.

For a list of style elements, see Chapter 20, "Style Elements," on page 777.

style-attribute-name

specifies the attribute to change.

See For information about style attributes and their values, see Chapter 21, "Style Attributes," on page 807.

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values.

See For information about style attributes and their values, see Chapter 21, "Style Attributes," on page 807.

MVAR Statement

Defines a symbol that references a macro variable. ODS uses the value of the variable as a string. References to the macro variable are resolved when ODS binds the template and the data component to produce an output object.

Restriction: When replaying an ODS document with PROC DOCUMENT, values created by the MVAR statement must be re-created in the same session that is replaying the document.

Tip: You can use the MVAR statement in the template of an ODS list, ODS textblock, table, column, header, or footer. A macro variable that is defined in a template is available to that template and to all the templates that it contains.

See: "Example 3: Creating a New Table Template " on page 557 and "Example 1: Creating a Stand-Alone Style" on page 466

Syntax

MVAR *variable-name-1* <='*value-1*' <'*text-1*'>>
< *variable-name-2* <='*value-2*'> <'*text-2*'…>>;

Required Argument

variable-name
names a macro variable to reference in the template. ODS uses the value of the macro variable as a string. ODS does not resolve the value of the macro variable until it binds the template and the data component.

Tip Declare macro variables this way in a template. For example, to use the automatic macro variable SYSDATE9 in a template, declare it in an MVAR statement and reference it as SYSDATE9, without an ampersand, in the PROC TEMPLATE or PROC ODSTABLE step. If you use the ampersand, the macro variable resolves when the template is compiled instead of when ODS binds the template to the data component.

Optional Arguments

value
sets the default variable value.

text
is text that is placed in the template to explain the macro variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

NMVAR Statement

Defines a symbol that references a macro variable. ODS converts the variable's value to a number (stored as a double) before using it. References to the macro variable are resolved when ODS binds the template and the data component to produce an output object.

Restriction: The NMVAR statement can be used only in the template of an ODS list, ODS textblock, table, column, header, or footer. A macro variable that is defined in a template is available to that template and to all the templates that it contains.

See: "Example 4: Setting the Style Element for Cells Based on Their Values" on page 565

Syntax

NMVAR *variable-name-1* <='*value–1*' <'*text-1*'>>
< *variable-name-2* <='*value-2*'> <'*text-2*'…>>;

Required Argument

variable-name
names a macro variable to reference in the template. ODS converts the variable's value to a number (stored as a double) before using it. ODS does not resolve the macro variable until it binds the template and the data component.

Tip Declare macro variables this way in a template. For example, to use a macro variable as a number, declare it in an NMVAR statement and reference it without an ampersand. If you use the ampersand, the macro variable resolves when the template is compiled instead of when ODS binds the template to the data component.

Optional Arguments

value
sets the value of the variable.

text
is text that is placed in the template to explain the macro variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

P Statement

Specifies a paragraph. Multiple P statements are allowed within an item block.

Restriction: In the ODSLIST Procedure, the P statement can be specified only within an ITEM block. In the ODSTEXT procedure, the P statement can be used at the top level of PROC ODSTEXT as well as within list items.

Syntax

P *expression* / <FORMAT=*format-name*> <STYLE=*style-override*> ;

Required Argument

expression
is an expression that specifies the content of an item or paragraph.

expression has this form:

expression-1 < *expression-n*>

expression
is an arithmetic or logical sequence of operators and operands. An operator is a symbol that requests a logical operation or an arithmetic calculation.

An operand is one of the following:

constant
is a fixed value, such as the name of a column, text string, or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

Examples The following code creates a paragraph and an item list with PROC ODSTEXT:

```
proc odstext data=sashelp.class;
   p "My name is " || name;
   list;
     item;
       p "My age is " || put(age, 2.);
       p "My weight is " || put(weight, 3.);
     end;
   end;
run;
```

The following code creates an item list with PROC ODSLIST:

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

Restriction ITEM statements that do not specify an *expression* must end with an END statement.

Options to Use after a Slash Delimiter

FORMAT=*format-name*
specifies a default format for the value in each paragraph. You can use any SAS or user-defined format.

STYLE=<*style-element-name* >[*style-attribute-name=style-attribute-value*<… *style-attribute-name=style-attribute-value*>]
specifies the style element to use for the items in the list or paragraph.

style-element-name
is the name of a style element that is part of a style template that is registered with the Output Delivery System. SAS provides some style templates. You can create your own style templates with PROC TEMPLATE.

See Chapter 14, "TEMPLATE Procedure," on page 433 for information about PROC TEMPLATE and the default style templates.

For a list of style elements, see Chapter 20, "Style Elements," on page 777.

style-attribute-name
specifies the attribute to change.

See For information about style attributes and their values, see Chapter 21, "Style Attributes," on page 807.

style-attribute-value
specifies a value for the attribute. Each attribute has a different set of valid values.

See For information about style attributes and their values, see Chapter 21, "Style Attributes," on page 807.

Note You can use braces ({ and }) instead of square brackets ([and]).

Examples The following statement specifies that the text color of the paragraph is red:

```
p 'text block' / style=[color=red];
```

The following statement specifies that the text color for the item is red:

```
item 'first item' / style=[color=red];
```

TRANSLATE INTO Statement

Translates the specified numeric values to other values.

Restrictions: The TRANSLATE INTO statement can be used only in a column template, an ODS list, an ODS textblock, or a table template.

The TRANSLATE INTO statement in a table template applies only to numeric variables. To translate the values of a character variable, use TRANSLATE INTO in the template of that column.

Syntax

TRANSLATE *expression-1* **INTO** *expression-2* <, *expression-n* INTO *expression-m*>;

Required Argument

expression
is an expression that is evaluated for each list item, paragraph, table, or column cell that contains a numeric variable.

If *expression-1* resolves to TRUE (a nonzero value), the translation that is specified is used for the current cell. If *expression-1* is FALSE (zero), the next expression in the statement is evaluated. Thus, you can string multiple expressions together to format cells conditionally.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation.

An operand is one of the following:

constant
is a fixed value such as the name of a column or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in-variable
is a special type of WHERE expression operand that helps you find common values in table or column templates.

Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example “Example 5: Setting the Style Element for a Specific Column, Row, and Cell” on page 570

DATANAME
is a data column name.

DATATYPE
is the data type of the column variable. The data type is either numeric ("num") or character ("char").

LABEL
is a column label.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

ROW
is a row number. Row numbering begins with 1.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

STYLE
is a style element name.

Example "Example 6: Creating Top-Level Templates" on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example "Example 6: Creating Top-Level Templates" on page 577

comparison-operator
compares a variable with a value or with another variable.

The following table lists the comparison operators:

Table 9.2 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one or more from a list of values

Restriction You cannot reference the values of other columns in *expression-1*.

Tip Using an expression of 1 as the last expression in the TRANSLATE–INTO statement specifies a translation for any cells that did not meet an earlier condition.

See You can use any expression that can be used in the WHERE= data set option. For information about expressions that you can use in the WHERE data set option, see the section on WHERE-Expression Processing in *SAS Programmer's Guide: Essentials*.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

Usage: ODSTEXT Procedure

Using the ODSTEXT Procedure

PROC ODSTEXT uses many of the same statements as PROC TEMPLATE to create and customize your text. The CELLSTYLE AS, DYNAMIC, MVAR, NMVAR, and TRANSLATE INTO statements are all valid statements within PROC ODSTEXT that can be used to customize your text.

Changing the Bullet Type for Lists

By default, lists created by the ODS TEXT procedure are displayed as an unordered list with a disc as the bullet type. In markup family output, you can change the bullet type for lists by using the following two steps:

1. Specify the LISTSTYLETYPE= style attribute with the desired bullet type in a style override. The following shows the general syntax for specifying the LISTSTYLETYPE= attribute::

   STYLE={ LISTSTYLETYPE="*bullet-type*" <*more-style-attributes*>}

2. Specify the style override in an ITEM or LIST statement. The following shows the general syntax for specifying a style override in the LIST or ITEM statements:

   LIST / STYLE={ LISTSTYLETYPE="*bullet-type*" <*more-style-attributes*>}

   ITEM / STYLE={ LISTSTYLETYPE="*bullet-type*" <*more-style-attributes*>}

Note:

For HTML5 output, if you want to start a numbered list from a specific number with the START= option, then specify LISTSTYLETYPE= in the LIST statement.

IMPORTANT

The LISTSTYLETYPE= style attribute is valid in the Markup family, Excel, and PowerPoint destinations. The valid bullet-type values differ for each destination. For more information about valid bullet types, see the LISTSTYLETYPE= Style Attribute on page 861.

When you specify LISTSTYLETYPE= in the LIST statement, the bullets for all of the following items are changed. If you specify LISTSTYLETYPE= in an ITEM statement, only the bullet for that ITEM statement is changed.

The following example creates a numbered list. Because the style override is specified in the LIST statement, all of the items will have the same numbering scheme.

```
ods html5 file='entireList_html5.htm';
title;
proc odstext;
   p 'Follow these steps to change the bullet type for lists:'
                                        / style={fontsize=12pt};
   list / style={liststyletype="decimal" fontsize=12pt};
   item "Decide what bullet type to display.";
   item "Add a '/' after the LIST statement (if not already present).";
   item "Specify the LISTSTYLETYPE= attribute with the STYLE= option.";
end;
run;
ods _all_ close;
```

Output 9.1 *Changing the Bullet Type for All List Items*

Follow these steps to change the bullet type for lists:

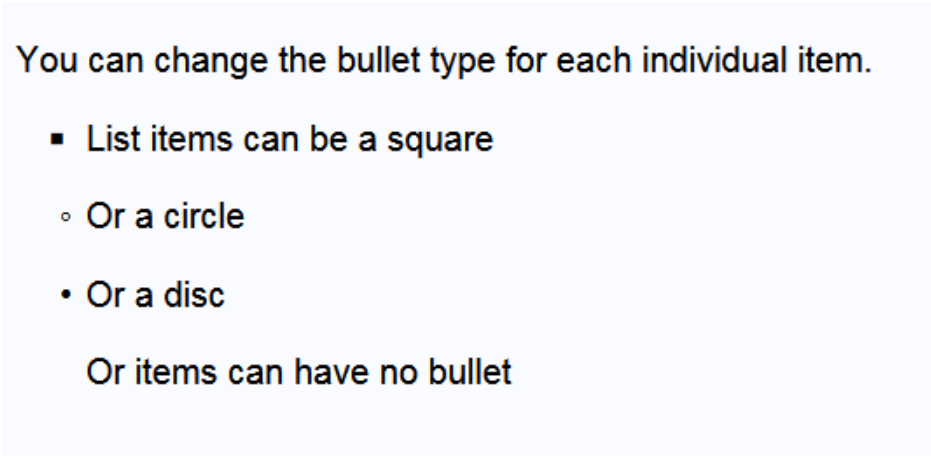
1. Decide what bullet type to display.
2. Add a '/' after the LIST statement (if not already present).
3. Specify the LISTSTYLETYPE= attribute with the STYLE= option.

You can also change the bullet type for each individual item by specifying a style override in each ITEM statement.

```
ods html5 file='individualItems_html5.htm';
title;
proc odstext;
   p 'You can change the bullet type for each individual item.'
                                        / style={fontsize=12pt};
   list / style={fontsize=12pt};
   item "List items can be a square" / style={liststyletype="square"};
   item "Or a circle" / style={liststyletype="circle"};
   item "Or a disc" / style={liststyletype="disc"};
   item "Or items can have no bullet" / style={liststyletype="none"};
end;
run;
```

```
ods _all_ close;
```

Output 9.2 *Changing the Bullet Type for Individual Items*



Here are the browser-supported bullet types. However, many browsers support other common types and the support is browser dependent. If an unsupported bullet type is specified, the bullet's display is browser dependent.

Table 9.3 *Valid Bullet Types*

Bullet Type	Description
	Ordered Lists
decimal	The list items are ordered with numbers (default).
upper_latin	The list items are ordered with uppercase letters.
lower_latin	The list items are ordered with lowercase letters.
upper_roman	The list items are ordered with uppercase roman numerals.
lower_roman	The list items are ordered with lowercase roman numerals.
	Unordered Lists
disc	The list items are marked with filled circles.
circle	The list items are marked with circles.
square	The list items are marked with squares.
none	The list items are not marked.

Examples: ODSTEXT Procedure

Example 1: Creating Content for the ODS Destination for PowerPoint

Features:

P statement
    STYLE= option
PROC ODSTEXT statement
    P statement
FOOTNOTE statement
ODS POWERPOINT statement
OPTIONS statement
TITLE statement

Details

The following example creates text output to add to a Microsoft PowerPoint slide. You can use the STYLE= option in the P statement to customize your text.

Program

```
ods html close;
options nodate;
title 'Using PROC ODSTEXT';
footnote 'The ODS Destination for PowerPoint';

ods powerpoint file="file-path/layoutTwocontent.ppt" layout=twocontent;


proc odstext;
   p 'You can use the ODSTEXT procedure to add paragraphs
      and lists to your output.';
   p 'You can also format your text.' / style=[color=red
fontsize=25pt];
   p 'This slide shows output created by PROC GMAP.'
                                      / style=[color=purple
fontsize=30pt];
run;
```

```
proc gmap map=maps.us data=maps.us all;
   id state;
   choro state/statistic=frequency discrete;
run;
quit;

ods _all_ close;
```

Program Description

Close the HTML destination, specify titles, and specify footnotes.

```
ods html close;
options nodate;
title 'Using PROC ODSTEXT';
footnote 'The ODS Destination for PowerPoint';
```

Create a file for the ODS destination for PowerPoint. The ODS POWERPOINT statement creates output formatted for the ODS destination for PowerPoint.

```
ods powerpoint file="file-path/layoutTwocontent.ppt" layout=twocontent;
```

Create content for the ODS destination for PowerPoint.

```
proc odstext;
   p 'You can use the ODSTEXT procedure to add paragraphs
      and lists to your output.';
   p 'You can also format your text.' / style=[color=red
fontsize=25pt];
   p 'This slide shows output created by PROC GMAP.'
                                       / style=[color=purple
fontsize=30pt];
run;
```

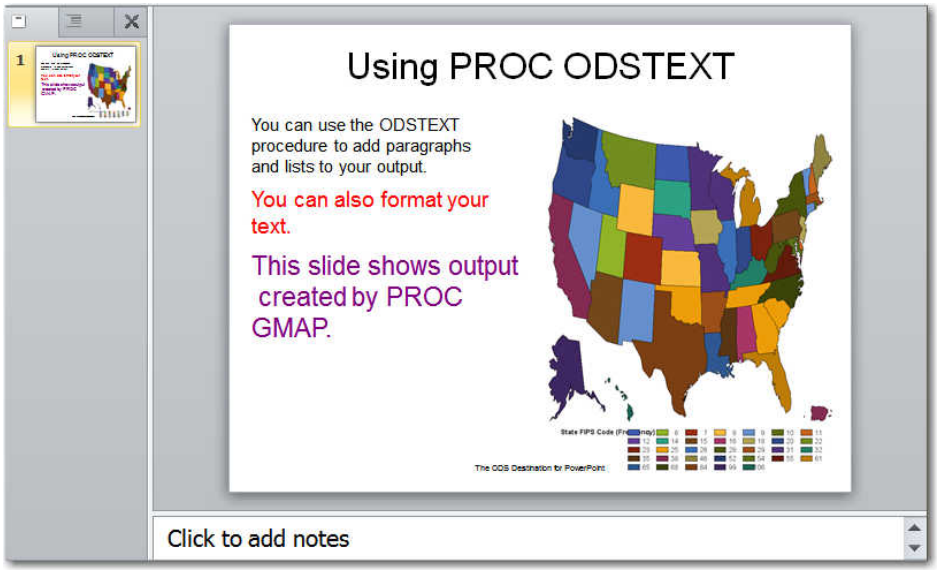
Create PROC GMAP output and close open destinations.

```
proc gmap map=maps.us data=maps.us all;
   id state;
   choro state/statistic=frequency discrete;
run;
quit;

ods _all_ close;
```

Adding Text to Your Output

Output 9.3 *Adding Text to Your Output*



Example 2: Comparing Lists Created with PROC ODSLIST and PROC ODSTEXT

Features:

ODSLIST Procedure
- CELLSTYLE AS statement
- END statement
- ITEM statement
- LIST statement
- P statement:
- PROC ODSLIST statement
- TRANSLATE INTO statement

ODSTEXT Procedure
- CELLSTYLE AS statement
- END statement
- ITEM statement
- LIST statement
- P statement
- PROC ODSTEXT statement
- TRANSLATE INTO statement

OPTIONS statement

TITLE statement

Details

The following examples create similar output using PROC ODSLIST and PROC ODSTEXT. Both procedures can be used to create lists and paragraphs. However, there are differences in how the syntax of the two procedures can be specified. The following table shows how the P, ITEM, and LIST statements are specified in each procedure.

PROC ODSTEXT	PROC ODSLIST
The P statement can be used at the top level of PROC ODSTEXT as well as within list items.	The P statement can be specified only within an ITEM block.
The ITEM statement must be used within a LIST block.	The ITEM statement does not have to be specified within a LIST block.
The LIST statement does not have to be specified within an item block.	The LIST statement must be used within an item block.

There are also slight differences in the appearance of the output created by the two procedures. Examine the output created by the two programs in this example. Notice that the spacing and bullet points are slightly different.

Program: Using PROC ODSLIST

```
options nodate nonumber obs=7;
title "Students' Names, Ages, and Weights";


proc odslist data=sashelp.class;
   item;
      p 'List of Students'' Names, Ages, and Weights' /
style=systemtitle;
      p 'Weights of children younger than fourteen are hidden for
legal reasons'
        / style={fontstyle=italic};

      list;
         cellstyle age<=13 as datastrong{background=green},
                   age>=14 as {background=lightpurple};

         item 'Student name is ' || name;
         item 'Age: ' || put(age, 2.);

         item;
            translate age<=13 into 'Weight: N/A';
            p 'Weight: ' || put(weight, 3.);
         end;
```

```
      end;
   end;
run;
```

Program Description

Specify the SAS system options and title.

```
options nodate nonumber obs=7;
title "Students' Names, Ages, and Weights";
```

Begin the ODSLIST procedure, begin an ITEM block, and specify explanatory text. The PROC ODSLIST statement specifies the input data set and begins the procedure. The ITEM statement with no options specified begins an ITEM block. The P statements specify the explanatory text. In the ODSLIST Procedure, the P statement must be specified within an ITEM block.

```
proc odslist data=sashelp.class;
   item;
      p 'List of Students'' Names, Ages, and Weights' /
style=systemtitle;
      p 'Weights of children younger than fourteen are hidden for
legal reasons'
        / style={fontstyle=italic};
```

Begin the list and conditionally set the style attributes. The LIST statement with no options specified begins a list block. The LIST statement must be used within an item block. The CELLSTYLE AS statement sets the style attributes of the list items based on the value of the variable Age. If the value of the variable Age is less than or equal to 13, the background color is green. If the value of the variable Age is greater than or equal to 14, the background color is light purple.

```
      list;
         cellstyle age<=13 as datastrong{background=green},
                   age>=14 as {background=lightpurple};
```

Specify the items. The ITEM statements specify the content of each item.

```
         item 'Student name is ' || name;
         item 'Age: ' || put(age, 2.);
```

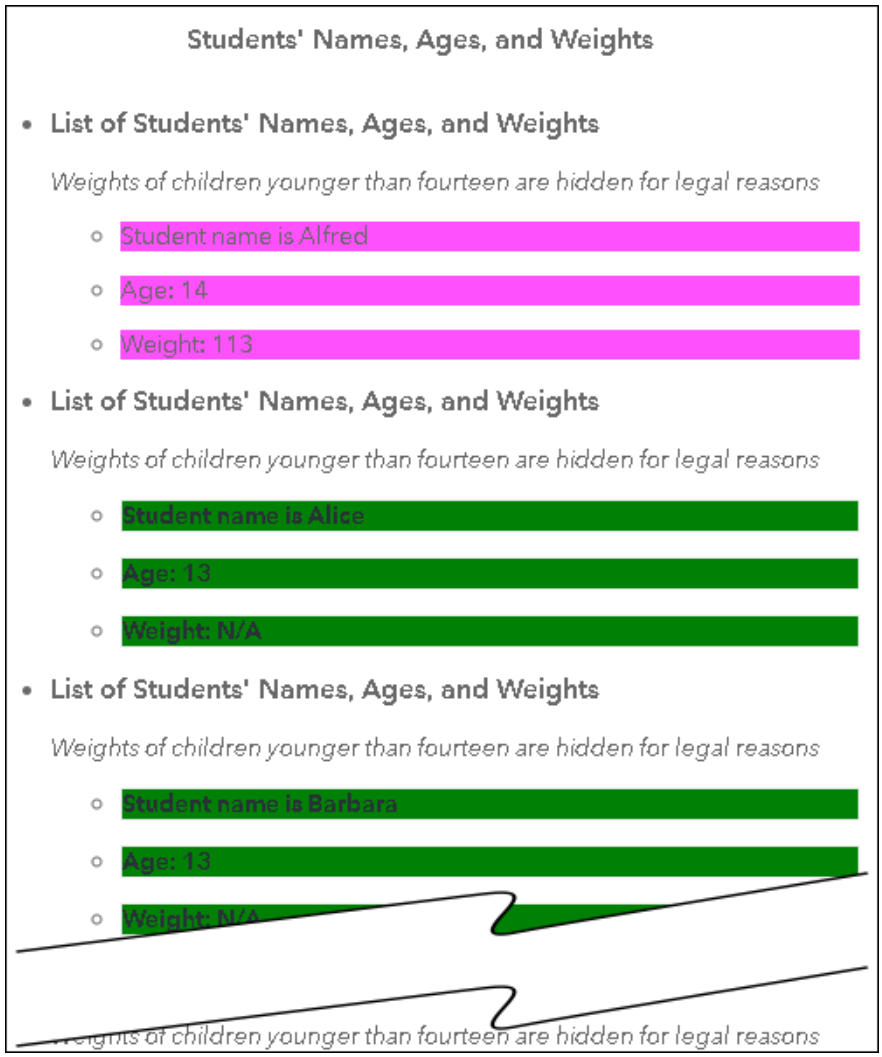
Conditionally transform the value for the AGE variable. The TRANSLATE INTO statement translates the value of Age based on the conditions specified. For values of the variable AGE that are equal to or less than thirteen, "N/A" is written to the output.

```
         item;
            translate age<=13 into 'Weight: N/A';
            p 'Weight: ' || put(weight, 3.);
         end;
```

End the LIST and ITEM blocks. The LIST block began with the LIST statement. An END statement ends the LIST block. Both ITEM blocks began with an ITEM statement with no expression specified. An END statement ends each ITEM block.

```
      end;
   end;
run;
```

Output 9.4 *Output Created with the PROC ODSLIST*



Program: PROC ODSTEXT

```
options nodate nonumber obs=7;
title "Students' Names, Ages, and Weights";

proc odstext data=sashelp.class;
   p 'List of Students'' Names, Ages, and Weights' / style=systemtitle;
   p 'Weights of children younger than fourteen are hidden for legal
reasons'
     / style={fontstyle=italic};

   list;
      cellstyle age<=13 as datastrong{background=green},
                age>=14 as {background=lightpurple};

      item 'Student name is ' || name;
      item 'Age: ' || put(age, 2.);

      item;
         translate age<=13 into 'Weight: N/A';
```

```
         p 'Weight: ' || put(weight, 3.);
      end;

   end;
run;
```

Program Description

Specify the SAS system options and title.

```
options nodate nonumber obs=7;
title "Students' Names, Ages, and Weights";
```

Begin the ODSTEXT procedure and specify explanatory text for the list. The PROC ODSLIST statement specifies the input data set and begins the procedure. The P statements specify the text that precedes the list. In the ODSTEXT procedure, the P statement does not have to be specified within an ITEM block.

```
proc odstext data=sashelp.class;
   p 'List of Students'' Names, Ages, and Weights' / style=systemtitle;
   p 'Weights of children younger than fourteen are hidden for legal
reasons'
     / style={fontstyle=italic};
```

Begin the list and conditionally set the style attributes. The LIST statement with no options specified begins a list block. In the ODSTEXT procedure, the LIST statement does not have to be specified within an item block. The CELLSTYLE AS statement sets the style attributes of the list items based on the value of the variable Age. If the value of the variable Age is less than or equal to 13, the background color is green. If the value of the variable Age is greater than or equal to 14, the background color is light purple.

```
      list;
         cellstyle age<=13 as datastrong{background=green},
                   age>=14 as {background=lightpurple};
```

Specify the items. The ITEM statements specify the content of each item.

```
         item 'Student name is ' || name;
         item 'Age: ' || put(age, 2.);
```

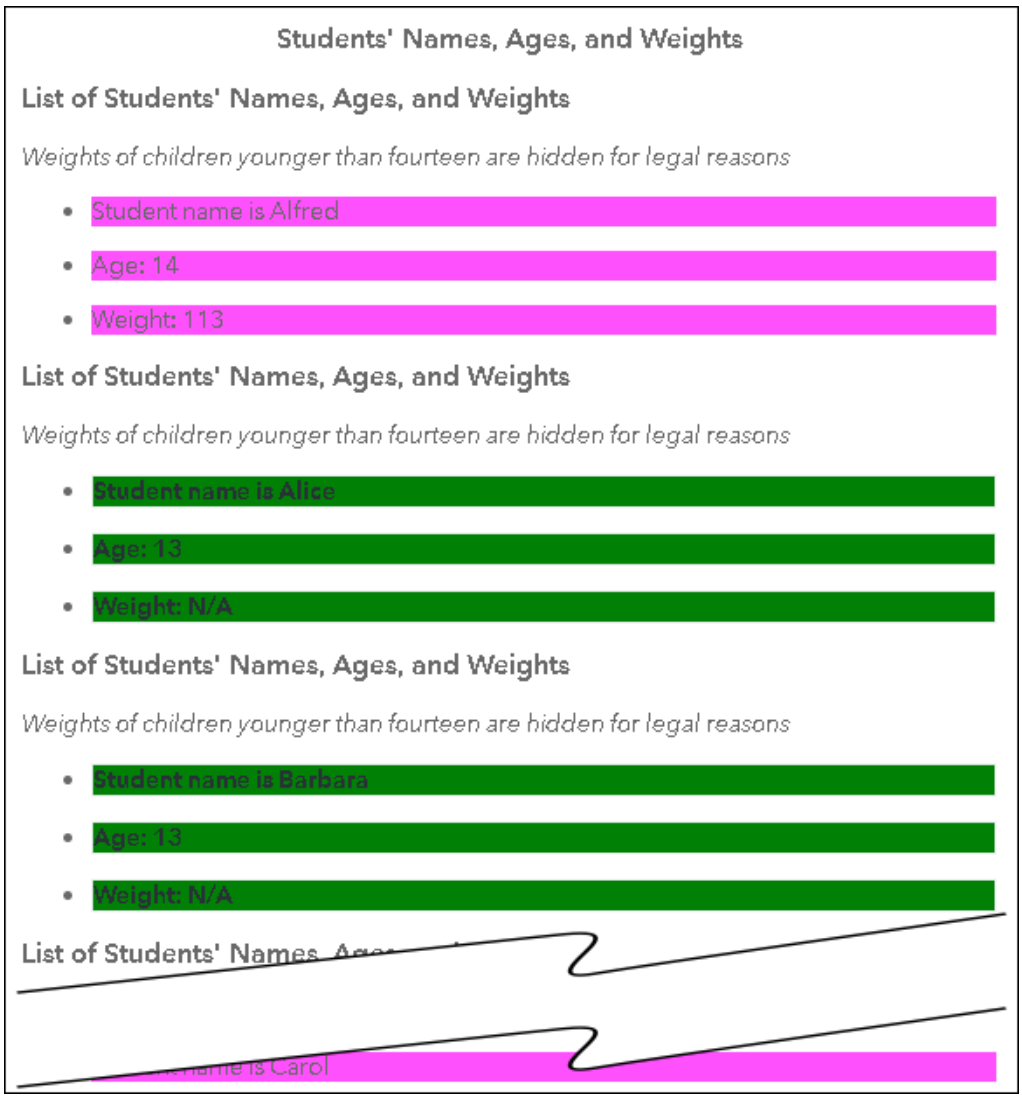
Conditionally transform the value for the AGE variable. The TRANSLATE INTO statement translates the value of Age based on the conditions specified. For values of the variable AGE that are equal to or less than thirteen, "N/A" is written to the output. The TRANSLATE INTO statement must be specified within the block of the variable you want to be translated. Otherwise, the variable is translated and the results apply to all variables.

```
         item;
            translate age<=13 into 'Weight: N/A';
            p 'Weight: ' || put(weight, 3.);
         end;
```

End the LIST block. The LIST block began with the LIST statement. An END statement ends the LIST block.

```
      end;
run;
```

Output 9.5 *Output Created with the PROC ODSTEXT*



PART 7

The TEMPLATE Procedure

Chapter 10
Overview **325**

Chapter 11
TEMPLATE Procedure: Managing Template Stores **339**

Chapter 12
TEMPLATE Procedure: Creating Crosstabulation Table Templates **365**

Chapter 13
TEMPLATE Procedure: Creating ODS Graphics **427**

Chapter 14
TEMPLATE Procedure: Creating a Style Template **433**

Chapter 15
TEMPLATE Procedure: Creating Tabular Templates **515**

Chapter 16
Tabular Attributes **585**

Chapter 17
TEMPLATE Procedure: Creating Markup Language Tagsets **627**

10

Overview

Introduction to the TEMPLATE Procedure 325
Terminology: TEMPLATE Procedure 326
The Backward Compatibility of ODS Templates 327
Syntax 328
Using the TEMPLATE Procedure 329
What Can You Do with the TEMPLATE Procedure? 329
PROC TEMPLATE Statements by Category 336
Where to Go from Here 337

Introduction to the TEMPLATE Procedure

The TEMPLATE procedure enables you to customize the appearance of your SAS output. For example, you can create, extend, or modify existing templates for various types of output:

- styles
- tables
- crosstabulation tables
- columns
- headers
- footers
- tagsets
- ODS Graphics

ODS then uses these templates to produce formatted output.

You can also use the TEMPLATE procedure to navigate and manage the templates stored in template stores. Here are some tasks that you can do with PROC TEMPLATE:

- edit an existing template
- create links to an existing template
- change the location where you write new templates
- search for existing templates
- view the source code of a template

Terminology: TEMPLATE Procedure

These terms frequently appear in discussions of PROC TEMPLATE:

aggregate storage location
: is a location on an operating system that can contain a group of distinct files. Different host operating systems call an aggregate grouping of files different names, such as a directory, a maclib, or a partitioned data set. The standard form for referencing an aggregate storage location from within SAS is *fileref(name)*, where *fileref* is the entire aggregate and *(name)* is a specific file or member of that aggregate.

event
: specifies the text that the markup destination produces when the specified event occurs. For example, the template of an event called ROW might specify to place the appropriate tags for starting a row at the beginning of an event and the appropriate tags for ending a row at the end of the event. SAS procedures that generate ODS output use a standard set of events, which you can customize with the TEMPLATE procedure.

graph template
: describes the contents and structure of a single-cell or multi-cell graph.

item store
: is a member of a SAS library. An item store is a hierarchical file system that is implemented as a single physical file. An item store can contain directories and files (called items) similar to the file systems in the UNIX and Windows operating environments. An item store is referenced by a two-level name: a libref and the name of the item store in the SAS library that the libref references. For example, the SAS registry is stored in two items stores, Sasuser.Registry and Sashelp.Registry.

style (template)
: describes how to display the presentation aspects (color, font face, font size, and so on) of your SAS output. A style determines the overall appearance of the documents that use it. Each style consists of style elements.

style element
: is a collection of style attributes that apply to a particular part of the output. For example, a style element can contain instructions for the presentation of column headings or for the presentation of the data inside cells. Style elements can also specify default colors and fonts for output that uses the style. Each style attribute specifies a value for one aspect of the presentation. For example, the

BACKGROUND= attribute specifies the color for the background of an HTML table, and the FONTSTYLE= attribute specifies whether to use a Roman, a slant, or an italic font.

table template
: describes how to display the output for a tabular output object. (Most ODS output is tabular.) A table template determines the order of table headers and footers, the order of columns, and the overall appearance of the output object that uses it. Each table template contains or references table elements.

table element
: is a collection of attributes that apply to a particular column, header, or footer. Typically, these attributes specify something about the data rather than about its presentation. For example, FORMAT= specifies the SAS format to use in a column. However, some attributes describe presentation aspects of the data.

Note: You can also define table elements such as columns, headers, and footers outside of a table template. Any table template can then reference these table elements. For more information about defining columns, headers, and footers outside of the table template, see Chapter 15, "TEMPLATE Procedure," on page 515.

tagset
: specifies instructions for creating a markup language for your SAS output. The resulting output contains embedded instructions in order to define layout and some content. Each tagset contains event templates and event attributes that control the generation of the output. SAS provides tagsets for a variety of markup languages. With the TEMPLATE procedure, you can modify any of these SAS tagsets, or you can create your own tagsets.

template store
: is an item store that stores templates that were created by the TEMPLATE procedure. Templates that SAS provides are in the item store Sashelp.Tmplmst. You can store templates that you create in any template store where you have Write access.

Note: A template store can contain multiple levels known as directories. When you specify a template store in the ODS PATH statement, however, you specify a two-level name that includes a libref and the name of a template store in the SAS library that the libref references.

The Backward Compatibility of ODS Templates

ODS templates are not binary compatible between SAS versions. However, with some templates, you can use a template created with an earlier version of SAS with a later version of SAS. The following table lists the ODS templates and whether they are forward or backward compatible between SAS versions.

Table 10.1 *Compatibility of ODS Templates between SAS Versions*

ODS Template	Backward Compatible	Forward Compatible
Table	No	Yes
Crosstabs	No	Yes
Style	No	Yes *
Tagset	No	No
ODS Graphics	No	No

* Styles that use inheritance might not be compatible forwards or backward.

If you would like to use a template created with a later version of SAS with an earlier version of SAS, you might be able to extract the template source and use it to compile the template in the earlier release.

Syntax

```
PROC TEMPLATE;
   DEFINE COLUMN column-path </ STORE=libref.template-store>;
      <column-attribute-1; <...column-attribute-n;>>
      statements
      END;
   DEFINE FOOTER footer-path </ STORE=libref.template-store>;
      <footer-attribute-1; <...footer-attribute-n;>>
      statements
      END;
   DEFINE HEADER template-name </ STORE=libref.template-store>;
      <header-attribute-1; <...header-attribute-n;>>
      statements
      END;
   DEFINE STYLE style-path </ STORE=libref.template-store>;
      <PARENT= style-path;>
      statements
      END;
   DEFINE TABLE table-path </ STORE=libref.template-store>;
      <table-attribute-1; <...table-attribute-n;>>
      statements
      END;
   DEFINE TAGSET tagset-path </ STORE=libref.template-store>;
```

```
    DEFINE EVENT event-name;
    <event-attribute-1; <...event-attribute-n;>>
    statements
    END;
DEFINE CROSSTABS table-path </ STORE=libref.template-store>;
    statements
    END;
DEFINE STATGRAPH graph-path </ STORE=libref.template-store>;
    statements
    END;
DELETE template-path </ STORE=libref.template-store >;
EDIT template-path-1 <AS template-path-2> </ STORE=libref.template-store > ;
    statements-and-attributes
    END;
LINK template-path-1 TO template-path-2 </ options>;
LIST <starting-path></ options>;
PATH location(s);
SOURCE template-path </ option(s)>;
TEST DATA=data-set </ STORE=libref.template-store>;
```

Using the TEMPLATE Procedure

What Can You Do with the TEMPLATE Procedure?

Modify a Table Template That a SAS Procedure Uses

This output shows the use of a customized table template for the Moments output object from PROC UNIVARIATE. The program used to create the modified table template does the following:

- creates and edits a copy of the default table template
- edits a header within the table template
- sets column attributes to enhance the appearance of the HTML output

For the code that creates the following default and customized output, see “Example 1: Editing a Table Template That a SAS Procedure Uses” on page 550.

Output 10.1 *Default Moments Table*

Default Moments Table

The UNIVARIATE Procedure
Variable: Quantity

Moments			
N	48	Sum Weights	48
Mean	20.5208333	Sum Observations	985
Std Deviation	14.4177633	Variance	207.871897
Skewness	3.6558946	Kurtosis	19.528114
Uncorrected SS	29983	Corrected SS	9769.97917
Coeff Variation	70.2591509	Std Error Mean	2.08102487

Output 10.2 *Customized HTML Output (Customized Moments Table) from PROC UNIVARIATE*

Custom Moments Table

The UNIVARIATE Procedure
Variable: Quantity

Moments			
N	*48*	*Sum Weights*	*48*
Mean	*20.5208333*	*Sum Observations*	*985*
Std Deviation	*14.4177633*	*Variance*	*207.871897*
Skewness	*3.6558946*	*Kurtosis*	*19.528114*
Uncorrected SS	*29983*	*Corrected SS*	*9769.97917*
Coeff Variation	*70.2591509*	*Std Error Mean*	*2.08102487*

Modify a Style

When you are working with styles, you are more likely to modify a style that SAS supplies than to write a completely new style. The following output uses the Styles.HTMLBlue template that SAS provides, but includes changes made to the style in order to customize the output's appearance.

In the contents file, the modified style makes changes to the following:

- the text of the header and the text that identifies the procedure that produced the output
- the colors for some parts of the text
- the font size for some parts of the text

- the spacing in the list of entries in the table of contents

In the body file, the modified style makes changes to the following:

- two of the colors in the color list. One of these colors is used as the foreground color for the table of contents, the byline, and column headings. The other is used for the foreground of many parts of the body file, including SAS titles and footnotes.
- the font size for titles and footnotes.
- the font style for headers.
- the presentation of the data in the table by changing attributes like cellspacing, rules, and borderwidth.

For the code that creates the following output, see “Example 3: Modifying the Default Style with the CLASS Statement” on page 482.

Figure 10.1 Customized HTML Output

MSRP by Make and Model (6 Cylinders Only)

Make=Acura

Make	Model	Cylinders	MSRP
Acura	MDX	6	$36,945.00
Acura	TL 4dr	6	$33,195.00
Acura	3.5 RL 4dr	6	$43,755.00
Acura	3.5 RL w/Navigation 4dr	6	$46,100.00
Acura	NSX coupe 2dr manual S	6	$89,765.00

Make=Audi

Make	Model	Cylinders	MSRP
Audi	A4 3.0 4dr	6	$31,840.00
Audi	A4 3.0 Quattro 4dr manual	6	$33,430.00
Audi	A4 3.0 Quattro 4dr auto	6	$34,480.00
Audi	A6 3.0 4dr	6	$36,640.00
Audi	A6 3.0 Quattro 4dr	6	$39,640.00
Audi	A4 3.0 convertible 2dr	6	$42,490.00
Audi	A4 3.0 Quattro convertible 2dr	6	$44,240.00
Audi	A6 2.7 Turbo Quattro 4dr	6	$42,840.00
Audi	TT 3.2 coupe 2dr (convertible)	6	$40,590.00
Audi	A6 3.0 Avant Quattro	6	$40,840.00

Create Your Own Tagset

Tagsets are used to create custom markup. You can create your own tagsets, extend existing tagsets, or modify a tagset that SAS supplies. This display shows the results from a new tagset **TAGSET.MYTAGS**.

To see the customized CHTML tagset, view the source from your web browser by selecting **View** ⇨ **Source** from your browser's toolbar.

Figure 10.2 *MYTAGS.CHTML Output*

These are my new colspecs

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

Create a Template-Based Graph

STATGRAPH templates are used to create output called ODS Graphics. For complete information, see *SAS Graph Template Language: User's Guide*.

The following code creates the STATGRAPH template MyGraphs.Regplot, which creates the following graph.

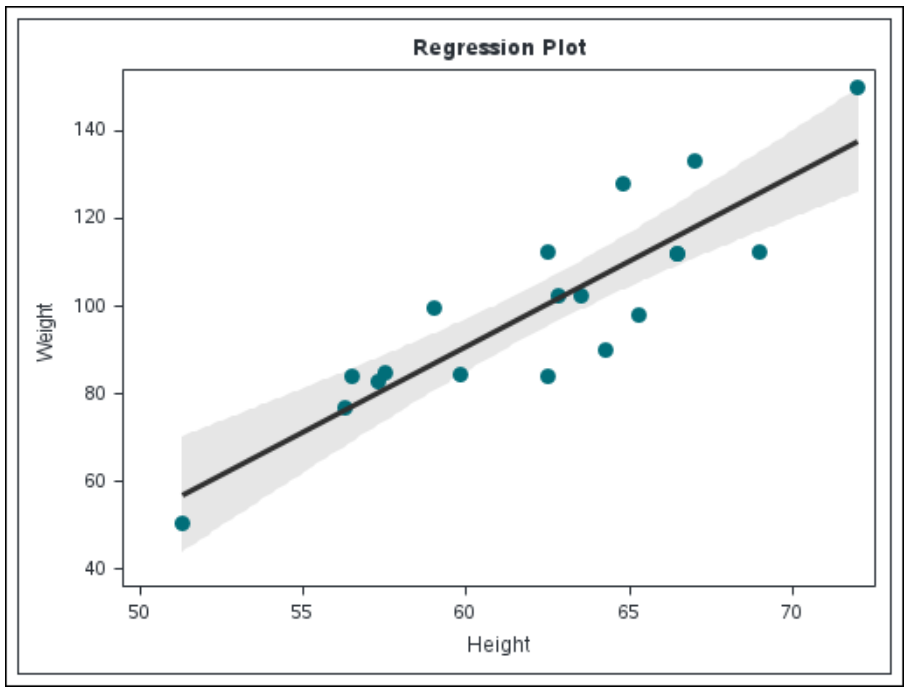
```
proc template;
define statgraph mygraphs.regplot;
```

```
begingraph;
  entrytitle "Regression Plot";
  layout overlay;
    modelband "mean";
    scatterplot x=height y=weight;
    regressionplot x=height y=weight / clm="mean";
  endlayout;
endgraph;
end;
run;
ods graphics / reset imagename="reg" width=500px;

proc sgrender data=sashelp.class template=mygraphs.regplot;
run;
```

The following display shows a scatter plot with an overlaid regression line and confidence limits of the mean for the HEIGHT and WEIGHT variables of a data set.

Figure 10.3 *Graph Created with a STATGRAPH Template*



Modify a Crosstabulation Table

The TEMPLATE procedure enables you to customize the appearance of crosstabulation (contingency) tables that are created with the FREQ procedure. By default, crosstabulation tables are formatted according to the CrossTabFreqs template that SAS provides. However, you can create a customized CrossTabFreqs table template by using the TEMPLATE procedure with the DEFINE CROSSTABS statement. For the SAS code that creates this output, see “Example 2: Creating a Crosstabulation Table Template with a Customized Legend” on page 409.

This output shows the use of a customized crosstabulation table template for the CrossTabFreqs table. The program used to create the modified crosstabulation table template does the following:

- modifies table regions
- customizes legend text
- modifies headers and footers
- modifies variable labels used in headers
- customizes styles for cellvalues

Figure 10.4 *Customized Crosstabulation Table Template for the CrossTabFreqs Table*

City Government Form by Number of Meetings Scheduled

Frequency Percent Row Percent Column Percent	Number of Meetings Scheduled						
City Government Form	?	Not Known	100 or Less	101-200	201-300	Over 300	Total
?	0 . . .	0 . . .	0 . . .	1 . . .	0 . . .	0 . . .	. .
Not Applicable	0 . . .	10 . . .	0 . . .	0 . . .	0 . . .	0 . . .	. .
Council Manager	0 . . .	0 . . .	47 12.30 22.27 55.95	63 16.49 29.86 58.88	49 12.83 23.22 62.03	52 13.61 24.64 46.43	211 55.24
Commission	0 . . .	0 . . .	6 1.57 28.57 7.14	7 1.83 33.33 6.54	3 0.79 14.29 3.80	5 1.31 23.81 4.46	21 5.50
Mayor Council	1 . . .	0 . . .	31 8.12 20.67 36.90	37 9.69 24.67 34.58	27 7.07 18.00 34.18	55 14.40 36.67 49.11	150 39.27
Total	. .	. .	84 21.99	107 28.01	79 20.68	112 29.32	**382** **100.00**
City Government Form by Number of Meetings Scheduled							
Frequency Missing = 12							

PROC TEMPLATE Statements by Category

This table lists and describes the categories and statements used in the TEMPLATE procedure.

Table 10.2 *Categories and PROC TEMPLATE Statements*

Task	Statements Category	Statements	Description
Navigate template stores and manage ODS templates	Template store	DELETE	Deletes the specified template
		LINK	Creates a link to an existing template
		LIST	Lists items in one or more template stores
		PATH	Specifies the locations to write to or read from when creating or using PROC TEMPLATE templates, and the order in which to search for them
		SOURCE	Writes the source code for the specified template
		TEST	Tests the most recently created template by binding it to the specified data set
Create or modify ODS style	Style	DEFINE STYLE	Creates a style for any destination that supports the STYLE= option
Create and modify ODS table, column,	Tabular	EDIT	Edits an existing template

Task	Statements Category	Statements	Description
header, and footer templates			
		DEFINE COLUMN	Creates a template for a column
		DEFINE FOOTER	Creates a template for a table footer
		DEFINE HEADER	Creates a template for a header
		DEFINE TABLE	Creates a template for a table
Create or modify markup language tagsets	Markup language tagsets	DEFINE TAGSET	Creates a template for a tagset
Create or modify a crosstabulation table template	Tabular	DEFINE CROSSTABS	Creates a template for a PROC FREQ crosstabulation table
Create or modify a graph template	Graphical	DEFINE STATGRAPH	Creates a template for a graph

Where to Go from Here

Creating statistical graphics with ODS:
For reference information about the Graph Template Language, see *SAS Graph Template Language: Reference*.

Creating statistical graphics with ODS:
For usage information about PROC TEMPLATE and the Graph Template Language, see *SAS Graph Template Language: User's Guide*.

Managing the various templates stored in template stores:
For reference information about the PROC TEMPLATE statements that help you manage and navigate around the many ODS templates, see Chapter 11, "TEMPLATE Procedure," on page 339.

Modifying an existing style or creating your own style:
For reference information about the style template statements in PROC TEMPLATE, see Chapter 14, "TEMPLATE Procedure," on page 433.

Creating and modifying ODS tabular output:
For reference information about the tabular template statements in PROC TEMPLATE, see Chapter 15, “TEMPLATE Procedure,” on page 515.

Modifying markup language tagsets that SAS provides or creating your own tagsets:
For reference information about the markup language tagset statements in PROC TEMPLATE, see Chapter 17, “TEMPLATE Procedure,” on page 627.

Chapter 11

TEMPLATE Procedure: Managing Template Stores

Overview: TEMPLATE Procedure: Managing Template Stores ***339***
What Is a Template Store? 340
Why Use the TEMPLATE Procedure to Manage Template Stores? 340

Concepts: TEMPLATE Procedure: Managing Template Stores ***341***
The Contents of Templates That SAS Supplies 341

Syntax: TEMPLATE Procedure: Managing Template Stores ***342***
PROC TEMPLATE Statement 343
DELETE Statement 343
LINK Statement 344
LIST Statement 345
PATH Statement 351
SOURCE Statement 353
TEST Statement 357

Examples: TEMPLATE Procedure: Managing Template Stores ***358***
Example 1: Listing Templates in a Template Store 358
Example 2: Using a WHERE Expression to Select Items in a Template Store . . . 361
Example 3: Viewing the Source of a Template 362

Overview: TEMPLATE Procedure: Managing Template Stores

What Is a Template Store?

A template store is an item store that stores items that were created by the TEMPLATE procedure. Items that SAS provides are in the item store Sashelp.Tmplmst. You can store items that you create in any template store where you have Write access.

Note: A template store can contain multiple levels known as directories. When you specify a template store in the ODS PATH statement, however, you specify a two-level name that includes a libref and the name of a template store in the SAS library that the libref references.

Why Use the TEMPLATE Procedure to Manage Template Stores?

You can use the TEMPLATE procedure to manage and navigate the template stores that store the items that SAS supplies or that you create. The TEMPLATE procedure enables you to perform the following management tasks for the template stores:

- delete column templates, header templates, footer templates, styles, table templates, or tagsets
- list items in one or more template stores
- view the source code of column templates, header templates, footer templates, styles, table templates, or tagsets
- test the most recently created item

You can navigate around the template stores by doing the following:

- create links to existing items
- specify which locations to write to or read from when you create or use PROC TEMPLATE items, and specify the order in which to search for them

Concepts: TEMPLATE Procedure: Managing Template Stores

The Contents of Templates That SAS Supplies

SAS provides templates for these items:

- tables
- crosstabulation tables
- SAS statistical graphics
- styles
- tagsets

To view the contents of a template, use the SAS windowing environment, the SAS window command ODSTEMPLATES, or the TEMPLATE procedure.

- **SAS Windowing Environment**

  1. From the SAS Explorer, select **View** ⇨ **Results**.
  2. In the Results window, select the **Results** folder. Right-click to open the Templates window.
  3. To view the definitions or templates that SAS supplies, click the plus sign that is next to the Sashelp.Tmplmst item store.
  4. Click the plus sign that is next to an icon to view the contents of that template store or directory in a template store. If there is no plus sign next to the icon, double-click the icon to view the contents of that directory.

- **SAS Windowing Command**

  1. To view the Templates window, submit this command in the command bar:

     ```
     odstemplates
     ```

     This display shows the Templates window that contains the item stores **Sasuser.Templat** and **Sashelp.Tmplmst**.

  2. When you double-click an item store, such as **Sashelp.Tmplmst**, that item store expands to list the directories where ODS templates are stored. The templates that SAS provides are in the item store **Sashelp.Tmplmst**.

- **TEMPLATE Procedure**

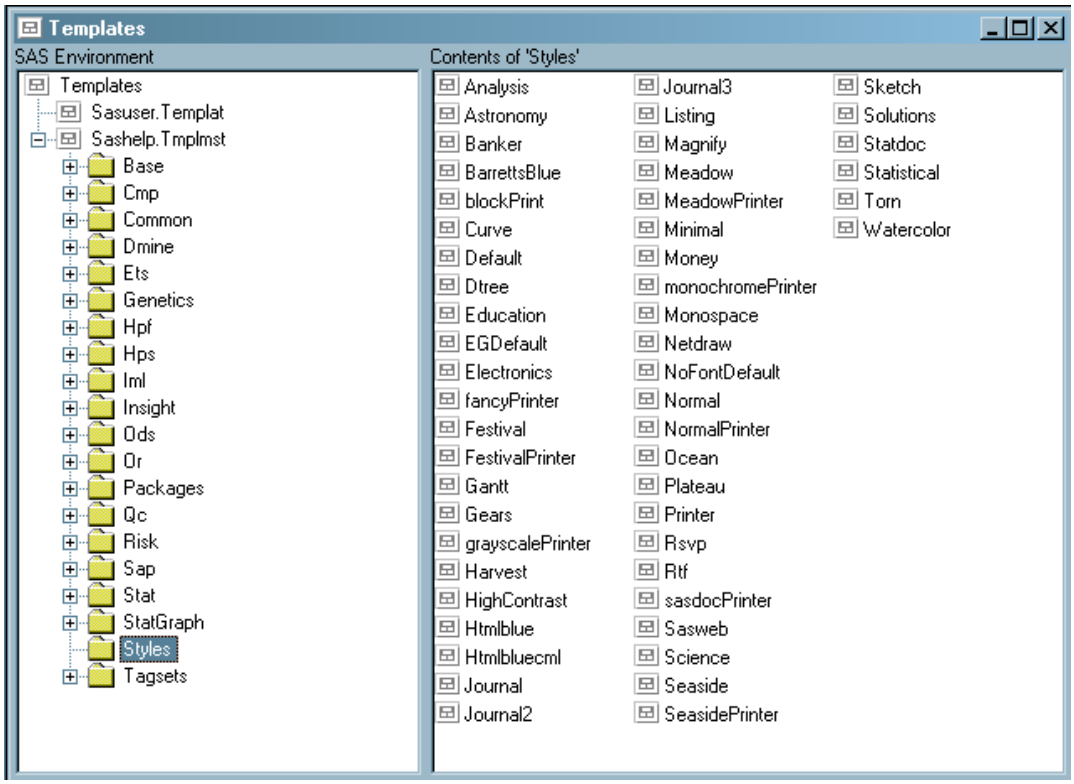
  The SOURCE statement writes the source code for the specified template to the SAS log. For example, if you want to view the source for all the objects in Base SAS, submit this code:

```
proc template;
source base;
run;
```

Note: For more information, see "SOURCE Statement" on page 353.

From the Templates window, you can see the definitions and templates that SAS supplies or that you created. This figure shows the styles that SAS supplies.

Figure 11.1 *Templates That SAS Supplies*



Syntax: TEMPLATE Procedure: Managing Template Stores

PROC TEMPLATE;
DELETE *item-path*< / STORE=*libref.template-store* >;
LINK *item-path-1* **TO** *item-path-2* <*/options*>;
LIST <*starting-path*><*/ options*>;
PATH *location(s)*;
SOURCE *item-path* <*/options*><STORE=*libref.template-store*>;
TEST DATA=*data-set*< / STORE=*libref.template-store*>;

Statement	Task	Example
PROC TEMPLATE	Begin a PROC TEMPLATE template	Ex. 1, Ex. 2, Ex. 3
DELETE	Delete the specified item	
LINK	Create a link to an existing item	
LIST	List items in one or more template stores	Ex. 1, Ex. 2
PATH	Specify which locations to write to or read from when you create or use PROC TEMPLATE items, and specify the order in which to search for them	Ex. 1, Ex. 2, Ex. 3
SOURCE	Write the source code for the specified item to the SAS log	Ex. 3
TEST	Test the most recently created item by binding it to the specified data set	

PROC TEMPLATE Statement

Begins a PROC TEMPLATE template.

Syntax

PROC TEMPLATE;
DELETE *item-path*< / STORE=*libref.template-store* >;
LINK *item-path-1* **TO** *item-path-2* <*/options*>;
LIST <*starting-path*><*/ options*>;
PATH *location(s)*;
SOURCE *item-path* <*/options*><STORE=*libref.template-store*>;
TEST DATA=*data-set*< / STORE=*libref.template-store*>;

DELETE Statement

Deletes the specified item.

Examples: "Example 2: Comparing the EDIT Statement to the DEFINE TABLE Statement" on page 553
"Example 1: Editing a Table Template That a SAS Procedure Uses" on page 550

Syntax

DELETE *item-path* < / STORE=*template-store*; >

Required Argument

item-path
specifies an item to delete. An *item-path* consists of one or more names, separated by periods. Each name represents a directory in a template store. (A template store is a type of SAS file.) If the same item exists in multiple template stores, PROC TEMPLATE deletes the item from the first template store in the current path where you have Write access.

CAUTION
Deleting a directory in a template store deletes all subdirectories and items in the directory. If the path that you specify is a directory rather than an item, PROC TEMPLATE deletes all the directories and all the items in that directory.

Optional Argument

STORE= *template-store*
specifies the template store where the template that you want to delete is stored.

LINK Statement

Creates a link to an existing item.

Tip: Creating a link to an item has the same effect as creating a new item that inherits its characteristics from another item (see the discussion of "PARENT= Statement" on page 461. However, using a link is more efficient than using inheritance, because linking does not actually create a new item.

Syntax

LINK *item-path-1* **TO** *item-path-2* </ *options*>;

Required Arguments

item-path-1
specifies the path of the item to create. PROC TEMPLATE creates the item in the first template store in the path that you can write to.

item-path-2
specifies the path of the item to link to. If the same item exists in multiple template stores, PROC TEMPLATE uses the one from the first template store in the current path that you can read.

Tip PROC TEMPLATE does not confirm that *item-path-2* exists when it compiles the item.

Optional Arguments

NOTES= '*text*'
specifies notes to store in the item.

Requirement Enclose the text in quotation marks.

Tip Notes of this type become part of the compiled item, which you can view with the SOURCE statement, whereas SAS comments do not.

STORE=*libref.template-store*
specifies the location where the link is created.

Restriction The STORE= option syntax does not become part of the compiled item.

Tip The link always points to the first item with the same name that it finds in the ODS path.

LIST Statement

Lists the items in one or more template stores.

Example: "Example 1: Listing Templates in a Template Store" on page 358

Syntax

LIST <*starting-path*></ *options*>;

Optional Argument

starting-path
specifies a level within each template store where PROC TEMPLATE starts listing items. For example, if *starting-path* is `base.univariate`, PROC TEMPLATE lists only `base.univariate` and the items within it and within all the levels that it contains.

Default If you omit a *starting-path*, then the LIST statement lists all items in all template stores unless the ODS PATH statement confines the search to the specified template stores.

Restriction This option must precede the forward slash (/) in the LIST statement.

Options

SORT=*statistic <sorting-order>*
sorts the list of items by the specified statistic in the specified sorting order.

statistic
is one of the following:

CREATED
is the date on which the item was created.

NOTES
is the content of any NOTES statement in the PROC TEMPLATE step that created the item.

Alias LABEL

LINK
is the name of the item that the current item links to (see "LINK Statement" on page 344).

PATH
is the path to the current item in the template store. (The path does not include the name of the template store).

SIZE
is the size of the item.

TYPE
is the type of the item: COLUMN, FOOTER, HEADER, STYLE, TABLE, or LINK. If the item is simply a level in the item store, its type is DIR.

Default PATH

sorting-order
specifies whether SORT= sorts from low values to high values or from high values to low values.

ASCENDING
sorts from low values to high values.

Alias A

DESCENDING
sorts from high values to low values.

Alias D

Default ASCENDING

STATS=ALL | STORE | *(statistic-1 <, … statistic-n>)*
specifies the information to include in the list of items.

ALL
includes all available information.

STORE
specifies the name of the template store that contains the items.

(statistic-1 <, … statistic-n>)
includes the specified information. *statistic* is one or more of the following:

CREATED
is the date on which the item was created.

NOTES
is the content of any NOTES statement in the PROC TEMPLATE step that created the item.

Alias LABEL

LINK
is the name of the item that the current item links to (see "LINK Statement" on page 344).

SIZE
is the size of the item.

Default Whether or not you specify STATS=, the list of items always includes an observation number, the path to the item, and its type.

STORE=*libref.template-store*
specifies the template store to process.

Default All template stores in the current template path.

See For information about setting the current template path, see the "PATH Statement" on page 351.

WHERE=*where-expression*
selects, for listing, items that meet a particular condition. For example, the following statement lists items that contain the word "Default" in the path to the current template:

```
list / where=(path ? 'Default');
```

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

where-expression has this form:

(*subsetting-variable* <*comparison-operator where-expression-n*>)

subsetting-variable
is a special type of WHERE expression operand used by the SOURCE statement to help you find common values in items. Subsetting variables are one or more of the following:

PATH
PATH
is the fully qualified path of a template.

Aliases NAME

NAME

TEMPLATE

TEMPLATE

Example This SOURCE statement displays the code for all items that contain the word "Default" in the name of the current template:

```
source / where=(path ? 'Default'); run;
.
```

STORE
is the name of the template store that contains the item.

TYPE
TYPE
is the type of the item. TYPE is one of the following:

COLUMN
specifies that the template is a column in a table.

FOOTER
specifies that the template is a footer in a table.

HEADER
specifies that the template is a header in a table.

LINK
specifies that the template is a link or URL.

STYLE
specifies that the definition is a style.

TABLE
specifies that the definition is a table template.

TAGSET
specifies that the definition is a tagset.

Example This SOURCE statement displays the source code for all tagsets that have the word "Default" in the path:

```
source / where=(lowcase(type) = 'tagset' && _path_ ? 'Default');
```

The LOWCASE function converts all letters in an argument to lowercase.

NOTES
is the content of any NOTES statement in the PROC TEMPLATE step that created the item. The contents is displayed in the LABEL field.

Alias LABEL

Example This SOURCE statement displays the source code for all items where the label contains the words "common matrix" and the item is a link:

```
source / where=(lowcase(label) ?
                      'common matrix' && _type_ = 'Link');
run;
```

The LOWCASE function converts all letters in an argument to lowercase.

SIZE
is the size of the item in bytes.

Example This SOURCE statement displays the source code for all items that are larger than 70000 bytes:

```
source / where=(size > 70000);
run;
```

CREATED
is the date on which the item was created.

Example This SOURCE statement displays the source code for all of the items that were created today in all of the template stores in the current template path:

```
source / where=(datepart(created) = today());
```

The DATEPART function extracts the date from a SAS datetime value.

CDATE
CDATE
is the creation date of the item.

Example This SOURCE statement displays the source code for all of the items with a creation date of 16JUL2004:

```
source / where=(_cdate_ = '16JUL2004'd);
run;
```

CDATETIME
CDATETIME
is the creation datetime of the item.

Example This SOURCE statement displays the source code for all items with a creation SAS datetime of May 1, 2003 at 9:30:

```
source / where=(_cdatetime_ = '01may04:9:30:00'dt);
run;
```

CTIME
CTIME
is the creation time of the item.

Example This SOURCE statement displays the source code of all items with a creation time of 9:25:19 PM:

```
source / where=(_ctime_ = '9:25:19pm't);
run;
```

MDATE
MDATE
is the modification date of the item.

Example This SOURCE statement displays the source code of all items with a modification date of 16JUL2004:

```
source / where=(_mdate_ = '16JUL2004'd);
run;
```

MDATETIME
MDATETIME
is the modification datetime of the item.

Example This SOURCE statement displays the source code of all items with a modification SAS datetime of May 1, 2003 at 9:30:

```
source / where=(_mdatetime_ = '01may04:9:30:00'dt);
run;
```

MODIFIED
is the date on which the item was modified.

Example This SOURCE statement displays the source code of all items that were modified today in all of the template stores in the current template path:

```
source / where=(datepart(modified) = today());
```

The DATEPART function extracts the date from a SAS datetime value.

MTIME
MTIME
is the modification time of the item.

Example This SOURCE statement displays the source code of all items with a modification time of 9:25:19 PM:

```
source / where=(_mtime_ = '9:25:19pm't);
run;
```

comparison-operator
compares a variable with a value or with another variable. The following table lists the comparison operators:

Table 11.1 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 2: Using a WHERE Expression to Select Items in a Template Store " on page 361

PATH Statement

Specifies locations to write to or read from when you create or use PROC TEMPLATE templates or definitions, and specifies the order in which to search for them. This statement overrides the ODS PATH statement for the duration of the PROC TEMPLATE step.

Examples: "Example 1: Listing Templates in a Template Store" on page 358
"Example 3: Viewing the Source of a Template" on page 362

Syntax

PATH <(APPEND)(| PREPEND)(| REMOVE)> *location(s)*;
PATH *path-argument*;

Required Arguments

location(s)
specifies one or more locations to write to or read from when creating or using PROC TEMPLATE items and the order in which to search for them. ODS searches the locations in the order in which they appear on the statement. It uses the first definition that it finds that has the appropriate access mode (Read, Write, or Update) set.

Each *location* has this form:

<*libref.*>*item-store* <(READ | UPDATE | WRITE)>

<libref.>item-store
identifies an item store to read from, to write to, or to update. If an item store does not already exist, then the PATH statement creates it.

(READ
UPDATE
WRITE)
specifies the access mode for the item. An access mode is one of the following:

READ
provides Read-Only access.

WRITE
provides Write access (always creating a new template store) as well as Read access.

UPDATE
provides Update access (creating a new template store only if the specified one does not exist) as well as Read access.

Default READ

Default The general default path is as follows: Sasuser.Templat (UPDATE), Sashelp.Tmplmst (READ). If you have specified the RSASUSER SAS

system option, then the default path is as follows: Work.Templat(UPDATE), Sasuser.Templat (READ), Sashelp.Tmplmst(READ). SAS stores all the items that it provides in Sashelp.Tmplmst.

Tip If you want to be able to ignore all the items that you create, then keep them in their own item stores so that you can leave them out of the list of item stores that ODS searches.

See "RSASUSER System Option" in *SAS System Options: Reference* for more information about how to open the Sasuser library for Read access or Read-Write access.

path-argument

sets or displays the ODS path.

path-argument is one of the following:

RESET
sets the ODS path to the default settings Sasuser.Templat (UPDATE) and Sashelp.Tmplmst (READ).

SHOW
displays the current ODS path.

VERIFY
sets the ODS path to include only templates supplied by SAS. Specifying VERIFY is the same as specifying ODS PATH Sashelp.Tmplmst (READ).

Default The general default path is as follows: Sasuser.Templat (UPDATE), Sashelp.Tmplmst (READ). If you have specified the RSASUSER SAS system option, then the default path is as follows: Work.Templat(UPDATE), Sasuser.Templat (READ), Sashelp.Tmplmst(READ). SAS stores all the items that it provides in Sashelp.Tmplmst.

Optional Argument

(APPEND | PREPEND | REMOVE)

adds one or more locations to a path, or removes one or more locations from a path.

APPEND
adds one or more locations to the end of a path. When you append a location to a path, all duplicate instances (with the same name and same permissions) of that item store are removed from the path. Only the last item store with the same name and permissions is kept.

PREPEND
adds one or more locations to the beginning of a path. When you prepend a location to a path, all duplicate instances (with the same name and same permissions) of that item store are removed from the path. Only the first item store with the same name and permissions is kept.

REMOVE
removes one or more locations from a path.

Default If you omit an APPEND, PREPEND, or REMOVE option, then the PATH statement overwrites the complete path.

SOURCE Statement

Writes the source code for the template specified to the SAS log.

Example: "Example 3: Viewing the Source of a Template" on page 362

Syntax

SOURCE *item-path* </ *options*>;

Required Argument

item-path
specifies the path of the item that you want to write to the SAS log. If the same item exists in multiple template stores, PROC TEMPLATE uses the one from the first template store that you can read in the current path.

Tip PROC TEMPLATE stores items in compiled form. The SOURCE statement actually decompiles the item. Because SAS comments are not compiled, comments that are in the source code do not appear when you decompile the item. If you want to annotate the item, use the NOTES statement inside the item or the block of editing instructions, or use the NOTES= option in the LINK statement. These notes do become part of the compiled item. (See the "NOTES Statement" on page 541 and the discussion of the "LINK Statement" on page 344. You can also specify notes as quoted strings in the DYNAMIC, MVAR, NMVAR, REPLACE, and STYLE statements.)

Optional Arguments

EXPAND
prints the source of all parents of a template.

Example The following PROC TEMPLATE block prints all of the parents of the template base.contents.variables:

```
proc template;
    source base.contents.variables / expand;
run;
```

FILE= '*file-specification*' | *fileref*
specifies a file to write the item to.

'*file-specification*'
is the name of an external file to write to.

Requirement The *external-file* that you specify must be enclosed in quotation marks.

fileref
is a file reference that has been assigned to an external file. Use the FILENAME statement to assign a fileref.

Default If you omit a filename where you want the source code written, then the SOURCE statement writes the source code to the SAS log.

See "Statements" in *SAS DATA Step Statements: Reference* for information about the FILENAME statement.

NOFOLLOW
specifies that the program does not resolve links in the PARENT= statement, which specifies the item that the current item inherits from. For information about the PARENT= statement, see "PARENT= Statement" on page 461 in the styles attribute section.

STORE= *libref.template-store*
specifies the template store where the item is located.

Interaction In most cases, the STORE= option is added to the definition statement when PROC TEMPLATE displays the source code. However, if the template store specified in the STORE= option is in the ODS path with only Read permission, then PROC TEMPLATE does not include the STORE= option in the source code that it displays. If there is no STORE= option, when you run the code, then the item that it creates goes to the first template store for which you have Update permission in the ODS path.

WHERE=(*where-expression*)
selects items that meet a particular condition. For example, the following statement displays the source code for items that contain the word "Default" in the path to the current template: `source / where=(path ? 'Default');`

where-expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

where-expression has this form:

(*subsetting-variable* <*comparison-operator where-expression-n*>)

subsetting-variable
a special type of WHERE expression operand used by the SOURCE statement to help you find common values in items. Subsetting variables are one or more of the following:

PATH
PATH
is the fully qualified path of a template.

Aliases NAME

NAME

TEMPLATE

TEMPLATE

Example This SOURCE statement displays the code for all items that contain the word "Default" in the name of the current template:

```
source / where=(path ? 'Default'); run;
```

TYPE
TYPE
is the type of the item. TYPE is one of the following:

COLUMN
specifies that the template is a column in a table.

FOOTER
specifies that the template is a footer in a table.

HEADER
specifies that the template is a header in a table.

LINK
specifies that the template is a link or URL.

STYLE
specifies that the definition is a style.

TABLE
specifies that the definition is a table template .

TAGSET
specifies that the definition is a tagset.

Example This SOURCE statement displays the source code for all tagsets that have the word "Default" in the path:

```
source / where=(lowcase(type) = 'tagset' && _path_ ? 'Default');
```

The LOWCASE function converts all letters in an argument to lowercase.

NOTES
is the content of any NOTES statement in the PROC TEMPLATE step that created the item. The contents is displayed in the LABEL field.

Alias LABEL

Example This SOURCE statement displays the source code for all items where the label contains the words "common matrix" and the item is a link:

```
source / where=(lowcase(label) ?
                        'common matrix' && _type_ = 'Link');
run;
```

The LOWCASE function converts all letters in an argument to lowercase.

SIZE
is the size of the item in bytes.

Example This SOURCE statement displays the source code for all items that are larger than 70000 bytes:

```
source / where=(size > 70000);
run;
```

CREATED
is the date on which the item was created.

Example This SOURCE statement displays the source code for all of the items that were created today in all of the template stores in the current template path:

```
source / where=(datepart(created) = today());
```

The DATEPART function extracts the date from a SAS datetime value.

CDATE
CDATE
is the creation date of the item.

Example This SOURCE statement displays the source code for all of the items with a creation date of 16JUL2004:

```
source / where=(_cdate_ = '16JUL2004'd);
run;
```

CDATETIME
CDATETIME
is the creation datetime of the item.

Example This SOURCE statement displays the source code for all items with a creation SAS datetime of May 1, 2003 at 9:30:

```
source / where=(_cdatetime_ = '01may04:9:30:00'dt);
run;
```

CTIME
CTIME
is the creation time of the item.

Example This SOURCE statement displays the source code of all items with a creation time of 9:25:19 PM:

```
source / where=(_ctime_ = '9:25:19pm't);
run;
```

MDATE
MDATE
is the modification date of the item.

Example This SOURCE statement displays the source code of all items with a modification date of 16JUL2004:

```
source / where=(_mdate_ = '16JUL2004'd);
run;
```

MDATETIME
MDATETIME
is the modification datetime of the item.

Example This SOURCE statement displays the source code of all items with a modification SAS datetime of May 1, 2003 at 9:30:

```
source / where=(_mdatetime_ = '01may04:9:30:00'dt);
run;
```

MODIFIED
is the date on which the item was modified.

Example This SOURCE statement displays the source code of all items that were modified today in all of the template stores in the current template path:

```
source / where=(datepart(modified) = today());
```

The DATEPART function extracts the date from a SAS datetime value.

MTIME
MTIME
is the modification time of the item.

Example This SOURCE statement displays the source code of all items with a modification time of 9:25:19 PM:

```
source / where=(_mtime_ = '9:25:19pm't);
run;
```

comparison-operator
compares a variable with a value or with another variable. The following table lists the comparison operators:

Table 11.2 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

TEST Statement

Tests the most recently created item by binding it to the specified data set.

Syntax

TEST DATA=*data-set* </ STORE=*libref.template-store*>;

Required Argument

DATA=*data-set*
specifies the SAS data set to bind to the most recently created item. ODS sends this output object to all open ODS destinations.

Optional Argument

STORE=*libref.template-store*
specifies the template store where the item is located.

Requirement If you specify this option, then the template store that you specify must match the template store in the DEFINE statement that created the item.

Examples: TEMPLATE Procedure: Managing Template Stores

Example 1: Listing Templates in a Template Store

Features: PATH statement
LIST statement:
starting-path option
SORT= option

Details

This example lists the items for the Base.Univariate directory in the item store Sashelp.Tmplmst.

Program

```
proc template;
path sashelp.tmplmst;

list base.univariate / sort=path descending;
run;
```

Program Description

Specify which locations to search for items that were created by PROC TEMPLATE. The PATH statement specifies to search for templates and definitions that were created by PROC TEMPLATE in the Sashelp.Tmplmst item store.

```
proc template;
path sashelp.tmplmst;
```

List in descending order the items that are stored within a specified level of the template store. The LIST statement lists the templates and definitions in one or more template stores. The starting path **`base.univariate`** specifies the level within the template store where PROC TEMPLATE is to start listing the items. The SORT= option sorts the list of items. The items are sorted in descending order.

```
list base.univariate / sort=path descending;
run;
```

Output

Output 11.1 *Partial Listing of Base.Univariate Template Store*

Listing of: SASHELP.TMPLBASE

Path Filter is: Base.Univariate

Sort by: PATH/DESCENDING

Obs	Path	Type
1	Base.Univariate.Wins	Table
2	Base.Univariate.Trim	Table
3	Base.Univariate.Robustscale	Table
4	Base.Univariate.Quantiles	Table
5	Base.Univariate.QQPlotData	Table
6	Base.Univariate.ProjGrad	Table
7	Base.Univariate.ProblemDescription	Table
8	Base.Univariate.ProbPlotData	Table
9	Base.Univariate.ParameterEstimatesStart	Table
10	Base.Univariate.ParameterEstimatesResults	Table
11	Base.Univariate.PValue	Column
12	Base.Univariate.PPPlotData	Table
13	Base.Univariate.Normal	Table
14	Base.Univariate.Moments	Link
15	Base.Univariate.Modes	Table
16	Base.Univariate.Missings	Table
17	Base.Univariate.Measures	Table
18	Base.Univariate.Location	Table
19	Base.Univariate.LocCount	Table
20	Base.Univariate.LinConSol	Tabl
21	Base.Uni	
	ivariate.Lagrange	Table

Example 2: Using a WHERE Expression to Select Items in a Template Store

Features: PATH statement

LIST statement:
- *where-expression* option
- *starting-path* option
- SORT= option

Details

This example uses a WHERE expression to select, for listing, fitted distribution table templates in the Base.Univariate directory.

Program

```
proc template;
path sashelp.tmplmst;

list base.univariate / sort=path descending
where=(lowcase(path) ? 'fit');
run;
```

Program Description

Specify which locations to search for items that were created by PROC TEMPLATE. The PATH statement specifies to search for items that were created by PROC TEMPLATE in the Sashelp.Tmplmst item store.

```
proc template;
path sashelp.tmplmst;
```

List, in descending order, the items with the word "fit" in their pathname. The LIST statement lists the items in one or more template stores. The starting path **`base.univariate`** specifies the level within the template store where PROC TEMPLATE is to start listing the items. The WHERE expression finds items in the template store that have the word "fit" in their pathname. The LOWCASE function converts all letters in an argument to lowercase. The SORT= option sorts the list of items. The items are sorted in descending order.

```
list base.univariate / sort=path descending
where=(lowcase(path) ? 'fit');
run;
```

Output

Output 11.2 *Listing of Fitted Distribution Templates in the Base.Univariate Template Store*

Listing of: SASHELP.TMPLBASE

Path Filter is: Base.Univariate

Sort by: PATH/DESCENDING

Obs	Path	Type
1	Base.Univariate.FitQuant	Table
2	Base.Univariate.FitParms	Table
3	Base.Univariate.FitGood	Table

Example 3: Viewing the Source of a Template

Features: PATH statement
SOURCE statement

Details

This example displays the source code for the Style_popup tagset that SAS provides.

Program

```
proc template;
path sashelp.tmplmst;

source Tagsets.Style_popup;
```

```
run;
```

Program Description

Specify which locations to search for items that were created by PROC TEMPLATE. The PATH statement specifies to search for items that were created by PROC TEMPLATE in the Sashelp.Tmplmst item store.

```
proc template;
path sashelp.tmplmst;
```

Write the source code of the specified item. The SOURCE statement writes the source code for the tagset Style_popup that SAS provides. The source code is written to the SAS log.

```
source Tagsets.Style_popup;
run;
```

Partial Source Code of the Template Tagset.Style_Popup That Is Written to the SAS Log

```
1    proc template;
NOTE: Writing HTML Body file: sashtml.htm
2    path sashelp.tmplmst;
3    source Tagsets.Style_popup;
define tagset Tagsets.Style_popup;
   notes "This is HTML pop up styles.";

   define event style_class;
      put "." HTMLCLASS " {font-family: " HTMLCLASS "}" NL;
      put "." HTMLCLASS NL;
      put "{" NL;

      trigger stylesheetclass;
      put "}" NL;

      trigger linkclass /if exists( linkcolor);

      trigger vlinkclass /if exists( visitedlinkcolor);

      trigger alinkclass /if exists( activelinkcolor);
   end;

   define event doc_head;
      start:
         put "<head>" NL;
         put VALUE NL;

      finish:

         trigger popup_script;
         put "</head>" NL NL;
   end;

   define event stylesheet_link;
      set $stylesheet_idx "0";
      break /if ^exists( url);
      put "<style type=""text/css"">" NL "<!--" NL;

      trigger alignstyle;
      put "-->" NL "</style>" NL;
      set $urlList url;


... more log output ...


           %nrstr("   if %( element.parentElement &&
element.parentElement.parentElement &&
 element.parentElement.parentElement.className %)") NL;
      put "   {" NL;
      put "      element.style.backgroundColor = '#ff6666';" NL;
      put "      element.style.color = 'white';" NL;
      put "      element.title = element.parentElement.parentElement.className;" NL;
      put "   }" NL;
      put %nrstr("   if %( element.parentElement && element.parentElement.className
%)") NL;
      put "   {" NL;
      put "      element.style.backgroundColor = '#ff6666';" NL;
      put "      element.style.color = 'white';" NL;
      put "      element.title = element.parentElement.className;" NL;
      put "   }" NL;
      put "   if ( element.className )" NL;
      put "   {" NL;
      put "      element.style.backgroundColor = '#ff6666';" NL;
```

Chapter 12

TEMPLATE Procedure: Creating Crosstabulation Table Templates

Overview: TEMPLATE Procedure: Creating Crosstabulation Table Templates . . ***365***
Using the TEMPLATE Procedure to Create a Customized Crosstabulation Table 366
What Can You Do with a Crosstabulation Template? 367

Concepts: TEMPLATE Procedure: Creating Crosstabulation Table Templates . ***370***
What Makes the Crosstabulation Table Unique? 370
Comparison between Table Templates and Crosstabulation Table Templates . . . 370

Syntax: TEMPLATE Procedure: Creating Crosstabulation Table Templates ***371***
DEFINE CELLVALUE Statement 372
DEFINE CROSSTABS Statement 375
DEFINE FOOTER Statement 381
DEFINE HEADER Statement 381
CELLSTYLE AS Statement 385
CELLVALUE Statement 387
DYNAMIC Statement 388
END Statement 389
FOOTER Statement 390
HEADER Statement 390
NOTES Statement 391
TEXT Statement 391

Usage: TEMPLATE Procedure: Creating Crosstabulation Table Templates ***395***
Working with the CrossTabFreqs Crosstabulation Table Template 395
Crosstabulation Table Regions and Corresponding Attributes 396

Examples: TEMPLATE Procedure: Creating Crosstabulation Table Templates . ***397***
Example 1: Creating a Customized Crosstabulation Table Template with No Legend 397
Example 2: Creating a Crosstabulation Table Template with a Customized Legend 409
Example 3: Adding Custom Formats to Cellvalues 420

Overview: TEMPLATE Procedure: Creating Crosstabulation Table Templates

Using the TEMPLATE Procedure to Create a Customized Crosstabulation Table

The TEMPLATE procedure enables you to customize the appearance of crosstabulation (contingency) tables that are created with the FREQ procedure.

By default, crosstabulation tables are formatted according to the CrossTabFreqs template that SAS provides. However, you can create a customized CrossTabFreqs table template by using the TEMPLATE procedure with the statements in the following table.

Table 12.1 *PROC TEMPLATE Statements*

Task	Statement
Create a crosstabulation table template	DEFINE CROSSTABS
Define a value that appears in the crosstabulation cells	DEFINE CELLVALUE
Specify the order in which the cellvalues are stacked in the cells	CELLVALUE
Create a template for a footer	DEFINE FOOTER
Create a template for a header	DEFINE HEADER
End a crosstabulation table template	END

What Can You Do with a Crosstabulation Template?

The CrossTabFreqs crosstabulation template describes how to display PROC FREQ's crosstabulation table. You can create a customized CrossTabFreqs crosstabulation template to do the following:

- use custom formats for cellvalues
- specify a style for each value in a cell
- change the stacking order of values in a cell
- change and style headers and footers
- use variable labels in headers and footers
- style table regions independently
- change or remove the legend

The following display shows a crosstabulation table that has been created with the default crosstabulation table template:

Figure 12.1 Crosstabulation Table Created with Default Crosstabulation Table Template

Frequency
Percent
Row Pct
Col Pct

Table of Citygovt by Robgrp

Citygovt(City Government Form)	Robgrp(Number of Meetings Scheduled) ?	Not Known	100 or Less	101-200	201-300	Over 300	Total
?	0 . . .	0 . . .	0 . . .	1 . . .	0 . . .	0 . . .	. .
Not Applicable	0 . . .	10 . . .	0 . . .	0 . . .	0 . . .	0 . . .	. .
Council Manager	0 . . .	0 . . .	47 12.30 22.27 55.95	63 16.49 29.86 58.88	49 12.83 23.22 62.03	52 13.61 24.64 46.43	211 55.24
Commission	0 . . .	0 . . .	6 1.57 28.57 7.14	7 1.83 33.33 6.54	3 0.79 14.29 3.80	5 1.31 23.81 4.46	21 5.50
Mayor Council	1 . . .	0 . . .	31 8.12 20.67 36.90	37 9.69 24.67 34.58	27 7.07 18.00 34.18	55 14.40 36.67 49.11	150 39.27
Total	. .	. .	84 21.99	107 28.01	79 20.68	112 29.32	382 100.00

Frequency Missing = 12

The following display shows PROC FREQ output that has been created with a modified CrossTabFreqs template:

Figure 12.2 *Crosstabulation Table Created with Modified Crosstabulation Table Template*

City Government Form by Number of Meetings Scheduled

Table of City Government Form by Number of Meetings Scheduled							
	Number of Meetings Scheduled						
City Government Form	?	Not Known	100 or Less	101-200	201-300	Over 300	Total
?	0 . . .	0 . . .	0 . . .	1 . . .	0 . . .	0 . . .	. .
Not Applicable	0 . . .	10 . . .	0 . . .	0 . . .	0 . . .	0 . . .	. .
Council Manager	0 . . .	0 . . .	47 12.30 22.27 55.95	63 16.49 29.86 58.88	49 12.83 23.22 62.03	52 13.61 24.64 46.43	211 55.24
Commission	0 . . .	0 . . .	6 1.57 28.57 7.14	7 1.83 33.33 6.54	3 0.79 14.29 3.80	5 1.31 23.81 4.46	21 5.50
Mayor Council	1 . . .	0 . . .	31 8.12 20.67 36.90	37 9.69 24.67 34.58	27 7.07 18.00 34.18	55 14.40 36.67 49.11	150 39.27
Total	. .	. .	84 21.99	107 28.01	79 20.68	112 29.32	382 100.00
Frequency Missing = 12							

Cellvalues are stacked in the following order:
Frequency
Percent
Row Percent
Column Percent

Concepts: TEMPLATE Procedure: Creating Crosstabulation Table Templates

What Makes the Crosstabulation Table Unique?

Crosstabulation tables produced by PROC FREQ are different from other tables that SAS produces. Most other tables consist of rows and columns with one value for each row-column combination. However, the crosstabulation table has these distinctive characteristics:

multiple values per cell
: The crosstabulation table can have up to nine values for each row-column combination, depending on the options specified by the TABLES statement. Most other tables that SAS creates have only one value for each row-column combination.

legend
: Crosstabulation tables have a separate box that is called a legend. The legend contains the labels for the cellvalues. No other table that SAS produces has a legend.

row variable column
: The far left column contains the row variable values. Each value in this column provides a label for the row in the same way that the column variable values provide labels for the columns.

column variable headers
: Each value in these headers provides a label for the columns of the table.

row and column totals
: The far right column contains the row totals. The bottom row contains the column totals.

grand total cell
: The grand total cell is the last cell of the row and column totals.

Comparison between Table Templates and Crosstabulation Table Templates

Because the crosstabulation table is unique, the syntax used to create crosstabulation templates differs significantly from other table templates.

- The crosstabulation template has no parent template, and it cannot serve as a parent to another template.
- Most of the attributes, such as CENTER and PANELS=, that are defined for classic table templates are not defined in the crosstabulation table template.
- Crosstabulation table templates use DEFINE CELLVALUE blocks instead of the DEFINE COLUMN blocks that are used in other table templates.
- Crosstabulation table templates use the CELLVALUE statement instead of the COLUMN statement that is used in other table templates.
- Both crosstabulation table templates and other table templates use DEFINE HEADER and DEFINE FOOTER blocks. However, the attributes that can be used in each of these blocks differ.
- DEFINE HEADER and DEFINE FOOTER blocks can contain multiple TEXT statements only in crosstabulation table templates. ODS then chooses which TEXT statement to use at execution time.
- A TEXT statement can specify a WHERE expression only in crosstabulation table templates. ODS uses the WHERE expression to determine whether to use the TEXT statement text as the header text.
- The CELL_STYLE, COLS_HEADER, COL_TOTAL_STYLE, COL_VAR_STYLE, GRAND_TOTAL_STYLE, LEGEND_STYLE, ROWS_HEADER, and ROW_VAR_STYLE attributes are unique to the crosstabulation table.
- The ROWS_HEADER and COLS_HEADER attributes are unique to the crosstabulation table.
- The NVAR, MVAR, and TRANSLATE-INTO statements are not supported for crosstabulation templates.

Syntax: TEMPLATE Procedure: Creating Crosstabulation Table Templates

```
PROC TEMPLATE
   DEFINE CROSSTABS table-path </ STORE=libref.template-store>;
      <table-attribute-1; <table-attribute-n>;>
      CELLVALUE cellvalues;
      DEFINE CELLVALUE cellvalue;
         statements-and-attributes;
      END;
      DEFINE HEADER header-name;
         statements-and-attributes;
      END;
      DEFINE FOOTER footer-name;
         statements-and-attributes;
      END;
      DYNAMIC variable-1 <'text-1'> <variable-n <'text-n'>>;
```

```
    FOOTER footer-name(s);
    HEADER header-name(s);
    NOTES text;
END;
```

Statement	Task	Example
CELLSTYLE AS	Set the style element of the cells in the column according to the values of the variables	Ex. 1, Ex. 2
CELLVALUE	Specify the order in which the cellvalues are stacked in the cells	Ex. 1, Ex. 2
DEFINE CELLVALUE	Define a value that appears in the crosstabulation cells	Ex. 1, Ex. 2
DEFINE CROSSTABS	Create a crosstabulation table template	Ex. 1, Ex. 2, Ex. 3
DEFINE FOOTER	Create a template for a footer	Ex. 1, Ex. 2
DEFINE HEADER	Create a template for a header	Ex. 1, Ex. 2
DYNAMIC	Define a symbol that references a value that the data component supplies from the procedure or DATA step	Ex. 1, Ex. 2
END	End a template	Ex. 1, Ex. 2, Ex. 3
FOOTER	Declare a symbol as a footer in the table and specify the order of the footers	Ex. 1, Ex. 2
HEADER	Declare a symbol as a header in the table and specify the order of the headers	Ex. 1, Ex. 2
NOTES	Provide information about a template	Ex. 1, Ex. 2
TEXT	Specify the text of the header or the footer	Ex. 1, Ex. 2

DEFINE CELLVALUE Statement

Defines a value that appears in the crosstabulation cells.

Note: The DEFINE CELLVALUE statement begins a DEFINE CELLVALUE block. The following statements are commonly used within the block: “CELLSTYLE AS Statement” on page 385, “DYNAMIC Statement” on page 388, “NOTES Statement” on page 391, and “END Statement” on page 389.

Example: “Example 1: Creating a Customized Crosstabulation Table Template with No Legend” on page 397

Syntax

DEFINE CELLVALUE <*cellvalue*>;

<*cellvalue-attribute-1*;> < *cellvalue-attribute-n*;>

CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*] >

<, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*]>>;

DYNAMIC *variable-1*<'*text-1*'> < *variable-n*<'*text-n*'>>;

NOTES '*text*';

END;

Optional Argument

cellvalue

specifies one of the possible values that PROC FREQ can produce for a crosstabulation table.

For a cellvalue to appear in a cell, it must meet one of these requirements:

- specified in a DEFINE CELLVALUE statement
- included in the CELLVALUE statement
- not suppressed by one of the following options for the TABLES statement in PROC FREQ: NOFREQ, NOPERCENT, NOROW, NOCOL, or CUMCOL
- requested by one of the following options: EXPECTED, DEVIATION, CELLCHI2, or TOTPCT

To prevent a cellvalue from appearing in a table, you need to change only one of the preceding specifications.

cellvalue is one of the following:

Frequency
is the frequency count.

Expected
is the expected frequency of the cell.

Deviation
is the deviation of the cell frequency from the expected value.

CellChiSquare
is the cell's contribution to the total Pearson chi-square statistic.

TotalPercent
is the percentage of total frequency on `n`-way tables when `n>2`.

Percent
is the percentage of the table frequency.

RowPercent
is the percentage of the row frequency.

ColPercent
is the percentage of the column frequency.

CumColPercent
is the cumulative percentage of the column frequency.

DEFINE CELLVALUE Attribute Statements

This section lists all the attribute statements that you can use in a cellvalue template and the tasks that are associated with the statements. For all attributes that support a value of ON, these forms are equivalent: ATTRIBUTE-NAME and ATTRIBUTE-NAME=ON.

Table 12.2 *DEFINE CELLVALUE Attribute Statements*

Task	Statement
Specify which format to use for the cellvalue if both a crosstabulation template and a data component specify a format	"DATA_FORMAT_OVERRIDE=ON | OFF" (p. 374)
Specify the format for the cellvalue	" FORMAT=*format_name <format-width<decimal-width> >*;" (p. 374)
Override the width specified by the FORMAT= attribute	"FORMAT_WIDTH=*positive-integer* " (p. 375)
Override the number of decimals specified by the FORMAT= attribute	"FORMAT_NDEC=*positive-integer*" (p. 375)
Specify the text in the legend	"HEADER='*text*'" (p. 375)
For the Output destination, specify the label for the data set column corresponding to the cellvalue	"LABEL='*text*' " (p. 375)
Specify whether a cellvalue appears in the crosstabulation table	"PRINT= ON | OFF" (p. 375)

DATA_FORMAT_OVERRIDE=ON | OFF
specifies which format to use if both a crosstabulation template and a data component specify a format for Frequency, Expected, and Deviation.

ON
selects the format specified in the data component.

OFF
selects the format specified in the crosstabulation template.

Default OFF

Interaction If you specify DATA_FORMAT_OVERRIDE=ON, and the FORMAT option is specified in the TABLES statement in PROC FREQ, then the data component specifies that format for the Frequency, Expected, and Deviation cellvalues.

FORMAT=*format_name <format-width<decimal-width>* >;
specifies the format for the column.

Default If you omit the FORMAT= option, PROC TEMPLATE uses the format that the data component provides. If the data component

does not provide a format, PROC TEMPLATE uses one of the following: BEST8. for integers, 12.3 for floating-point values, or the length of the variable for character variables.

Range The minimum cell width is 8, and the maximum width is 25.

Interaction For LISTING output, the width of the cells is governed by the format width. Cells are at least one character wider than the format width.

FORMAT_WIDTH=*positive-integer*
overrides the width specified by the FORMAT= attribute statement.

FORMAT_NDEC=*positive-integer*
overrides the number of decimals specified by the FORMAT= attribute statement.

HEADER='*text*'
specifies the text in the legend.

Tip For LISTING output, only the first 15 characters of text are displayed.

LABEL='*text*'
for the OUTPUT destination, specifies the label for the data set column that corresponds to the cellvalue.

PRINT= ON | OFF
specifies whether the cellvalue appears in the crosstabulation table.

Both this attribute and the TABLES statement option for the cellvalue control the presence of the cellvalue in the table. For example, the expected cell frequency is present only when the EXPECTED option is used and the Expected cellvalue template has PRINT=ON specified.

DEFINE CROSSTABS Statement

Creates a crosstabulation table template.

Note: The DEFINE CROSSTABS statement begins a crosstabs table template. The following statements are typically used within a DEFINE CROSSTABS block: "CELLVALUE Statement" on page 387, "DEFINE CELLVALUE Statement" on page 372, "DEFINE HEADER Statement" on page 381, "DEFINE FOOTER Statement" on page 381, "DYNAMIC Statement" on page 388, "FOOTER Statement" on page 390, "HEADER Statement" on page 390, and "NOTES Statement" on page 391.

Example: "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

DEFINE CROSSTABS *table-path* </ STORE=*libref.template-store*>;
<*table-attribute-1*; <*table-attribute-n*>;>
CELLVALUE *cellvalues*;
DEFINE CELLVALUE *cellvalue*;

```
    statements-and-attributes;
  END;
  DEFINE HEADER header-name;
    statements-and-attributes;
  END;
  DEFINE FOOTER footer-name;
    statements-and-attributes;
  END;
  DYNAMIC variable-1 <'text-1'> <variable-n <'text-n'>>;
  FOOTER footer-name(s);
  HEADER header-name(s);
  NOTES text;
END;
```

Summary of Optional Arguments

CELL_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]> | *style-name*
: Specify a style element and any changes to its attributes to use for the cellvalues in the non-summary rows and columns

COL_TOTAL_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
: Specify the style element and any changes to its attributes to use for the cellvalues in the last row in the table

COL_VAR_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
: Specify the style element and any changes to its attributes to use for the column variable values used as headers over the column variable value columns

COLS_HEADER=*header-name*
: Specify the name of the header to use over the column variable value columns in the table

GRAND_TOTAL_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
: Specify the style element and any changes to its attributes to use for the cellvalues in the rightmost column of the last row in the table

LABEL="*text*"
: Specify a label for the table

LEGEND_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
: Specify the style element and any changes to its attributes to use for the legend table that appears near the upper left corner of the table

ROW_TOTAL_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
: Specify the style element and any changes to its attributes to use for the cellvalues in the cells that contain row totals

ROW_VAR_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
: Specify the style element and any changes to its attributes to use for the row variable values in the leftmost column of the table

ROWS_HEADER=*header-name*
: Specify the name of the header to use over the row variable values (leftmost) column in the table

STORE= *template-store*
: Specify the template store in which to store the crosstabulation template.

STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
: Specify a style element and any changes to its attributes to use for the table

Required Argument

table-path
specifies where to store the crosstabulation table template. A table-path consists of one or more names, separated by periods. Each name represents a directory in a template store, which is a type of SAS file. For more information about template stores, see "Understanding Item Stores and Template Stores" in *SAS Output Delivery System: User's Guide*. PROC TEMPLATE writes the template to the first writable template store in the current path.

Requirement Crosstabulation table templates must be named CrossTabFreqs.

Optional Argument

STORE= *template-store*
specifies the template store in which to store the crosstabulation template. If the template store does not exist, it is created.

Restrictions The STORE= option does not become part of the template.

If the template is nested inside another template, do not use the STORE= option for the nested template, because the nested template is stored where the original template is stored.

Requirement The STORE= option must be proceeded by the forward slash (/) symbol.

DEFINE CROSSTABS Attributes

This section lists all of the attributes that you can use in a crosstabulation template.

Table 12.3 *Crosstabulation Attributes*

Task	Statement
Specify a style element and any changes to its attributes to use for the cellvalues in the non-summary rows and columns	CELL_STYLE= (p. 378)
Specify the name of the header to use over the column variable value columns in the table	COLS_HEADER= (p. 379)

Task	Statement
Specify the style element and any changes to its attributes to use for the cellvalues in the last row in the table	COL_TOTAL_STYLE= (p. 379)
Specify the style element and any changes to its attributes to use for the column variable values used as headers over the column variable value columns	COL_VAR_STYLE= (p. 379)
Specify the style element and any changes to its attributes to use for the cellvalues in the rightmost column of the last row in the table	GRAND_TOTAL_STYLE= (p. 379)
Specify a label for the table	LABEL= (p. 379)
Specify the style element and any changes to its attributes to use for the legend table that appears near the upper left corner of the table	LEGEND_STYLE= (p. 379)
Specify the name of the header to use over the row variable values (leftmost) column in the table	ROWS_HEADER= (p. 380)
Specify the style element and any changes to its attributes to use for the cellvalues in the cells that contain row totals	ROW_TOTAL_STYLE= (p. 380)
Specify the style element and any changes to its attributes to use for the row variable values in the leftmost column of the table	ROW_VAR_STYLE= (p. 380)
Specify a style element and any changes to its attributes to use for the table	STYLE= (p. 380)

CELL_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]> | *style-name*

specifies the style element and any changes to its attributes that you can use for the cellvalues in the non-summary rows and columns. This refers to the cellvalues that are not in the row totals column (see ROW_TOTAL_STYLE), the column totals row (see COL_TOTAL_STYLE), or the grand total cell (see GRAND_TOTAL_STYLE).

Default Data

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

COLS_HEADER=*header-name*
specifies the name of the header to use over the column variable value columns in the table.

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

COL_TOTAL_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element and any changes to its attributes to use for the cellvalues in the last row in the table.

Default Data

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

COL_VAR_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element and any changes to its attributes to use for the column variable values used as headers over the column variable value columns.

Default Header

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

GRAND_TOTAL_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element and any changes to its attributes to use for the cellvalues in the rightmost column of the last row in the table.

Default Data

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

LABEL="*text*"
specifies a label for the table.

Default "Frequency Counts and Percentages"

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

LEGEND_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element and any changes to its attributes to use for the legend table that appears near the upper left corner of the table.

Default Header

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

ROWS_HEADER=*header-name*
specifies the name of the header to use over the row variable values (leftmost) column in the table.

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

ROW_TOTAL_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element and any changes to its attributes to use for the cellvalues that contain row totals.

Default Data

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

ROW_VAR_STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
specifies the style element and any changes to its attributes to use for the row variable values in the leftmost column of the table.

Default RowHeader

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>
Specifies the style element and any changes to its attributes to use for the table.

style-element-name
is the name of the style element to use to display the table. The style element must be part of a style that is registered with the Output Delivery System. SAS provides some style. You can create customized styles with PROC TEMPLATE. For more information, see "DEFINE STYLE Statement" on page 454. By default, ODS produces different parts of ODS output with different elements. For example, by default, a table is produced with the style element Table. The Table style element that SAS is provides is uniquely designed to describe elements necessary to a table. However, you might have a user-defined style element at your site that would be appropriate to specify.

The style element provides the basis for displaying the table. Additional style attributes that you provide can modify the display.

style-element-name is either the name of a style element or a variable whose value is a style element.

See "Working with Styles " on page 438

"Finding and Viewing the Default Style for ODS Destinations" on page 438

"Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

style-attribute-specification
describes the style attribute to change.

Each *style-attribute-specification* has this general form:

See For information about the style attributes, see Chapter 21, "Style Attributes," on page 807.

"Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

Default Table

See "Crosstabulation Table Regions and Corresponding Attributes" on page 396 to see an illustration of the crosstabulation table regions and the DEFINE CROSSTABS attributes that affect each region.

DEFINE FOOTER Statement

Creates a footer template.

Note: The DEFINE FOOTER statement begins a footer template block. The following statements are commonly used within a DEFINE FOOTER block: "DYNAMIC Statement" on page 388, "NOTES Statement" on page 391, and "TEXT Statement" on page 391.

See: The substatements in DEFINE FOOTER and the footer attributes are the same as the substatements in DEFINE HEADER and the header attributes. For details about substatements and footer attributes, see the "DEFINE HEADER Statement" on page 381.

"Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

DEFINE FOOTER *symbol*;
<*attribute-1*;>< *attribute-n*>;
DYNAMIC *variable-1*<'*text-1*'> < *variable-n*<'*text-n*'>>;
NOTES '*text*';
TEXT *header-specification* </ *expression*>;
END;

DEFINE HEADER Statement

Creates a header template.

Note: The DEFINE HEADER statement begins a header template block. The following statements are commonly used within a DEFINE HEADER block: "DYNAMIC Statement" on page 388, "NOTES Statement" on page 391, and "TEXT Statement" on page 391.

See: "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

DEFINE HEADER *symbol*;
<*attribute-1*;>< *attribute-n*>;
DYNAMIC *variable-1*<'*text-1*'> < *variable-n*<'*text-n*'>>;
NOTES '*text*';
TEXT *header-specification* </ *expression*>;
END;

Summary of Optional Arguments

CINDENT='*character*'
Specify alignment for headers and footers that wrap

SPACE=*positive-integer*
Specify the number of blank lines to place between the current header and the next header or between the current footer and the previous footer

STYLE=<[*style-element-specification(s)*]>
Specify the style element and any changes to its attributes to use for the header or footer

Required Argument

symbol
specifies a name to be referenced by the HEADER statement.

DEFINE HEADER and DEFINE FOOTER Attribute Statements

This section lists the attributes that you can use in a header or footer template.

Table 12.4 *DEFINE HEADER and DEFINE FOOTER Attribute Statements*

Task	Attribute
Specify alignment for headers and footers that wrap	"CINDENT='*character*' " (p. 383)
Specify the number of blank lines to place between the current header and the next header or between the current footer and the previous footer	"SPACE=*positive-integer* " (p. 383)

Task	Attribute
Specify the style element and any changes to its attributes to use for the header or footer	"STYLE=<[*style-element-specification(s)*]>" (p. 383)

CINDENT='*character*'
specifies alignment for headers or footers that wrap. If a header or footer is too wide to fit on a single line, insert the specified character at the column position at which the second and subsequent lines should start. The first use of the CINDENT character determines the column position. For example, the following TEXT statement makes wrapped lines start at the same column as the open parenthesis:

```
text _COL_NAME_ "(;" _COL_LABEL_ ")"; CINDENT=';';
```

SPACE=*positive-integer*
specifies the number of blank lines to place between the current header and the next header or between the current footer and the previous footer.

Default 0 for headers and 1 for footers

Tip The SPACE= attribute is valid only in the LISTING destination.

Example "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

STYLE=<[*style-element-specification(s)*]>
specifies the style element and any changes to its attributes to use for the current column. Neither *style-attribute-specification* nor *style-element-name* is required. However, you must use at least one of them. You can use braces ({ and }) instead of square brackets ([and]).

style-element-name
is the name of the style element to use to display the data in the column. The style element must be part of a style template that is registered with the Output Delivery System. SAS provides some styles.

You can create customized styles with PROC TEMPLATE. For details, see "DEFINE STYLE Statement" on page 454. By default, ODS produces different parts of ODS output with different elements. For example, by default, a table header is displayed with the style element Header. The style elements that you would most likely use with the STYLE= attribute for a table header are as follows:

- Header
- HeaderFixed
- HeaderEmpty
- HeaderEmphasis
- HeaderEmphasisFixed
- HeaderStrong
- HeaderStrongFixed

The style elements that you would most likely use with the STYLE= attribute for a footer are as follows:

- Footer
- FooterFixed
- FooterEmpty
- FooterEmphasis
- FooterEmphasisFixed
- FooterStrong
- FooterStrongFixed

The style element provides the basis for displaying the header or footer. Additional style attributes that you provide can modify the display.

For more information, see "Working with Styles " on page 438.

style-element-name is either the name of a style element or a variable whose value is a style element.

style-attribute-specification
describes the style attribute to change.

Each *style-attribute-specification* has this general form:

style-attribute-name=< | >*style-attribute-value*

style-attribute-name
is the name of an attribute that is listed in "Style Attributes Tables" on page 808, or it is the name of a user-defined style attribute.

Tip If *style-attribute-name* refers to a user-defined attribute, then enclose the name in quotation marks. If *style-attribute-name* refers to an attribute that is listed in "Style Attributes Tables" on page 808, then do not enclose the name in quotation marks.

style-attribute-value
assigns the value to the attribute. If an attribute from the table in "Style Attributes Tables" on page 808 is specified, then specify the type of value that the attribute expects.

For more information about style-attribute values, see Chapter 21, "Style Attributes," on page 807.

|
prevents the style attribute from being inherited by any child style elements.

For information about the style attributes that you can specify, see Chapter 21, "Style Attributes," on page 807.

Tips The STYLE= attribute is valid only in the markup family, printer family, and RTF destinations.

If you use the STYLE= attribute inside a quoted string, then add a space before or after the carriage return to prevent errors. SAS does not interpret a carriage return as a space. You must explicitly specify spaces in quoted strings.

Example "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

CELLSTYLE AS Statement

Sets the style element of the cells in the column according to the values of the variables. Use this statement to set the presentation characteristics (such as foreground color, font face, and flyover) of individual cells in all destinations except the LISTING destinations.

Restriction: The CELLSTYLE AS statement can be used only with the DEFINE CELLVALUE statement.

See: "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*] >
<, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*]>>;

Required Argument

expression
is an expression that is evaluated for each cell. If *expression* resolves to TRUE (a nonzero value), the style element that is specified is used for the current cell. If *expression* is FALSE (zero), the next expression in the statement is evaluated. Thus, you can string multiple expressions together to format cells conditionally.

expression has this form:

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation.

An operand is one of the following:

1
is a fixed value that you can use to set a constant style element.

Example These statements set the cellvalue Frequency background to gray:

```
define cellvalue Frequency;
other-statements ...;
cellstyle 1 as {backgroundcolor=gray};
end;
```

VAL
is the value of the current cell.

Example The following statements change the foreground color of the cellvalue Percent depending on its magnitude:

```
define cellvalue Percent;
other-statements ...;
```

```
cellstyle _val_ > 75.00 as {color=red},
_val_ > 50.00 as {color=orange}, _val_ > 25.00 as {color=green};
end;
```

comparison-operator
compares a variable with a value or with another variable.

Table 12.5 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Tip Using an expression of 1 as the last expression in the CELLSTYLE AS statement sets the style element for any cells that did not meet an earlier condition.

Optional Arguments

style-attribute-specification
describes a style attribute to set.

Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

Default If you do not specify any style attributes to modify, ODS uses the unmodified *style-element-name*.

Note Neither *style-attribute-specification* nor *style-element-name* is required. However, you must use at least one of them.

See For information about the style attributes that you can set in a column template, see Chapter 21, “Style Attributes,” on page 807.

style-element-name
is the name of the style element that displays the data in the column. The style element must be part of a style that is registered with the Output Delivery System. SAS provides some styles. You can create customized styles by using PROC TEMPLATE. For more information, see “DEFINE STYLE Statement” on

page 454. By default, ODS displays different parts of ODS output with different style elements.

For example, by default, the data in a column is displayed with the style element Data. The style elements that you would probably use with the CELLSTYLE AS statement in a column template are the following:

- Data
- DataFixed
- DataEmpty
- DataEmphasis
- DataEmphasisFixed
- DataStrong
- DataStrongFixed

The style element provides the basis for displaying the column. Additional style attributes that you provide can modify the display.

Default Data

Note Neither *style-attribute-specification* nor *style-element-name* is required. However, you must use at least one of them.

See "Working with Styles " on page 438

"Finding and Viewing the Default Style for ODS Destinations" on page 438

CELLVALUE Statement

Specifies the order in which the cellvalues are stacked in the cells.

Interaction: If a cellvalue symbol that was specified by the DEFINE CELLVALUE statement is not present in the list, it does not appear in the crosstabulation table.

See: "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

CELLVALUE *cellvalue(s)*;

Required Argument

cellvalue(s)
specifies one of the nine possible cellvalues created by the DEFINE CELLVALUE statement. *cellvalues* are ordered from the top to the bottom.

See "DEFINE CELLVALUE Statement" on page 372

DYNAMIC Statement

Defines a symbol that references a value that the data component supplies from the procedure or DATA step.

Restriction: The DYNAMIC statement can be used only with the DEFINE CELLVALUE, DEFINE HEADER, and DEFINE FOOTER statements.

Tip: A dynamic variable that is defined in a template is available to that template and all the templates that it contains.

Example: "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

DYNAMIC *dynamic-variable(s)*;

Required Argument

dynamic- variable(s)
is a variable that is defined by SAS in the crosstabulation template. After a dynamic variable has been defined, you can use it in the TEXT statement within a footer or header template.

FMISSING
is the number of missing values in the table.

Requirement The FMISSING dynamic variable must be specified by the DYNAMIC statement before you can use the dynamic variable in an expression.

NOTITLE
is set to 1 if the PROC FREQ's NOTITLE option was used, and it is set to 0 if the NOTITLE option was not used.

Requirement The NOTITLE dynamic variable must be specified by the DYNAMIC statement before you can use the dynamic variable in an expression.

SAMPLESIZE
is set to 0 if the table is empty. Otherwise, it is set to 1.

Requirement The SAMPLESIZE dynamic variable must be specified by the DYNAMIC statement before you can use the dynamic variable in an expression.

STRATNUM
is the current stratum number if the table has multiple strata. If the table has only one stratum, then the value is 0.

Requirement The STRATNUM dynamic variable must be specified by the DYNAMIC statement before you can use the dynamic variable in an expression.

Example "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

STRATAVARIABLENAMES
is a string that identifies the current stratum by the name of the stratum variables.

var-1=value-1<var-n=value-n>

var-1–var-n
specifies the stratum variables.

value-1–value-n
specifies the values of the stratum variables.

Requirement The DYNAMIC statement must specify the STRATAVARIABLELABELS dynamic variables before the dynamic variables can be used in an expression.

Tip The value is undefined if the table has only one stratum.

STRATAVARIABLELABELS
is a string that identifies the current stratum by the label of the stratum variables.

var-1=value-1<var-n=value-n>

var-1–var-n
specifies the stratum variables.

value-1–value-n
specifies the values of the stratum variables.

Tip The value is undefined when the table has only one stratum.

Requirement The DYNAMIC statement must specify the STRATAVARIABLELABELS dynamic variables before you can use the dynamic variables in an expression.

Examples "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

"Example 2: Creating a Crosstabulation Table Template with a Customized Legend" on page 409

END Statement

Ends the crosstabulation template or a DEFINE CELLVALUE, DEFINE HEADER, or DEFINE FOOTER code block.

Restriction: The END statement must be used with the DEFINE CELLVALUE, DEFINE HEADER, DEFINE FOOTER, and DEFINE CROSSTABS statements.

See: "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

END;

FOOTER Statement

Declares a symbol as a footer in the table and specifies the order of the footers.

Restriction: The FOOTER statement can be used only within a crosstabulation table template.

Syntax

FOOTER *footer-specification(s)*;

Required Argument

footer-specification(s)
specifies a symbol defined by the DEFINE FOOTER statement within the same table template.

Default If you omit a FOOTER statement, ODS creates a footer for each footer template (DEFINE FOOTER statement) and places the footers in the same order that the footer templates have in the table template.

See "DEFINE FOOTER Statement" on page 381

HEADER Statement

Declares a symbol as a header in the table and specifies the order of the headers.

Restriction: The HEADER statement can be used only within a crosstabulation table template.

Syntax

HEADER *header-specification(s)*;

Required Argument

header-specification(s)
specifies a symbol defined by the DEFINE HEADER statement within the same table template.

Default If you omit a HEADER statement, then ODS makes a header for each header template (DEFINE HEADER statement) and places the headers in the same order that the header templates have in the table template.

NOTES Statement

Provides information about a template.

Restriction: The NOTES statement can be used only with the DEFINE CROSSTABS, DEFINE CELLVALUE, and DEFINE HEADER statements.

Tip: The NOTES statement becomes part of the compiled template, which you can view with the SOURCE statement. SAS comments do not become part of the compiled template.

Syntax

NOTES '*text*';

Required Argument

'text'
provides information about the template.

TEXT Statement

Specifies the text of the header or the footer.

Restriction: TheTEXT statement can be used only with the DEFINE HEADER or DEFINE FOOTER statements.

Example: “Example 1: Creating a Customized Crosstabulation Table Template with No Legend” on page 397

Syntax

TEXT *header-specification(s)* </ *options*>;

Required Argument

header-specification(s)
specifies the text of the header.

header-specification(s) can be any dynamic variable that is specified by the DYNAMIC statement, or it can be one of the following:

dynamic-variable
is a variable that is automatically defined by SAS in the crosstabulation template.

dynamic-variable can be one of the following:

_COL_LABEL_
is the label of the column variable, which is the last variable in a table request.

_COL_NAME_
is the name of the column variable, which is the last variable in a table request. If the column variable does not have a name, then the value of _COL_LABEL_ is an empty text string (" ").

_ROW_LABEL_
is the label of the row variable, which is the next to the last variable in a table request. If the row variable does not have a label, the value of _ROW_LABEL_ is an empty text string (" ").

_ROW_NAME_
is the name of the row variable, which is the next to the last variable in a table request.

text-specification(s)
specifies the text to use in the header.

Each *text-specification* is one of the following:

- a quoted string.
- a variable followed by an optional format. The variable is any variable that is declared in a DYNAMIC statement or is any of the variables above.

Tip If the quoted string is a blank and it is the only item in the header specification, the header is a blank line.

Optional Argument

expression
is an expression that is evaluated for a header or footer. If *expression* is omitted, the default is 1. Each DEFINE HEADER statement can contain any number of TEXT statements. The template evaluates each expression in turn from top to bottom and thereby determines the text of the header. The *header-specification* in the first TEXT statement whose expression evaluates to true becomes the header text. After an expression evaluates to true, the template examines no more TEXT statements. If no expression is true, then the header is not used.

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation.

An operand is one of the following:

constant
is a fixed value, such as a number or text string.

dynamic variable
is a variable that is defined by SAS in the crosstabulation template or by the DYNAMIC statement within a header or footer template.

_COL_LABEL_
specifies the label of the column variable, which is the last variable in a table request. If the column variable does not have a label, then the value of _COL_LABEL_ is an empty text string (" ").

_COL_NAME_
specifies the name of the column variable, which is the last variable in a table request. If the column variable does not have a name, then the value of _COL_NAME_ is an empty text string (" ").

FMISSING
is the number of missing values in the table.

Requirement The FMISSING dynamic variable must be specified by the DYNAMIC statement before you can use it in an expression.

NOTITLE
is set to 1 if PROC FREQ's NOTITLE option was used, and it is set to 0 if the NOTITLE option was not used.

Requirement The NOTITLE dynamic variable must be specified by the DYNAMIC statement before you can use it in an expression.

_ROW_LABEL_
is the label of the row variable, which is the next to the last variable in a table request. If the row variable does not have a name, then the value of _ROW_LABEL_ is an empty text string (" ").

_ROW_NAME_
specifies the name of the row variable, which is the next to the last variable in a table request. If the row variable does not have a name, then the value of _ROW_NAME_ is an empty text string (" ").

SAMPLESIZE
is set to 0 if the table is empty. Otherwise, it is set to 1.

Requirement The SAMPLESIZE dynamic variable must be specified by the DYNAMIC statement before you can use it in an expression.

STRATNUM
is the current stratum number if the table has multiple strata. If the table has only one stratum, then the value is 0.

Requirement The STRATNUM dynamic variable must be specified by the DYNAMIC statement before you can use it in an expression.

STRATAVARIABLENAMES
is a string that identifies the current stratum by the name of the stratum variables.

var-1=value-1<var-n=value-n>

var-1–var-n
specifies the stratum variables.

value-1–value-n
specifies the values of the stratum variables.

Tip The value is undefined when the table has only one stratum.

Requirement The STRATAVARIABLENAMES dynamic variable must be specified by the DYNAMIC statement before you can use it in an expression.

STRATAVARIABLELABELS
is a string that identifies the current stratum by the label of the stratum variables.

STRATAVARIABLELABELS has the following form:

var-1=value-1<*var-n*=value-n?>

var-1–var-n
specifies the stratum variables.

value-1–value-n
specifies the values of the stratum variables.

Tip The value is undefined when the table has only one stratum.

Requirement The DYNAMIC statement must specify the STRATAVARIABLELABELS dynamic variables before you can use them in an expression.

comparison-operator
compares a variable with a value or another variable.

Table 12.6 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than

Symbol	Mnemonic Equivalent	Definition
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Usage: TEMPLATE Procedure: Creating Crosstabulation Table Templates

Working with the CrossTabFreqs Crosstabulation Table Template

When creating your own crosstabulation table template, you always define the new table with the same name as the existing table, which is Base.Freq.CrossTabFreqs. By default, the existing crosstabulation table that PROC FREQ creates is stored in the Sashelp.Tmplmst template store.

With PROC TEMPLATE, you can create a modified version of Base.Freq.CrossTabFreqs that you can save in a different template store by using the ODS PATH statement. All crosstabulation templates must have the same name. If you want to have multiple crosstabulation templates, put each one in a different template store. Then you can use the ODS PATH statement to add the template store that contains the version of the crosstabulation template that you want to use.

For example, suppose that you have a crosstabulation template in the template store Corporat.Template and another crosstabulation template in Govment.Templat. In the following code, the first ODS PATH statement adds the template store Corporat.Templat. The first PROC FREQ code is then formatted using the crosstabulation table template from Corporat.Templat. The second ODS PATH statement removes Corporat.Templat, and the third ODS PATH statement adds Govment.Templat. The last PROC FREQ step then uses the crosstabulation template from Corporat.Templat.

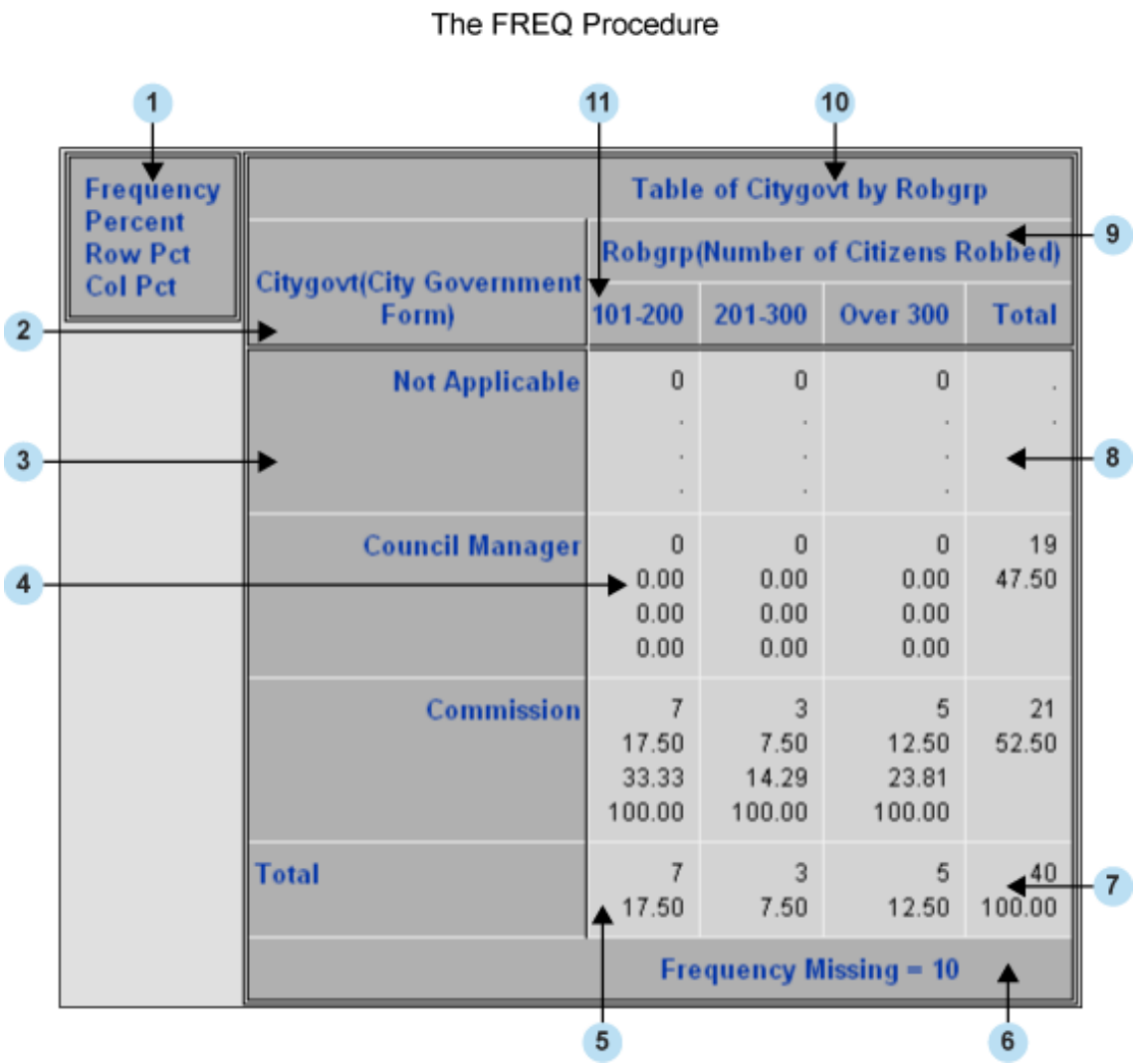
```
ods path(prepend) corporat.templat(read);
... proc freq code ...
ods path(remove) corporat.templat;
ods path(prepend) govment.templat;
... proc freq code ...
```

For more information about the ODS PATH statement, see "ODS PATH Statement" in *SAS Output Delivery System: User's Guide*.

Crosstabulation Table Regions and Corresponding Attributes

When creating a crosstabulation template, you can use attributes to modify individual table regions. The following figure and corresponding table identify the different parts of the crosstabulation table and the attributes that control the style of each part.

Figure 12.3 *Crosstabulation Table Regions That Can Be Modified*



Most regions use DEFINE CROSSTABS style attributes to specify a style. The following table shows the style attribute that affects each table region. For complete documentation on DEFINE CROSSTABS attributes, see "DEFINE CROSSTABS Attributes" on page 377. Headers and footers use the STYLE= attribute that is valid for the DEFINE HEADER and DEFINE FOOTER statements. For information about

the STYLE= attribute, see “DEFINE HEADER and DEFINE FOOTER Attribute Statements” on page 382.

Table 12.7 *Table Region and Corresponding Style Attribute*

Item	Crosstabulation Table Region	Style Attribute
1	Legend	LEGEND_STYLE=
2	Row variable name	ROWS_HEADER=
3	Row variable value	ROW_VAR_STYLE=
4	Data cell	CELL_STYLE=
5	Column total	COL_TOTAL_STYLE=
6	Footer	STYLE=
7	Grand total	GRAND_TOTAL_STYLE=
8	Row total	ROW_TOTAL_STYLE
9	Column variable name	COLS_HEADER=
10	Header	STYLE=
11	Column variable value	COL_VAR_STYLE=

Examples: TEMPLATE Procedure: Creating Crosstabulation Table Templates

Example 1: Creating a Customized Crosstabulation Table Template with No Legend

Features: *crosstabs-attributes* statements
CELLVALUE statement

DEFINE CELLVALUE statement
- CELLSTYLE AS statement
- END statement
- FORMAT= attribute
- HEADER= attribute
- LABEL= attribute

DEFINE HEADER statement
- END statement
- SPACE= attribute
- STYLE= attribute
- TEXT statement

DEFINE FOOTER statement
- END statement
- DYNAMIC statement
- SPACE= attribute
- STYLE= attribute
- TEXT statement

END statement

FOOTER statement

HEADER statement

NOTES statement

Other ODS features
- ODS HTML statement
- ODS PATH statement
- DEFINE STYLE statement

Details

The following example creates the crosstabulation table template Base.Freq.CrossTabFreqs. The template has the following features:

- footnote used to display cellvalue labels instead of a legend
- modified headers and footers
- variable labels used in headers
- modified table regions

Program

```
Proc Format;
   Value Govtfmt -3='Council Manager'
                  0='Commission'
                  3='Mayor Council'
                 .N='Not Applicable'
                  .='   ?';
   Value Robfmt   1='100 or  Less'
                  2='101-200'
                  3='201-300'
```

```
                4='Over 300'
               .N='Not     Known'
                .='   ?';
   Value Colfg  1='yellow'
                2='red'
                3='blue'
                4='purple'
               .N='green'
                .='black'
                other='black';
   Value Rowfg  -3='red'
                0='purple'
                3='blue'
               .N='green'
                .='black'
                other='black';

run;

data gov;
   Label Citygovt='City Government Form'
         Robgrp='Number of Meetings Scheduled';
   Input Citygovt Robgrp Weight;  Missing N;
   Format Citygovt Govtfmt. Robgrp Robfmt.;
   LOOP: OUTPUT; WEIGHT=WEIGHT-1; IF WEIGHT>0 THEN GOTO LOOP;
   DROP WEIGHT;
datalines;
0 1  6
0 3  3
0 2  7
0 4  5
N N 10
-3 1 47
-3 3 49
-3 2 63
-3 4 52
. 2  1
3 1 31
3 2 37
3 3 27
3 4 55
3 .  1
;
ods path (prepend) work.templat(update);
ods noproctitle;


proc template;
   define style white;
      parent=styles.htmlblue;
      style body /
         backgroundcolor=white;
      style systemtitle /
         backgroundcolor=white
         fontsize=6
         fontweight=bold
```

```
         fontstyle=italic;
      style systemfooter /
         backgroundcolor=white
         fontsize=2
         fontstyle=italic;
      style proctitle /
         backgroundcolor=white
         color=#6078bf
         fontweight=bold
         fontstyle=italic;
    end;

   define crosstabs Base.Freq.CrossTabFreqs;
      notes "Crosstabulation table";

      style=table {backgroundcolor=#BFCFFF};
      cell_style=data {backgroundcolor=#FFFFF0};
      row_var_style=rowheader {backgroundcolor=#BFCFFF  color=rowfg.};
      col_var_style=header {backgroundcolor=#BFCFFF  color=colfg.};
      row_total_style=data {backgroundcolor=#F0F0F0};
      col_total_style=data {backgroundcolor=#F0F0F0};
      grand_total_style=datastrong {backgroundcolor=#F0F0F0};
      legend_style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};

      rows_header=RowsHeader cols_header=ColsHeader;
      label = "Frequency Counts and Percentages";

      define header TableOf;
            text "Table of " _ROW_LABEL_ " by " _COL_LABEL_ /
_ROW_LABEL_ ^= ''
             & _COL_LABEL_ ^= '';
            text "Table of " _ROW_LABEL_ " by " _COL_NAME_  /
_ROW_LABEL_ ^= '';
            text "Table of " _ROW_NAME_  " by " _COL_LABEL_ /
_COL_LABEL_ ^= '';
            text "Table of " _ROW_NAME_  " by " _COL_NAME_;
         style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};
         end;

      define header RowsHeader;
         text _ROW_LABEL_ / _ROW_LABEL_ ^= '';
         text _ROW_NAME_;
         style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};
         space=0;
         end;

      define header ColsHeader;
         text _COL_LABEL_ / _COL_LABEL_ ^= '';
         text _COL_NAME_;
         style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};
         space=1;
         end;

   define header ControllingFor;
      dynamic StratNum StrataVariableNames StrataVariableLabels;
      text "Controlling for" StrataVariableNames / StratNum > 0;
```

```
      style=header;
      end;

      define footer Missing;
         dynamic FMissing;
         text "Frequency Missing = " FMissing -12.99 / FMissing ^= 0;
         style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};
         space=1;
         end;

      define footer NoObs;
         dynamic SampleSize;
         text "Effective Sample Size = 0" / SampleSize = 0;
         space=1;
         style=header;
         end;


      define cellvalue Frequency;
         header="";
         label="Frequency Count";
         format=BEST7.; data_format_override=on; print=on;
         cellstyle _val_ < 10 as datastrong {color=green},
                   _val_ > 40 & _val_ < 50 as datastrong
{color=orange},
                   _val_ >= 50 as datastrong {color=red};
         end;

      define cellvalue Expected;
         header="";
         label="Expected Frequency";
         format=BEST6. data_format_override=on print=on;
         end;

      define cellvalue Deviation;
         header="";
         label="Deviation from Expected Frequency";
         format=BEST6. data_format_override=on print=on;
         end;

      define cellvalue CellChiSquare;
         header="";
         label="Cell Chi-Square";
         format=BEST6. print=on;
         end;

      define cellvalue TotalPercent;
         header="";
         label="Percent of Total Frequency";
         format=6.2 print=on;
         end;

      define cellvalue Percent;
         header="";
         label="Percent of Two-Way Table Frequency";
         format=6.2 print=on;
```

```
      end;

   define cellvalue RowPercent;
      header="";
      label="Percent of Row Frequency";
      format=6.2 print=on;
      end;

   define cellvalue ColPercent;
      header="";
      label="Percent of Column Frequency";
      format=6.2 print=on;
      end;

   define cellvalue CumColPercent;
      header="";
      label="Cumulative Percent of Column Frequency";
      format=6.2 print=on;
      end;

   cellvalue
      Frequency Expected Deviation
      CellChiSquare TotalPercent Percent
      RowPercent ColPercent CumColPercent;

   header TableOf ControllingFor;
   footer NoObs Missing;
   end;

ods html file='MyCrosstabsTable.html' style=white;

title "City Government Form by Number of Meetings Scheduled";
footnote "Cellvalues are stacked in the following order:";
footnote2  "Frequency";
footnote3 "Percent";
footnote4 "Row Percent";
footnote5 "Column Percent";
ods noproctitle;

proc freq;
    tables citygovt*robgrp / missprint;
run;

ods html close;
```

Program Description

Create the user-defined formats and create the data set. The FORMAT procedure creates two user-defined formats that can be used in the crosstabulation template. The DATA step creates the Gov data set.

```
Proc Format;
   Value Govtfmt -3='Council Manager'
                  0='Commission'
                  3='Mayor Council'
                 .N='Not Applicable'
                  .='   ?';
```

```
   Value Robfmt   1='100 or  Less'
                  2='101-200'
                  3='201-300'
                  4='Over 300'
                 .N='Not     Known'
                  .='   ?';
   Value Colfg   1='yellow'
                  2='red'
                  3='blue'
                  4='purple'
                 .N='green'
                  .='black'
                  other='black';
   Value Rowfg   -3='red'
                  0='purple'
                  3='blue'
                 .N='green'
                  .='black'
                  other='black';

run;

data gov;
   Label Citygovt='City Government Form'
        Robgrp='Number of Meetings Scheduled';
   Input Citygovt Robgrp Weight;  Missing N;
   Format Citygovt Govtfmt. Robgrp Robfmt.;
   LOOP: OUTPUT; WEIGHT=WEIGHT-1; IF WEIGHT>0 THEN GOTO LOOP;
   DROP WEIGHT;
datalines;
0 1  6
0 3  3
0 2  7
0 4  5
N N 10
-3 1 47
-3 3 49
-3 2 63
-3 4 52
. 2  1
3 1 31
3 2 37
3 3 27
3 4 55
3 .  1
;
```

Establish the ODS path and create the White style. The ODS PATH statement specifies the locations to write to or read from when creating the PROC TEMPLATE templates. The PROC TEMPLATE statement, DEFINE STYLE statement, and collection of STYLE statements create the style template White. The ODS NOPROCTITLE statement suppresses the writing of the title of the FREQ procedure.

```
ods path (prepend) work.templat(update);
ods noproctitle;
```

```
proc template;
   define style white;
      parent=styles.htmlblue;
      style body /
         backgroundcolor=white;
      style systemtitle /
         backgroundcolor=white
         fontsize=6
         fontweight=bold
         fontstyle=italic;
      style systemfooter /
         backgroundcolor=white
         fontsize=2
         fontstyle=italic;
      style proctitle /
         backgroundcolor=white
         color=#6078bf
         fontweight=bold
         fontstyle=italic;
    end;
```

Create the crosstabulation template Base.Freq.CrossTabFreqs. The DEFINE statement creates the crosstabulation template Base.Freq.CrossTabFreqs in the first template store in the path for which you have Write access (Work, in this example). The NOTES statement provides information about the crosstabulation table.

```
define crosstabs Base.Freq.CrossTabFreqs;
   notes "Crosstabulation table";
```

Change the appearance of individual table regions. The following DEFINE CROSSTABS statement attributes modify the appearance of individual table regions. Each attribute corresponds to a specific region of the table.

To see which attribute corresponds to which table region, see “Crosstabulation Table Regions and Corresponding Attributes” on page 396 .

```
      style=table {backgroundcolor=#BFCFFF};
      cell_style=data {backgroundcolor=#FFFFF0};
      row_var_style=rowheader {backgroundcolor=#BFCFFF  color=rowfg.};
      col_var_style=header {backgroundcolor=#BFCFFF  color=colfg.};
      row_total_style=data {backgroundcolor=#F0F0F0};
      col_total_style=data {backgroundcolor=#F0F0F0};
      grand_total_style=datastrong {backgroundcolor=#F0F0F0};
      legend_style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};
```

Specify a row header, a column header, and a label for the table. The ROWS_HEADER= style attribute specifies RowsHeader as the header for rows. The COLS_HEADER= style attribute specifies ColsHeader as the header for columns. The LABEL= attribute specifies a label for the crosstabulation template. The label appears in the Results window.

```
      rows_header=RowsHeader cols_header=ColsHeader;
      label = "Frequency Counts and Percentages";
```

Create the TableOf header template. The DEFINE HEADER statement and its attributes create the header template TableOf, which is specified by the HEADER statement later on in the program. The TEXT statement specifies the text of the header by using dynamic variables that represent label variables and names. The

TEXT statements also use expressions to determine whether row labels and column labels are assigned to the row and column variables. Only TEXT statements that have true expressions are displayed in the output. In this example, both the row label and the column label exist. Therefore, the first TEXT statement is used and the text resolves to: "Table of City Government Form by Number of Meetings Scheduled". The STYLE= attribute specifies style information for the header.

```
        define header TableOf;
              text "Table of " _ROW_LABEL_ " by " _COL_LABEL_ /
_ROW_LABEL_ ^= ''
               & _COL_LABEL_ ^= '';
              text "Table of " _ROW_LABEL_ " by " _COL_NAME_  /
_ROW_LABEL_ ^= '';
              text "Table of " _ROW_NAME_  " by " _COL_LABEL_ /
_COL_LABEL_ ^= '';
              text "Table of " _ROW_NAME_  " by " _COL_NAME_;
           style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};
           end;
```

Create the RowsHeader header template. The DEFINE HEADER statement creates the header RowsHeader. RowsHeader is specified as a row heading by the preceding ROWS_HEADER= style attribute. The TEXT statements specify the text of the header by using dynamic variables that represent label variables and names. The first TEXT statement uses an expression to determine whether a label is assigned to the variable. If there is no label, the next TEXT statement, which specifies the row name, is used. In this example, there is a row label for the row variable, so in the output, _ROW_LABEL_ resolves to "City Government Form". The STYLE= attribute specifies style information for the header, and the SPACE attribute specifies that the current header and the previous header should have one blank line between them.

```
        define header RowsHeader;
           text _ROW_LABEL_ / _ROW_LABEL_ ^= '';
           text _ROW_NAME_;
           style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};
           space=0;
           end;
```

Create the ColsHeader header template. The DEFINE HEADER statement creates the header ColsHeader. ColsHeader is specified as a column heading by the preceding COLS_HEADER= style attribute. The TEXT statements specify the text of the header by using dynamic variables that represent label variables and names. The first TEXT statement uses an expression to determine whether a label is assigned to the column variable. If there is no label, the next TEXT statement, which specifies the row name, is used. In this example, there is a column label, so in the output, _COL_LABEL_ resolves to "Number of Meetings Scheduled". The STYLE= attribute specifies style information for the header, and the SPACE attribute specifies that the current header and the previous header should have one blank line between them.

```
        define header ColsHeader;
           text _COL_LABEL_ / _COL_LABEL_ ^= '';
           text _COL_NAME_;
           style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};
           space=1;
           end;
```

Create the ControllingFor header template. The DEFINE HEADER statement and its attributes create the header template ControllingFor. The DYNAMIC statement declares dynamic variables so that they can be used in expressions. The TEXT statement specifies the text of the header by using dynamic variables that represent label variables and names. In this example, the expression in the TEXT statement resolves to false, so the ControllingFor header does not show up in the output. The STYLE= attribute specifies style information for the headers.

```
define header ControllingFor;
   dynamic StratNum StrataVariableNames StrataVariableLabels;
   text "Controlling for" StrataVariableNames / StratNum > 0;
   style=header;
   end;
```

Create footer templates. Each of these DEFINE FOOTER statements and its attributes creates a footer template. For the footers to show up in the output, they must be specified by the FOOTER statement. The DYNAMIC statements declare the dynamic variables FMissing and SampleSize, so that they can be used in the TEXT statements. The TEXT statements conditionally select text to use as footers. In the first TEXT statement, the expression is true, because FMissing is not 0. Therefore, the first TEXT statement is displayed in the output. In the second TEXT statement, the expression resolves to false, so the NoObs footer does not appear in the output. The STYLE attribute specifies style information for the footers, and the SPACE attribute specifies that the current footer and the previous footer should have one blank line between them.

```
define footer Missing;
   dynamic FMissing;
   text "Frequency Missing = " FMissing -12.99 / FMissing ^= 0;
   style=header {backgroundcolor=#BFCFFF color=black
fontstyle=italic};
   space=1;
   end;

define footer NoObs;
   dynamic SampleSize;
   text "Effective Sample Size = 0" / SampleSize = 0;
   space=1;
   style=header;
   end;
```

Create the cellvalue definitions. The DEFINE CELLVALUE statements define the values that will appear in the cells of the crosstabulation table. The HEADER= attribute specifies the text that appears in the legend. Because there is no text specified for any of these cellvalues, there is no legend in the output. The FORMAT= attribute specifies the format to use for the cellvalue. The DATA_FORMAT_OVERRIDE=ON attribute specifies to use the format specified in the data component. The PRINT=ON attribute specifies the cellvalue to appear in the table. The CELLSTYLE AS statement uses expressions to set the style element of the cells conditionally according to the values of the variables for the Frequency cellvalue. The _VAL_ variable represents the value of a cell. Therefore, in this example, if the value in a cell is less than ten, then the font color for the DataStrong style element is green. If the value in the cell is between 40 and 50, then the font color for the DataStrong style element is orange. If the value is greater than 50, then the font color is red.

```
define cellvalue Frequency;
   header="";
```

```
         label="Frequency Count";
         format=BEST7.; data_format_override=on; print=on;
         cellstyle _val_ < 10 as datastrong {color=green},
                   _val_ > 40 & _val_ < 50 as datastrong
{color=orange},
                   _val_ >= 50 as datastrong {color=red};
         end;

      define cellvalue Expected;
         header="";
         label="Expected Frequency";
         format=BEST6. data_format_override=on print=on;
         end;

      define cellvalue Deviation;
         header="";
         label="Deviation from Expected Frequency";
         format=BEST6. data_format_override=on print=on;
         end;

      define cellvalue CellChiSquare;
         header="";
         label="Cell Chi-Square";
         format=BEST6. print=on;
         end;

      define cellvalue TotalPercent;
         header="";
         label="Percent of Total Frequency";
         format=6.2 print=on;
         end;

      define cellvalue Percent;
         header="";
         label="Percent of Two-Way Table Frequency";
         format=6.2 print=on;
         end;

      define cellvalue RowPercent;
         header="";
         label="Percent of Row Frequency";
         format=6.2 print=on;
         end;

      define cellvalue ColPercent;
         header="";
         label="Percent of Column Frequency";
         format=6.2 print=on;
         end;

      define cellvalue CumColPercent;
         header="";
         label="Cumulative Percent of Column Frequency";
         format=6.2 print=on;
         end;
```

Specify which cellvalues appear in the table and the order in which the cellvalues are stacked in the cells. The CELLVALUE statement specifies which cellvalues appear in the output. In this example, all of the cellvalues that were created appear in the table. The CELLVALUE statement also specifies the order in which the cellvalues are stacked in the cells.

```
cellvalue
   Frequency Expected Deviation
   CellChiSquare TotalPercent Percent
   RowPercent ColPercent CumColPercent;
```

Specify which headers and footers appear in the output. The HEADER statement specifies which header templates are applied to your output. The FOOTER statement specifies which footer templates are applied to your output. In order for any of the headers and footers defined by a DEFINE statement to appear in your output, they must be specified by the FOOTER or HEADER statement.

```
header TableOf ControllingFor;
footer NoObs Missing;
end;
```

Create the HTML output and specify the name of the HTML file. The ODS HTML statement opens the HTML destination and creates HTML output. The STYLE= option specifies template White for the output style.

```
ods html file='MyCrosstabsTable.html' style=white;
```

Specify a title and footnote, and suppress the printing of the procedure title. The TITLE and FOOTNOTE statements specify titles and footnotes for the output. The ODS NOPROCTITLE statement prevents the printing of the FREQ procedure's title in the output.

```
title "City Government Form by Number of Meetings Scheduled";
footnote "Cellvalues are stacked in the following order:";
footnote2  "Frequency";
footnote3 "Percent";
footnote4 "Row Percent";
footnote5 "Column Percent";
ods noproctitle;
```

Create the crosstabulation table. The FREQ procedure creates a Citygovt by Robgrp crosstabulation table.

```
proc freq;
    tables citygovt*robgrp / missprint;
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination, as well as all the files that are open for that destination.

```
ods html close;
```

Output

Output 12.1 *Output Using Customized Crosstabulation Table Template*

City Government Form by Number of Meetings Scheduled

Table of City Government Form by Number of Meetings Scheduled							
	Number of Meetings Scheduled						
City Government Form	?	Not Known	100 or Less	101-200	201-300	Over 300	Total
?	0 . . .	0 . . .	0 . . .	1 . . .	0 . . .	0 . . .	. .
Not Applicable	0 . . .	10 . . .	0 . . .	0 . . .	0 . . .	0 . . .	. .
Council Manager	0 . . .	0 . . .	47 12.30 22.27 55.95	63 16.49 29.86 58.88	49 12.83 23.22 62.03	52 13.61 24.64 46.43	211 55.24
Commission	0 . . .	0 . . .	6 1.57 28.57 7.14	7 1.83 33.33 6.54	3 0.79 14.29 3.80	5 1.31 23.81 4.46	21 5.50
Mayor Council	1 . . .	0 . . .	31 8.12 20.67 36.90	37 9.69 24.67 34.58	27 7.07 18.00 34.18	55 14.40 36.67 49.11	150 39.27
Total	. .	. .	84 21.99	107 28.01	79 20.68	112 29.32	382 100.00
Frequency Missing = 12							

Cellvalues are stacked in the following order:
Frequency
Percent
Row Percent
Column Percent

Example 2: Creating a Crosstabulation Table Template with a Customized Legend

Features: *crosstabs-attributes* statements
CELLVALUE statement
DEFINE CELLVALUE statement
CELLSTYLE AS statement

END statement
FORMAT= attribute
HEADER= attribute
LABEL= attribute
DEFINE HEADER statement
END statement
SPACE= attribute
STYLE= attribute
TEXT statement
DEFINE FOOTER statement
END statement
DYNAMIC statement
SPACE= attribute
STYLE= attribute
TEXT statement
END statement
FOOTER statement
HEADER statement
NOTES statement
Other ODS features
ODS HTML statement
ODS PATH statement
DEFINE STYLE statement

Details

The following example creates a new crosstabulation table template for the CrossTabFreqs table. The template has the following features:

- a legend with customized text
- modified headers and footers
- variable labels used in headers
- modified table regions
- customized styles for cellvalues

Program

```
Proc Format;
  Value Govtfmt -3='Council Manager'
                 0='Commission'
                 3='Mayor Council'
                .N='Not Applicable'
                 .='   ?';
  Value Robfmt   1='100 or  Less'
                 2='101-200'
                 3='201-300'
```

```
                 4='Over 300'
                .N='Not     Known'
                 .='   ?';
   Value colfg   1='yellow'
                 2='red'
                 3='blue'
                 4='purple'
                .N='green'
                 .='black'
                 other='black';
   Value rowfg   -3='red'
                 0='purple'
                 3='blue'
                .N='green'
                 .='black'
                 other='black';
run;

data gov;
   Label Citygovt='City Government Form'
        Robgrp='Number of Meetings Scheduled';
   Input Citygovt Robgrp Weight;  Missing N;
   Format Citygovt Govtfmt. Robgrp Robfmt.;
   LOOP: OUTPUT; WEIGHT=WEIGHT-1; IF WEIGHT>0 THEN GOTO LOOP;
   DROP WEIGHT;
datalines;
0 1  6
0 3  3
0 2  7
0 4  5
N N 10
-3 1 47
-3 3 49
-3 2 63
-3 4 52
. 2  1
3 1 31
3 2 37
3 3 27
3 4 55
3 .  1
;

ods path (prepend) work.templat(update);

proc template;
   define crosstabs Base.Freq.CrossTabFreqs;
      notes "Crosstabulation table with legend";

      rows_header=RowsHeader cols_header=ColsHeader;
      label = "Frequency Counts and Percentages";
      grand_total_style=data {fontweight=bold};

   define header ControllingFor;
      dynamic StratNum StrataVariableNames StrataVariableLabels;
      text "Controlling for" StrataVariableNames / StratNum > 0;
      style=header;
      end;
```

```
   define header RowsHeader;
      text _ROW_LABEL_ / _ROW_LABEL_ ^= '';
      text _ROW_NAME_;
      space=0;
      style=header;
      cindent=';';
      end;

   define header ColsHeader;
      text _COL_LABEL_ / _COL_LABEL_ ^= '';
      text _COL_NAME_;
      space=1;
      style=header;
      cindent=';';
      end;


   define footer TableOf;
      notes 'NoTitle is 1 if the NOTITLE option was specified.';
      dynamic StratNum NoTitle;
      text "Table " StratNum 3. " of " _ROW_LABEL_ " by "
_COL_LABEL_ / NoTitle= 0
           & StratNum > 0 & _ROW_LABEL_ ^= '' & _COL_LABEL_ ^= '';
      text "Table " StratNum 3. " of " _ROW_LABEL_ " by "
_COL_NAME_  / NoTitle= 0
           & StratNum > 0 & _ROW_LABEL_ ^= ''                    ;
      text "Table " StratNum 3. " of " _ROW_NAME_  " by "
_COL_LABEL_ / NoTitle= 0
           & StratNum > 0 & _COL_LABEL_ ^= '';
      text _ROW_LABEL_ " by " _COL_LABEL_ /  NoTitle = 0 &
_ROW_LABEL_ ^=''
           & _COL_LABEL_ ^= '';
      text _ROW_LABEL_ " by " _COL_NAME_  /  NoTitle = 0 &
_ROW_LABEL_ ^='';
      text _ROW_NAME_  " by " _COL_LABEL_ /  NoTitle = 0 &
_COL_LABEL_ ^='';
      text "Table " StratNum 3. " of " _ROW_NAME_  " by "
_COL_NAME_  / NoTitle= 0
           & StratNum > 0;
      text _ROW_NAME_  " by " _COL_NAME_ / NoTitle = 0;
      style=header;
      end;

   define footer Missing;
      dynamic FMissing;
      text "Frequency Missing = " FMissing -12.99 / FMissing ^= 0;
      space=1;
      style=header;
      end;


   define footer NoObs;
      dynamic SampleSize;
      text "Effective Sample Size = 0" / SampleSize = 0;
      space=1;
      style=header;
      end;

   define cellvalue Frequency;
```

```
   header="Frequency";
   format=BEST7.;
   label="Frequency Count";
   data_format_override=on print=on;
   end;

define cellvalue Expected;
   header="Expected";
   format=BEST6.;
   label="Expected Frequency";
   data_format_override=on print=on;
   end;

define cellvalue Deviation;
   header="Deviation";
   format=BEST6.;
   label="Deviation from Expected Frequency";
   data_format_override=on print=on;
   end;

define cellvalue CellChiSquare;
   header="Cell Chi-Square";
   format=BEST6.;
   label="Cell Chi-Square";
   print=on;
   end;

define cellvalue TotalPercent;
   header="Total Percent";
   format=6.2;
   label="Percent of Total Frequency";
   print=on;
   end;

define cellvalue Percent;
   header="Percent";
   format=6.2;
   label="Percent of Two-Way Table Frequency";
   print=on;
   cellstyle _val_ > 20.0 as {color=#BF6930};
   end;

define cellvalue RowPercent;
   header="Row Percent";
   format=6.2;
   label="Percent of Row Frequency";
   print=on;
   end;

define cellvalue ColPercent;
   header="Column Percent";
   format=6.2;
   label="Percent of Column Frequency";
   print=on;
   end;
```

```
   define cellvalue CumColPercent;
      header="Cumulative Column Percent";
      format=6.2;
      label="Cumulative Percent of Column Frequency";
      print=on;
end;

      header ControllingFor;
      footer TableOf NoObs Missing;

   cellvalue
      Frequency Expected Deviation
      CellChiSquare TotalPercent Percent
      RowPercent ColPercent CumColPercent;
end;
run;

title "City Government Form by Number of Meetings Scheduled";
ods html file='file-path/MyCrosstabsTableLegend.html'  style=daisy;

proc freq;
   tables citygovt*robgrp / missprint;
run;

ods html close;
```

Program Description

Create the user-defined formats and the data set. The FORMAT procedure creates two user-defined formats that can be used in the crosstabulation template. The DATA step creates the Gov data set.

```
Proc Format;
  Value Govtfmt -3='Council Manager'
                 0='Commission'
                 3='Mayor Council'
                .N='Not Applicable'
                 .='   ?';
  Value Robfmt   1='100 or  Less'
                 2='101-200'
                 3='201-300'
                 4='Over 300'
                .N='Not     Known'
                 .='   ?';
  Value colfg   1='yellow'
                 2='red'
                 3='blue'
                 4='purple'
                .N='green'
                 .='black'
                 other='black';
  Value rowfg   -3='red'
                 0='purple'
                 3='blue'
                .N='green'
                 .='black'
                 other='black';
```

```
run;

data gov;
   Label Citygovt='City Government Form'
        Robgrp='Number of Meetings Scheduled';
   Input Citygovt Robgrp Weight;  Missing N;
   Format Citygovt Govtfmt. Robgrp Robfmt.;
   LOOP: OUTPUT; WEIGHT=WEIGHT-1; IF WEIGHT>0 THEN GOTO LOOP;
   DROP WEIGHT;
datalines;
0 1  6
0 3  3
0 2  7
0 4  5
N N 10
-3 1 47
-3 3 49
-3 2 63
-3 4 52
. 2  1
3 1 31
3 2 37
3 3 27
3 4 55
3 .  1
;
```

Establish the ODS path. The ODS PATH statement specifies the locations to write to or read from when you create the PROC TEMPLATE templates.

```
ods path (prepend) work.templat(update);
```

Create the crosstabulation template Base.Freq.CrossTabFreqs. The DEFINE statement creates the crosstabulation template Base.Freq.CrossTabFreqs in the first template store in the path for which you have Write access. The NOTES statement provides information about the crosstabulation table.

```
proc template;
   define crosstabs Base.Freq.CrossTabFreqs;
      notes "Crosstabulation table with legend";
```

Specify a row heading, a column heading, and a label for the table. The ROWS_HEADER= style attribute specifies RowsHeader as the header for rows. The COLS_HEADER= style attribute specifies ColsHeader as the header for columns. The LABEL= attribute specifies a label for the crosstabulation template. The GRAND_TOTAL_STYLE= changes the FontWeight style attribute in the Data style element to bold. This change affects the values in the rightmost column of the last row in the table.

```
      rows_header=RowsHeader cols_header=ColsHeader;
      label = "Frequency Counts and Percentages";
      grand_total_style=data {fontweight=bold};
```

Create the ControllingFor header template. The DEFINE HEADER statement and its attributes create the header template ControllingFor. The DYNAMIC statement declares dynamic variables so that they can be used in expressions. The TEXT statement specifies the text of the header by using dynamic variables that represent label variables and names. In this example, the expression in the TEXT statement resolves to false, so the ControllingFor header does not show up in the output. The STYLE= attribute specifies style information for the headers.

```
define header ControllingFor;
   dynamic StratNum StrataVariableNames StrataVariableLabels;
   text "Controlling for" StrataVariableNames / StratNum > 0;
   style=header;
   end;
```

Create the RowsHeader header template. The DEFINE HEADER statement creates the header RowsHeader, which is specified by the preceding ROWS_HEADER= style attribute. The TEXT statements specify the text of the header by using dynamic variables that represent label variables and names. The first TEXT statement uses an expression to determine whether a label is assigned to the row variable. If there is no label, the next TEXT statement is used, which specifies the row name. In this example, there is a row label for the row variable, so in the output, _ROW_LABEL_ resolves to "City Government Form".The STYLE= attribute specifies style information for the header. The SPACE= attribute specifies that the current header and the previous header should have one blank line between them. The CINDENT= attribute specifies that wrapped lines start at the same column as the open parenthesis.

```
define header RowsHeader;
   text _ROW_LABEL_ / _ROW_LABEL_ ^= '';
   text _ROW_NAME_;
   space=0;
   style=header;
   cindent=';';
   end;
```

Create the ColsHeader header template. The DEFINE HEADER statement creates the header ColsHeader, which is specified by the preceding COLS_HEADER= style attribute. The TEXT statements specify the text of the header by using dynamic variables that represent label variables and names. The first TEXT statement uses an expression to determine whether a label is assigned to the variable. If there is no label, the next TEXT statement is used, which specifies the row name. In this example, there is a column label, so in the output, _COL_LABEL_ resolves to "Number of Meetings Scheduled".The STYLE= attribute specifies style information for the header. The SPACE= attribute specifies that the current header and the previous header should have one blank line between them. The CINDENT= attribute specifies that wrapped lines start at the same column as the open parenthesis.

```
define header ColsHeader;
   text _COL_LABEL_ / _COL_LABEL_ ^= '';
   text _COL_NAME_;
   space=1;
   style=header;
   cindent=';';
   end;
```

Create the TableOf footer template. The DEFINE FOOTER statement and its attributes create the footer template TableOf, which is specified by the FOOTER statement later on in the program. The TEXT statements specify the text of the header by using dynamic variables that represent label variables and names, the NOTITLE option, and the current stratum number. The TEXT statements use expressions with these variables to determine which text is displayed. Only the TEXT statements that have a true expression are displayed in the output. In this example, the only text statement that has a true expression is the fourth TEXT statement, and the text resolves to: " City Government Form by Number of Meetings Scheduled".

```
   define footer TableOf;
      notes 'NoTitle is 1 if the NOTITLE option was specified.';
      dynamic StratNum NoTitle;
      text "Table " StratNum 3. " of " _ROW_LABEL_ " by "
_COL_LABEL_ / NoTitle= 0
           & StratNum > 0 & _ROW_LABEL_ ^= '' & _COL_LABEL_ ^= '';
      text "Table " StratNum 3. " of " _ROW_LABEL_ " by "
_COL_NAME_  / NoTitle= 0
           & StratNum > 0 & _ROW_LABEL_ ^= ''                    ;
      text "Table " StratNum 3. " of " _ROW_NAME_  " by "
_COL_LABEL_ / NoTitle= 0
           & StratNum > 0 & _COL_LABEL_ ^= '';
      text _ROW_LABEL_ " by " _COL_LABEL_ /  NoTitle = 0 &
_ROW_LABEL_ ^=''
           & _COL_LABEL_ ^= '';
      text _ROW_LABEL_ " by " _COL_NAME_  /  NoTitle = 0 &
_ROW_LABEL_ ^='';
      text _ROW_NAME_  " by " _COL_LABEL_ /  NoTitle = 0 &
_COL_LABEL_ ^='';
      text "Table " StratNum 3. " of " _ROW_NAME_  " by "
_COL_NAME_  / NoTitle= 0
           & StratNum > 0;
      text _ROW_NAME_  " by " _COL_NAME_ / NoTitle = 0;
      style=header;
      end;
```

Create additional footer templates. Each of these DEFINE FOOTER statements and each of its attributes creates a footer template. To apply these footers to your output, you must specify them in the FOOTER statement. The DYNAMIC statements declare the dynamic variables FMissing, Stratnum, NoTitle, and SampleSize, so that they can be used in the TEXT statements. The TEXT statements conditionally select text to use as footers. In the first TEXT statement, the expression is true, because FMissing is not 0. Therefore, the first TEXT statement is displayed in the output. In the second TEXT statement, the expression resolves to false, and the NoObs footer does not appear in the output. The STYLE attribute specifies style information for the footers. The SPACE attribute specifies that the current footer and the previous footer should have one blank line between them.

```
   define footer Missing;
      dynamic FMissing;
      text "Frequency Missing = " FMissing -12.99 / FMissing ^= 0;
      space=1;
      style=header;
      end;

   define footer NoObs;
      dynamic SampleSize;
      text "Effective Sample Size = 0" / SampleSize = 0;
      space=1;
      style=header;
      end;
```

Create the cellvalue definitions. The DEFINE CELLVALUE statements define the values that appear in the cells of the crosstabulation table. The HEADER= attribute specifies the text that appears in the legend. The LABEL= attribute specifies the label for the data set column that corresponds to the cellvalue. The LABEL=

attribute affects only the Output destination. The DATA_FORMAT_OVERRIDE=ON attribute specifies to use the format specified in the data component. The PRINT=ON attribute causes the cellvalue to appear in the table. The CELLSTYLE AS statement uses expressions to conditionally set the style element of the cells according to the values of the variables for the Percent cellvalue. The _VAL_ variable represents the value of a cell. Therefore, in this example, if the value in a cell is less than ten, then the font color for the DataStrong style element is green. If the value in the cell is greater than twenty, the font color is #BF6930.

```
define cellvalue Frequency;
   header="Frequency";
   format=BEST7.;
   label="Frequency Count";
   data_format_override=on print=on;
   end;

define cellvalue Expected;
   header="Expected";
   format=BEST6.;
   label="Expected Frequency";
   data_format_override=on print=on;
   end;

define cellvalue Deviation;
   header="Deviation";
   format=BEST6.;
   label="Deviation from Expected Frequency";
   data_format_override=on print=on;
   end;

define cellvalue CellChiSquare;
   header="Cell Chi-Square";
   format=BEST6.;
   label="Cell Chi-Square";
   print=on;
   end;

define cellvalue TotalPercent;
   header="Total Percent";
   format=6.2;
   label="Percent of Total Frequency";
   print=on;
   end;

define cellvalue Percent;
   header="Percent";
   format=6.2;
   label="Percent of Two-Way Table Frequency";
   print=on;
   cellstyle _val_ > 20.0 as {color=#BF6930};
   end;

define cellvalue RowPercent;
   header="Row Percent";
   format=6.2;
   label="Percent of Row Frequency";
   print=on;
```

```
      end;

   define cellvalue ColPercent;
      header="Column Percent";
      format=6.2;
      label="Percent of Column Frequency";
      print=on;
      end;

   define cellvalue CumColPercent;
      header="Cumulative Column Percent";
      format=6.2;
      label="Cumulative Percent of Column Frequency";
      print=on;
end;
```

Specify header and footer templates. The HEADER statement specifies the header templates that are applied to your output. The FOOTER statement specifies the footer templates that are applied to your output. In order for any of the headers and footers that were defined by a DEFINE statement to appear in your output, they must be specified by the FOOTER or HEADER statement.

```
      header ControllingFor;
      footer TableOf NoObs Missing;
```

Specify cellvalues and their order. The CELLVALUE statement specifies which cellvalues appear in the table and the order. In this example, all of the cellvalues that you created appear in the table, in the order specified by the CELLVALUE statement.

```
   cellvalue
      Frequency Expected Deviation
      CellChiSquare TotalPercent Percent
      RowPercent ColPercent CumColPercent;
end;
run;
```

Specify a title, create the HTML output, and specify the name of the HTML file. The TITLE statement provides a title for the output. The ODS HTML statement with the STYLE= option specifies the style template Daisy for the output.

```
title "City Government Form by Number of Meetings Scheduled";
ods html file='file-path/MyCrosstabsTableLegend.html'  style=daisy;
```

Create the crosstabulation table. The FREQ procedure creates a Citygovt by Robgrp crosstabulation table.

```
proc freq;
   tables citygovt*robgrp / missprint;
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination and all the files that are open for that destination.

```
ods html close;
```

Output

Output 12.2 *Output Using Customized Crosstabulation Table Template*

City Government Form by Number of Meetings Scheduled

Frequency Percent Row Percent Column Percent	Number of Meetings Scheduled						
City Government Form	?	Not Known	100 or Less	101-200	201-300	Over 300	Total
?	0 . . .	0 . . .	0 . . .	1 . . .	0 . . .	0 . . .	. .
Not Applicable	0 . . .	10 . . .	0 . . .	0 . . .	0 . . .	0 . . .	. .
Council Manager	0 . . .	0 . . .	47 12.30 22.27 55.95	63 16.49 29.86 58.88	49 12.83 23.22 62.03	52 13.61 24.64 46.43	211 55.24
Commission	0 . . .	0 . . .	6 1.57 28.57 7.14	7 1.83 33.33 6.54	3 0.79 14.29 3.80	5 1.31 23.81 4.46	21 5.50
Mayor Council	1 . . .	0 . . .	31 8.12 20.67 36.90	37 9.69 24.67 34.58	27 7.07 18.00 34.18	55 14.40 36.67 49.11	150 39.27
Total	. .	. .	84 21.99	107 28.01	79 20.68	112 29.32	**382** **100.00**
City Government Form by Number of Meetings Scheduled							
Frequency Missing = 12							

Example 3: Adding Custom Formats to Cellvalues

Features: EDIT statement

Other ODS features
- ODS HTML statement
- ODS PATH statement

Details

This example does not use the DEFINE CROSSTABS statement. Instead, it uses the EDIT statement to edit the crosstabulation table template Base.Freq.CrossTabFreqs that was created in “Example 2: Creating a Crosstabulation Table Template with a Customized Legend” on page 409 by changing the formats of several cellvalues. In “Example 2: Creating a Crosstabulation Table Template with a Customized Legend” on page 409, the following format values were used:

- Frequency: BEST6
- Percent, RowPercent, ColPercent: 6.2

In this example, the Frequency cellvalue is changed to COMMA12; and the Percent, RowPercent, and ColPercent cellvalues are changed to 6.3.

Program

```
 Proc Format;
   Value Govtfmt -3='Council Manager'
                  0='Commission'
                  3='Mayor Council'
                 .N='Not Applicable'
                  .='   ?';
   Value rowfg   -3='red'
                  0='purple'
                  3='blue'
                 .N='green'
                  .='black'
                  other='black';
   Value Robfmt   1='100 or  Less'
                  2='101-200'
                  3='201-300'
                  4='Over 300'
                 .N='Not     Known'
                  .='   ?';
    Value colfg   1='yellow'
                  2='red'
                  3='blue'
                  4='purple'
                 .N='green'
                  .='black'
                  other='black';
run;

data gov;
   Label Citygovt='City Government Form'
        Robgrp='Number of Meetings Scheduled';
   Input Citygovt Robgrp Weight;  Missing N;
   Format Citygovt Govtfmt. Robgrp Robfmt.;
```

```
   LOOP: OUTPUT; WEIGHT=WEIGHT-1; IF WEIGHT>0 THEN GOTO LOOP;
   DROP WEIGHT;
datalines;
0 1  6
0 3  3
0 2  7
0 4  5
N N 10
-3 1 47
-3 3 49
-3 2 63
-3 4 52
. 2  1
3 1 31
3 2 37
3 3 27
3 4 55
3 .  1
;

ods noproctitle;
ods path (prepend) work.templat(update);

proc template;
  edit Base.Freq.CrossTabFreqs;

     edit Frequency;
        format=COMMA12.;

     end;
     edit Percent;
        format=6.3;
     end;
     edit RowPercent;
        format=6.3;
     end;
     edit ColPercent;
        format=6.3;
     end;
  end;
run;

ods html file="file-path/userfmt.html" style=daisy;

title "Applying Custom Formats to Cellvalues";
proc freq;
   tables citygovt*robgrp / missprint;
run;

ods html close;
```

Program Description

Create the user-defined formats and the data set. The FORMAT procedure creates four user-defined formats that can be used in the crosstabulation template. The DATA step creates the Gov data set.

```
 Proc Format;
   Value Govtfmt -3='Council Manager'
                  0='Commission'
                  3='Mayor Council'
                 .N='Not Applicable'
                  .='   ?';
   Value rowfg   -3='red'
                  0='purple'
                  3='blue'
                 .N='green'
                  .='black'
                  other='black';
   Value Robfmt   1='100 or  Less'
                  2='101-200'
                  3='201-300'
                  4='Over 300'
                 .N='Not     Known'
                  .='   ?';
    Value colfg   1='yellow'
                  2='red'
                  3='blue'
                  4='purple'
                 .N='green'
                  .='black'
                  other='black';
run;

data gov;
   Label Citygovt='City Government Form'
        Robgrp='Number of Meetings Scheduled';
   Input Citygovt Robgrp Weight;  Missing N;
   Format Citygovt Govtfmt. Robgrp Robfmt.;
   LOOP: OUTPUT; WEIGHT=WEIGHT-1; IF WEIGHT>0 THEN GOTO LOOP;
   DROP WEIGHT;
datalines;
0 1  6
0 3  3
0 2  7
0 4  5
N N 10
-3 1 47
-3 3 49
-3 2 63
-3 4 52
. 2  1
3 1 31
3 2 37
3 3 27
3 4 55
3 .  1
;
```

Establish the ODS path. The ODS PATH statement specifies the locations to write to or read from when creating the PROC TEMPLATE templates. The ODS NOPROCTITLE statement suppresses the title of the FREQ procedure.

```
ods noproctitle;
```

```
ods path (prepend) work.templat(update);
```

Edit the crosstabulation template Base.Freq.CrossTabFreqs. The EDIT statement changes the crosstabulation table template

Base.Freq.CrossTabFreqs that was created in "Example 2: Creating a Crosstabulation Table Template with a Customized Legend" on page 409.

```
proc template;
  edit Base.Freq.CrossTabFreqs;
```

Apply new formats to the cellvalues Frequency, Percent, RowPercent, and ColPercent. The FORMAT= attribute specifies a format for the cellvalues. The format COMMA12. is applied to Frequency, and the format 6.3 is applied to Percent, RowPercent, and ColPercent.

```
      edit Frequency;
         format=COMMA12.;

      end;
      edit Percent;
         format=6.3;
      end;
      edit RowPercent;
         format=6.3;
      end;
      edit ColPercent;
         format=6.3;
      end;
  end;
run;
```

Create the HTML output and specify the name of the HTML file. The ODS HTML statement with the STYLE= option specifies the style template Daisy for the output.

```
ods html file="file-path/userfmt.html" style=daisy;
```

Create the crosstabulation table and add a title. The FREQ procedure creates a Citygovt by Robgrp crosstabulation table. The TITLE statement specifies a title.

```
title "Applying Custom Formats to Cellvalues";
proc freq;
   tables citygovt*robgrp / missprint;
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination and all the files that are open for that destination.

```
ods html close;
```

Output

Output 12.3 *Crosstabulation Output with Custom Formats Applied to Cellvalues*

Applying Custom Formats to Cellvalues

Frequency Percent Row Percent Column Percent	Number of Meetings Scheduled						
City Government Form	?	Not Known	100 or Less	101-200	201-300	Over 300	Total
?	0 . . .	0 . . .	0 . . .	1 . . .	0 . . .	0 . . .	. .
Not Applicable	0 . . .	10 . . .	0 . . .	0 . . .	0 . . .	0 . . .	. .
Council Manager	0 . . .	0 . . .	47 12.304 22.275 55.952	63 16.492 29.858 58.879	49 12.827 23.223 62.025	52 13.613 24.645 46.429	211 55.236
Commission	0 . . .	0 . . .	6 1.571 28.571 7.143	7 1.832 33.333 6.542	3 0.785 14.286 3.797	5 1.309 23.810 4.464	21 5.497
Mayor Council	1 . . .	0 . . .	31 8.115 20.667 36.905	37 9.686 24.667 34.579	27 7.068 18.000 34.177	55 14.398 36.667 49.107	150 39.267
Total	. .	. .	84 21.990	107 28.010	79 20.681	112 29.319	**382** **100.00**

City Government Form by Number of Meetings Scheduled

Frequency Missing = 12

13

TEMPLATE Procedure: Creating ODS Graphics

Introduction to the Graph Template Language 427
Syntax 430
Where to Go from Here 431

Introduction to the Graph Template Language

Graphics are an indispensable part of statistical analysis. Graphics reveal patterns, identify differences, and provoke meaningful questions about your data. Graphics add clarity to an analytical presentation and stimulate deeper investigation.

The GTL applies accepted principles of graphics design to produce plots that are clean and uncluttered. Colors, fonts, and relative sizes of graph elements are all designed for optimal impact. By default, the GTL produces PNG files, which support true color (the full 24-bit RGB color model) and enable visual effects such as anti-aliasing and transparency, but retain a small file size. GTL statement options enable you to control the content and appearance of the plot down to the smallest detail.

The GTL is designed to produce graphics with minimal syntax. The GTL uses a flexible, building-block approach to create a graph by combining statements in a template called a STATGRAPH template. STATGRAPH templates are defined with the TEMPLATE procedure.

You can create custom graphs by defining your own STATGRAPH templates. To create a custom graph, you must perform the following steps:

1 Define a STATGRAPH template with the TEMPLATE procedure.

2 Use the Graph Template Language to specify the parameters of your graph.

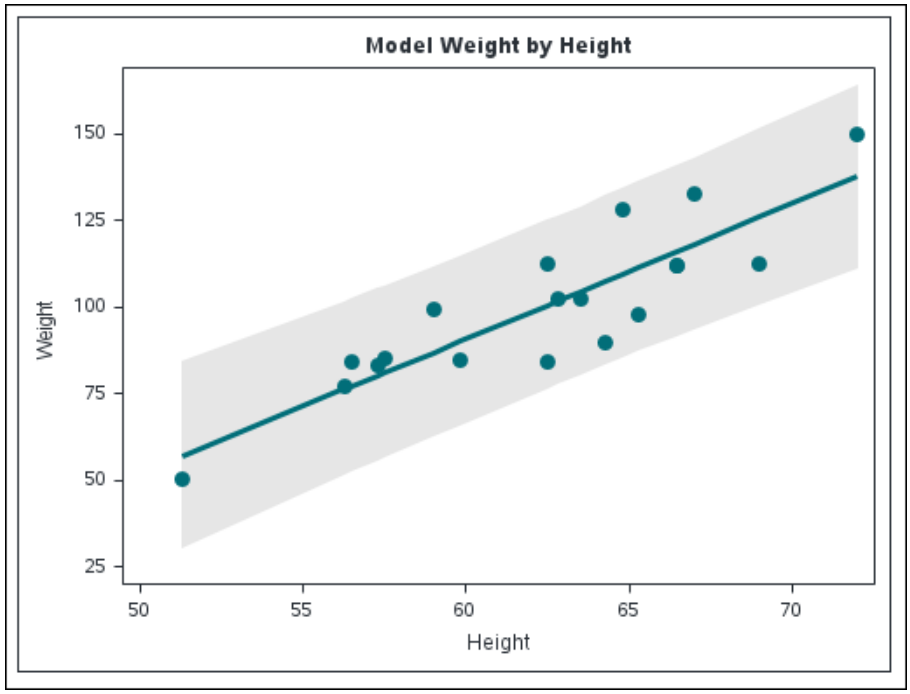
3 Associate your data with the template by using the SGRENDER procedure.

With a few statements, you can create the plots that you need to analyze your data. For example, you can create the following model fit plot with these statements:

```
proc template;
define statgraph mytemplate;
beginGraph;
   entrytitle "Model Weight by Height";
   layout overlay;
   bandplot x=height limitupper=upper limitlower=lower;
   scatterplot y=weight x=height;
   seriesplot y=predict x=height;
endlayout;
endGraph;
end;
run;

proc sgrender data=sashelp.classfit
              template=mytemplate;
run;
```

Figure 13.1 *Model Fit Plot Using Mytemplate and Sashelp.Classfit*



This example defines a STATGRAPH template named `mytemplate`, which uses values from the data set Sashelp.Classfit. This data set contains data variables HEIGHT and WEIGHT and precomputed values for the fitted model (PREDICT) and confidence band (LOWER and UPPER). The SGRENDER procedure uses the data in Sashelp.Classfit and the template `mytemplate` to render the graph. (This example is member GTLMFIT1 in the SAS Sample Library.)

The following two graphics are just examples of what you can do with ODS graphics.

Figure 13.2 *PROC SGSCATTER (SAS) with LISTING Style*

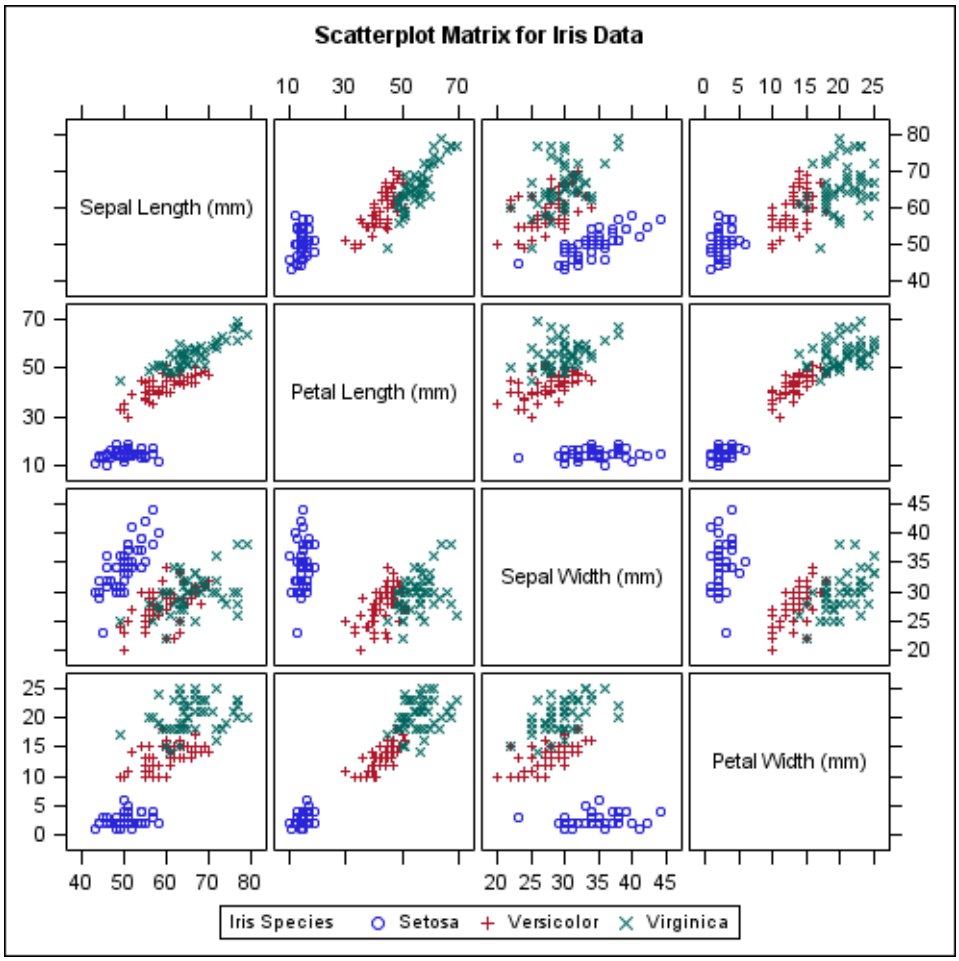
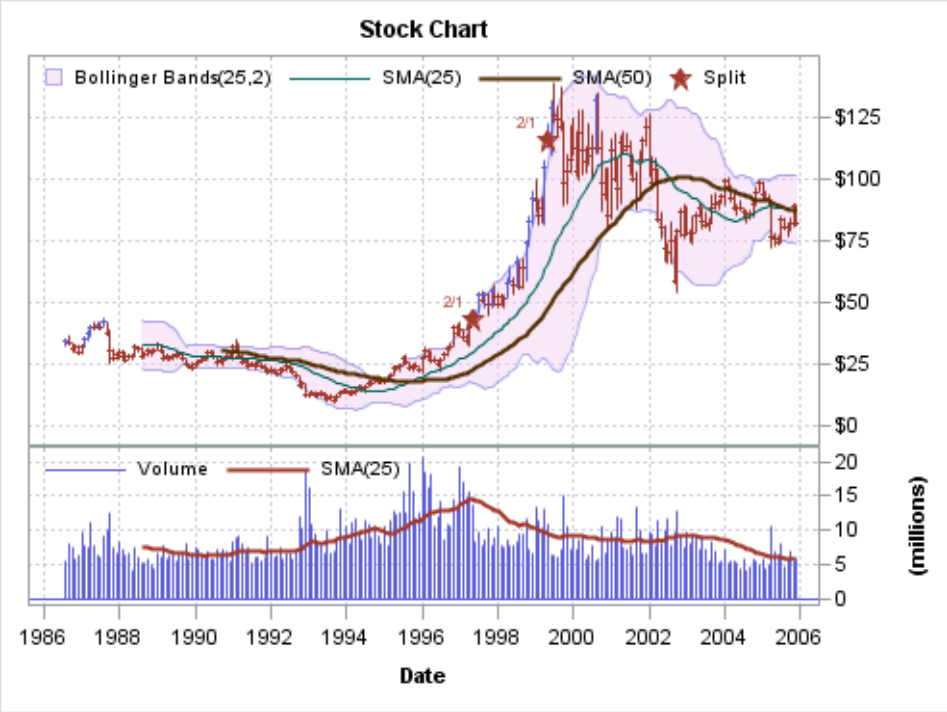


Figure 13.3 Custom Template Rendered with PROC SGRENDER (SAS) and a Custom Style



For complete documentation on the syntax and usage of the Graph Template Language, see the following documentation: *SAS Graph Template Language: Reference* and *SAS Graph Template Language: User's Guide*.

Syntax

```
PROC TEMPLATE;
    DEFINE STATGRAPH graph-path </ STORE=libref.template-store>;
        DYNAMIC variable-1<'text-1'><variable-n<'text-n'>>;
        MVAR variable-1<'text-1'><variable-n<'text-n'>>;
        NMVAR variable-1<'text-1'><variable-n<'text-n'>>;
        NOTES 'text';
        graph-template-language-statements
        END;
RUN;
```

Where to Go from Here

Creating statistical graphics with ODS:
For reference information about the Graph Template Language, see *SAS Graph Template Language: Reference*.

Creating statistical graphics with ODS:
For usage information about PROC TEMPLATE and the Graph Template Language, see *SAS Graph Template Language: User's Guide*.

Managing the various templates stored in template stores:
For reference information about the PROC TEMPLATE statements that help you manage and navigate around the many ODS templates, see Chapter 11, "TEMPLATE Procedure," on page 339.

Modifying an existing style or creating your own style:
For reference information about the style template statements in PROC TEMPLATE, see Chapter 14, "TEMPLATE Procedure," on page 433.

Chapter 14

TEMPLATE Procedure: Creating a Style Template

Overview: TEMPLATE Procedure: Creating a Style Template ***433***
Using the TEMPLATE Procedure to Create a Style 434
Default Style for HTML 434
Customized Version of the HTML Style 436

Concepts: TEMPLATE Procedure: Creating a Style Template ***438***
Terminology 438
Working with Styles 438
Understanding Styles, Style Elements, and Style Attributes 441
Understanding Inheritance 446
Understanding Style References 448
Using the FROM Option 450

Syntax: TEMPLATE Procedure: Creating a Style Template ***452***
PROC TEMPLATE Statement 453
DEFINE STYLE Statement 454
CLASS Statement 456
EDIT Statement 457
END Statement 459
IMPORT Statement 459
NOTES Statement 460
PARENT= Statement 461
REPLACE Statement 462
STYLE Statement 462

Usage: TEMPLATE Procedure: Creating a Style Template ***465***
About Style Attributes 465

Examples: TEMPLATE Procedure: Creating a Style Template ***466***
Example 1: Creating a Stand-Alone Style 466
Example 2: Using User-Defined Attributes 473
Example 3: Modifying the Default Style with the CLASS Statement 482
Example 4: Defining Multiple Style Elements in One STYLE Statement 489
Example 5: Importing a CSS File 495
Example 6: Table Header and Footer Border Formatting 501
Example 7: Enhancing Titles and Footnotes in PDF Output 506
Example 8: Customizing Graphic and Tabular Titles in PDF Output 510

Overview: TEMPLATE Procedure: Creating a Style Template

Using the TEMPLATE Procedure to Create a Style

The TEMPLATE procedure enables you to customize the look of your SAS output. The TEMPLATE procedure creates and modifies styles. The Output Delivery System then uses these styles to produce customized formatted output.

By default, ODS output is formatted according to the various styles that the procedure or DATA step specifies. However, you can also customize the appearance of the output by using the DEFINE STYLE statement in the TEMPLATE procedure.

Default Style for HTML

By default, ODS uses styles to display results. Modify the appearance of the output by customizing these styles. The first output that follows shows the HTML output from PROC PRINT using the default style. The second output that follows shows the same HTML output from PROC PRINT with a customized style. The default style for HTML output in SAS Studio is Styles.HTMLBlue. For information about customizing the output environment in SAS Studio, see “Customized Output Environment” in *SAS Studio: User’s Guide*.

Figure 14.1 *HTML Output from PROC PRINT That Uses the Default Style*

MSRP by Make and Model
(6 Cylinders Only)

Make=Acura

Make	Model	Cylinders	MSRP
Acura	MDX	6	$36,945.00
Acura	TL 4dr	6	$33,195.00
Acura	3.5 RL 4dr	6	$43,755.00
Acura	3.5 RL w/Navigation 4dr	6	$46,100.00
Acura	NSX coupe 2dr manual S	6	$89,765.00

Make=Audi

Make	Model	Cylinders	MSRP
Audi	A4 3.0 4dr	6	$31,840.00
Audi	A4 3.0 Quattro 4dr manual	6	$33,430.00
Audi	A4 3.0 Quattro 4dr auto	6	$34,480.00
Audi	A6 3.0 4dr	6	$36,640.00
Audi	A6 3.0 Quattro 4dr	6	$39,640.00
Audi	A4 3.0 convertible 2dr	6	$42,490.00
Audi	A4 3.0 Quattro convertible 2dr	6	$44,240.00
Audi	A6 2.7 Turbo Quattro 4dr	6	$42,840.00
Audi	TT 3.2 coupe 2dr (convertible)	6	$40,590.00
Audi	A6 3.0 Avant Quattro	6	$40,840.00

Customized Version of the HTML Style

When you are working with styles, you are more likely to modify a SAS style than to write a completely new style. The next display shows the types of changes that you can make to the default style for the HTML output. The new style affects both the contents file and the body file in the HTML output. In particular, in the contents file, the style makes changes to the following attributes:

- the background of the contents file
- the background of the contents title
- the name of the table of contents (Contents instead of Table of Contents)

In the body file, the new style makes changes to the following attributes:

- the text of the header and the text that identifies the procedure that produced the output
- the colors for some parts of the text
- the colors of the borders
- the background colors

Figure 14.2 *HTML Output from PROC PRINT with the Customized Style*

MSRP by Make and Model (6 Cylinders Only)

Make=Acura

Make	Model	Cylinders	MSRP
Acura	MDX	6	$36,945.00
Acura	TL 4dr	6	$33,195.00
Acura	3.5 RL 4dr	6	$43,755.00
Acura	3.5 RL w/Navigation 4dr	6	$46,100.00
Acura	NSX coupe 2dr manual S	6	$89,765.00

Make=Audi

Make	Model	Cylinders	MSRP
Audi	A4 3.0 4dr	6	$31,840.00
Audi	A4 3.0 Quattro 4dr manual	6	$33,430.00
Audi	A4 3.0 Quattro 4dr auto	6	$34,480.00
Audi	A6 3.0 4dr	6	$36,640.00
Audi	A6 3.0 Quattro 4dr	6	$39,640.00
Audi	A4 3.0 convertible 2dr	6	$42,490.00
Audi	A4 3.0 Quattro convertible 2dr	6	$44,240.00
Audi	A6 2.7 Turbo Quattro 4dr	6	$42,840.00
Audi	TT 3.2 coupe 2dr (convertible)	6	$40,590.00
Audi	A6 3.0 Avant Quattro	6	$40,840.00

Concepts: TEMPLATE Procedure: Creating a Style Template

Terminology

For more definitions of terms used in this section, see “Terminology: TEMPLATE Procedure” on page 326.

child
: within a dimension hierarchy, a descendant in level n-1 of a member that is at level n. For example, if a Geography dimension includes the levels Country and City, then Bangkok would be a child of Thailand, and Hamburg would be a child of Germany.

parent
: within a dimension hierarchy, the ancestor in level n of a member in level n-1. For example, if a Geography dimension includes the levels Country and City, then Thailand would be the parent of Bangkok, and Germany would be the parent of Hamburg. The parent value is usually a consolidation of all of its children's values.

Working with Styles

Finding and Viewing the Default Style for ODS Destinations

You can submit the following PROC TEMPLATE step to write the style to the SAS log:

```
proc template;
   source styles.template-name;
run;
```

The following table lists the ODS destinations and their default styles:

Table 14.1 *Recommended Styles for ODS Destinations*

Destination	Default Style	Recommended Styles
EPUB	Daisy	Daisy Moonflower[1]
ODS destination for Excel	Excel	Excel ExcelAqua ExcelHighContrast[4] ExcelIgnite[3] ExcelIlluminate[2] ExcelMarine[6] ExcelMidnight ExcelOpal ExcelUmstead[5]
PDF	Pearl	Pearl PearlJ Sapphire
RTF	RTF	RTF
TAGSETS.RTF	RTF	RTF
ODS destination for PowerPoint	PowerPointLight	PowerPointAqua PowerPointDark PowerPointHighContrast[4] PowerPointIgnite[3] PowerPointIlluminate[2] PowerPointLight ExcelMarine[6] ExcelMidnight PowerPointOpal ExcelUmstead[5]
HTML5	HTMLBlue	Default Dove EGDefault HighContrast HighContrastLarge HTMLBlue HTMLEncore

Destination	Default Style	Recommended Styles
		Ignite
		Illuminate
		Journal
		Journal2
		Journal3
		Plateau
		Raven
		Snow
		Statistical
ODS destination for Word	Word	Word
		WordAqua
		WordHighContrast[4]
		WordIgnite[3]
		WordIlluminate[2]
		WordMarine[6]
		WordMidnight
		WordOpal
		WordUmstead[5]

1 The Moonflower style for ODS EPUB is designed for nighttime or low-light reading.
2 The Illuminate style for Microsoft Office destinations is synchronized to the Opal report theme in SAS Visual Analytics.
3 The Ignite style for Microsoft Office destinations is synchronized to the Midnight2020 report theme in SAS Visual Analytics.
4 The HighContrast style for Microsoft Office destinations is synchronized to the HighContrast2020 report theme in SAS Visual Analytics.
5 The Umstead style for Microsoft Office destinations is synchronized to the Umstead report theme in SAS Visual Analytics.
6 The Marine style for Microsoft Office destinations is synchronized to the Marine report theme in SAS Visual Analytics.

Modifying Style Elements in the Default Style for HTML and Markup Languages

When you work with styles, it is often more efficient to modify a SAS style than to write a completely new style. “Example 3: Modifying the Default Style with the CLASS Statement” on page 482 shows you how to modify the default style.

To customize the style for use at a specific site, it is helpful to know what each style element in the style specifies. For a list of the default HTML and markup languages style elements, see Chapter 20, “Style Elements,” on page 777.

Understanding Styles, Style Elements, and Style Attributes

Understanding Styles, Style Elements, and Style Attributes

The appearance of SAS output is controlled by ODS style templates (ODS styles). ODS styles are produced from compiled STYLE templates written in PROC TEMPLATE style syntax. An ODS style template is a collection of style elements that provides specific visual attributes for your SAS output.

- A style element is a named collection of style attributes that apply to a particular part of the output. Each area of ODS output has a style element name that is associated with it. The style element name specifies where the style attributes are applied. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside the cells. Style elements might also specify default colors and fonts for output that uses the style.
- A style attribute is a visual property, such as color, font properties, and line characteristics, that is defined in ODS with a reserved name and value. Style attributes are collectively referenced by a style element within a style template. Each *style attribute* specifies a value for one aspect of the presentation. For example, the BACKGROUNDCOLOR= attribute specifies the color for the background of an HTML table or for a colored table in printed output. The FONTSTYLE= attribute specifies whether to use a Roman font or an italic font.

Note: Because styles control the presentation of the data, they have no effect on output objects that go to the DOCUMENT or OUTPUT destination.

Available styles are in the SASHELP.TMPLMST item store.In batch mode or SAS Studio, you can display the list of available style templates by using the LIST statement in PROC TEMPLATE:

```
proc template;
   list styles / store=sashelp.tmplmst;
run;
```

For complete information about viewing ODS styles, see “Viewing ODS Styles Supplied by SAS” on page 723.

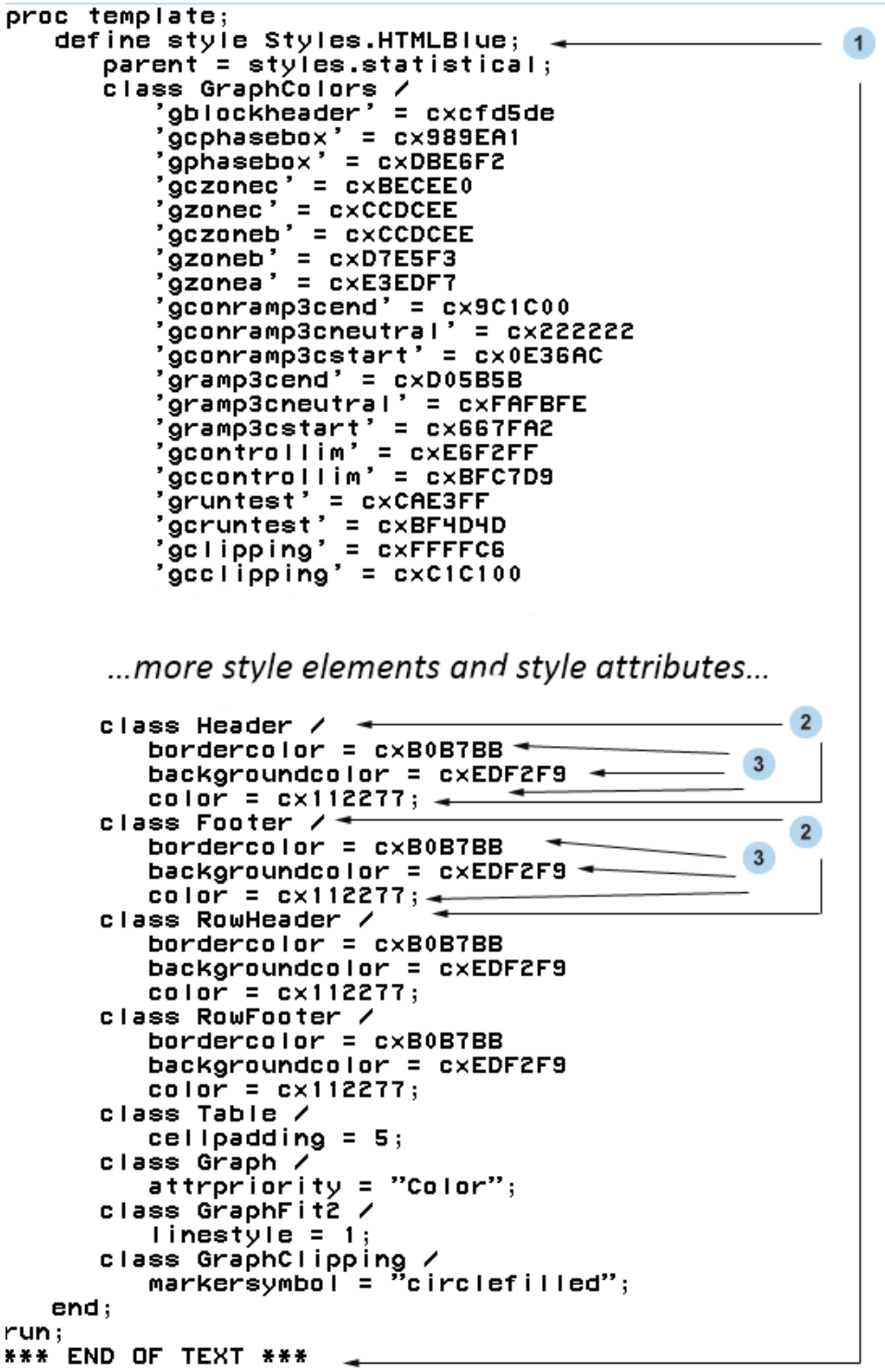
HTML5 output uses the HTMLEncore style template. To help you become familiar with styles, style elements, and style attributes, look at the relationship between them.

You can use the SOURCE statement in PROC TEMPLATE to display the structure of a style template. The following code prints the structure of the HTMLBlue style template to the SAS log:

```
proc template;
   source styles.HTMLBlue;
run;
```

The following figure illustrates the structure of a style. The figure shows the relationship between the style, the style elements, and the style attributes.

Figure 14.3 *Diagram of the HtmlBlue Style*



The following list corresponds to the numbered items in the preceding figure:

1 Styles.HtmlBlue is the *style*. Styles describe how to display presentation aspects (color, font, font size, and so on) of the SAS output. A style determines the overall appearance of the ODS documents that use it. The default style for HTML output is HtmlBlue. Each style consists of style elements.

   You can create new styles with the "DEFINE STYLE Statement" on page 454. New styles can be created independently or from an existing style. You can use "PARENT= Statement" on page 461 to create a new style from an existing style. For complete documentation about ODS styles, see Chapter 19, "Style Templates," on page 723.

2 Header and Footer are examples of *style elements*. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside table cells. Style elements might also specify default colors and fonts for output that uses the style. Style elements exist inside styles and consist of one or more style attributes. Style elements can be user-defined or supplied by SAS. User-defined style elements can be created by the "STYLE Statement" on page 462.

   Note: For a list of the default style elements used for HTML and markup languages and their inheritance, see Chapter 20, "Style Elements," on page 777.

3 BORDERCOLOR=, BACKGROUNDCOLOR=, and COLOR= are examples of *style attributes*. Style attributes specify a value for one aspect of the area of the output that its style element applies to. For example, the COLOR= attribute specifies the value `cx112277` for the font color. For a list of style attributes supplied by SAS, see Chapter 21, "Style Attributes," on page 807.

   Style attributes can be referenced with style references. See "*style-reference*" on page 879 for more information about style references.

The following table shows commonly used style attributes that you can set with the STYLE= option in PROC PRINT, PROC TABULATE, and PROC REPORT. Most of these attributes apply to parts of the table other than cells (for example, table borders and the lines between columns and rows). Note that not all attributes are valid in all destinations. For more information about these style attributes, their valid values, and their applicable destinations, see "Style Attributes Tables" on page 808.

Table 14.2 *Style Attributes for PROC REPORT, PROC TABULATE, and PROC PRINT*

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
ASIS=	X	X		X		X
BACKGROUNDCOLOR=	X	X	X	X	X	X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
BACKGROUNDIMAGE=	X	X	X	X	X	X
BORDERBOTTOMCOLOR=	X	X		X		
BORDERBOTTOMSTYLE=	X	X	X	X		
BORDERBOTTOMWIDTH=	X	X	X	X		
BORDERLEFTCOLOR=	X	X		X		
BORDERLEFTSTYLE=	X	X	X	X		
BORDERLEFTWIDTH=	X	X	X	X		
BORDERCOLOR=	X	X		X	X	X
BORDERCOLORDARK=	X	X	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X	X	X
BORDERRIGHTCOLOR=	X	X		X		
BORDERRIGHTSTYLE=	X	X	X	X		
BORDERRIGHTWIDTH=	X	X	X	X		
BORDERTOPCOLOR=	X	X		X		
BORDERTOPSTYLE=	X	X	X	X		
BORDERTOPWIDTH=	X	X	X	X		

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
BORDERWIDTH=	X	X	X	X	X	X
CELLPADDING=	X		X		X	
CELLSPACING=	X		X		X	
CELLWIDTH=	X	X	X	X		X
CLASS=	X	X	X	X	X	X
COLOR=	X	X	X			
FLYOVER=	X	X		X		X
FONT=	X	X	X	X	X	X
FONTFAMILY=	X	X	X	X	X	X
FONTSIZE=	X	X	X	X	X	X
FONTSTYLE=	X	X	X	X	X	X
FONTWEIGHT=	X	X	X	X	X	X
FONTWIDTH=	X	X	X	X		X
FRAME=	X		X		X	
HEIGHT=	X	X		X	X	X
HREFTARGET=		X		X		X
HTMLSTYLE=	X	X	X	X	X	
NOBREAKSPACE=[2]	X	X		X		X
OUTPUTWIDTH=	X	X	X	X	X	
POSTHTML=[1]	X	X	X	X	X	X
POSTIMAGE=	X	X	X	X	X	X
POSTTEXT=[1]	X	X	X	X	X	X
PREHTML=[1]	X	X	X	X	X	X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
PREIMAGE=	X	X	X	X	X	X
PRETEXT=[1]	X	X	X	X	X	X
PROTECTSPECIALCHARS=		X		X	X	X
RULES=	X		X		X	
TAGATTR=	X	X	X	X	X	X
TEXTALIGN=	X	X	X	X	X	X
URL=		X		X		X
VERTICALALIGN=		X		X		X
WIDTH=	X	X	X	X	X	

1 When you use these attributes in this location, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table. For complete documentation about style attributes and their values, see Chapter 21, “Style Attributes,” on page 807.

2 To help prevent unexpected wrapping of long text strings when using PROC REPORT with the ODS RTF destination, set NOBREAKSPACE=OFF in a location that affects the LINE statement. The NOBREAKSPACE=OFF attribute must be set in the PROC REPORT code either on the LINE statement or on the PROC REPORT statement where style(lines) is specified.

Understanding Inheritance

Overview

Inheritance can be initiated by the PARENT= statement or the FROM= option in the STYLE statement.

The PARENT= statement specifies that PROC TEMPLATE copy all of the style elements from the parent style to the new child style. The style elements are used in the new template unless the new template has style elements that overrides them.

The FROM= option specifies that PROC TEMPLATE copy all of the style attributes from the parent style element to the specified child style element.

Inheritance between Styles

Inheritance between styles is initiated by the PARENT= option, and involves the following process:

1. When the PARENT= statement is specified, style elements in the parent style are copied into the new style. This copying occurs before any inheritance can occur within the new style.
2. If there is a like-named style element within the child style that does not have a FROM option specified, then the style element from the child style overrides the style element from the parent style.
3. If there is a like-named style element within the child style that does have the FROM option specified, then the child style element absorbs the style attributes from the parent style element. If there are like-named style attributes in the two style elements, then the style attributes from the child style element are used.

The following code shows an example of inheritance between two styles:

Example Code 14.1 *Original Code for Creating Style2*

```
define style style1;
    style fonts /
        "docfont" = ("Arial", 3)
        "tablefont" = ("Times", 2);
     style output /
        cellpadding = 5
        borderspacing = 0
        font = fonts("docfont");
    style table from output /
        borderspacing = 2
        font = fonts("tablefont");
     style header /
         backgroundcolor=white
         color=blue
         fontfamily="arial, helvetica"
         fontweight=bold;
 end;

define style style2;
   parent = style1;
     style fonts from fonts /
        "docfont" = ("Helvetica", 3);
     style table from table /
        borderspacing = 4;
     style header /
         fontstyle=roman
         fontsize=5;
end;
```

The Style2 style from the previous code could also be written this way:

Example Code 14.2 *Expanded Version of Style2*

```
define style style2;
   style fonts/
      "docfont" = ("Helvetica", 3)
      "tablefont" = ("Times", 2);
   style output /
      cellpadding = 5
      borderspacing = 0
      font = fonts("docfont");
   style table from output /
      borderspacing=4
      font = fonts("tablefont");
   style header /
         fontstyle=roman
         fontsize=5;
end;
```

Inheritance between Style Elements

The FROM option in a STYLE statement is used to initiate inheritance from another style element. The style element referenced by the FROM option can exist in either the current style or the parent style (if a parent template is specified using the PARENT= statement).

For example, in both the Example Code 14.3 on page 447 and the Example Code 14.4 on page 448 the Table style element, which is created with the `style table from output / ...,` statement, ends up with the following style attributes:

- `cellpadding= 5`
- `borderspacing= 4`
- `font=fonts("tablefont")`

Understanding Style References

A style reference references a style attribute in a style element. The style element can exist either in the current style or in the parent style.

For example, suppose that you create a style element named DataCell that uses the COLOR= and BACKGROUNDCOLOR= style attributes:

```
style datacell / backgroundcolor=blue
                 color=white;
```

To ensure that another style element, NewCell, uses the same background color, use a style reference in the NewCell element, like this:

```
style newcell / backgroundcolor=datacell(backgroundcolor);
```

The style reference `datacell(backgroundcolor)` indicates that the value for the style attribute BACKGROUNDCOLOR= of the style element named DataCell should be used.

Similarly, suppose that you create a style element named HighLighting that defines three style attributes:

```
style highlighting /
   "go"=green
   "caution"=yellow
   "stop"=red;
```

You can then define a style element named Messages that references the colors that are defined in the HighLighting style element:

```
style messages;
   "note"=highlighting("go")
   "warning"=highlighting("caution")
   "error"=highlighting("stop");
```

Because you used style references, multiple style elements can use the colors defined in the HighLighting style element. If you change the value of **go** to **blue** in the HighLighting style element, then every style element that uses the style reference `highlighting("go")` will use blue instead of green.

In the following code, the FONT= style attribute in the Output style element is defined in terms of the Fonts style element. The value **fonts("docfont")** tells PROC TEMPLATE to go to the last instance of the style element named Fonts and use the value for the style attribute DocFont.

The FONT= style attribute in the Table style element is also defined in terms of the Fonts style element. The value **fonts("tablefont")** tells PROC TEMPLATE to go to the last instance of the style element named Fonts and use the value for the style attribute TableFont.

Example Code 14.3 *Program with Unresolved Style References*

```
define style style1;
    style fonts /
        "docfont" = ("Arial", 3)
        "tablefont" = ("Times", 2);
     style output /
        cellpadding = 5
        borderspacing = 0
        font = fonts("docfont");
    style table from output /
        borderspacing = 2
        font = fonts("tablefont");
     style header /
         backgroundcolor=white
         color=blue
         fontfamily="arial, helvetica"
         fontweight=bold;
 end;

define style style2;
   parent = style1;
     style fonts from fonts /
        "docfont" = ("Helvetica", 3);
     style table from table /
        borderspacing = 4;
     style header /
         fontstyle=roman
```

```
            fontsize=5;
   end;
```

When you submit the code in SAS, the output is created as if you submitted the following program. Notice that in the Output style element, the style reference resolves to **("helvetica", 3)**, not **("Arial", 3)**. This is because the "DocFont" user-supplied style attribute in the Style2 style overrides the like-named style attribute in the Style1 style.

Example Code 14.4 *Program with Unresolved Style References*

```
define style style1;
    style fonts /
        "docfont" = ("Arial", 3)
        "tablefont" = ("Times", 2);
     style output /
        cellpadding = 5
        borderspacing = 0
   /*** Resolved from "docfont" in Style2***/
        font = fonts("helvetica", 3);
    style table from output /
        borderspacing = 2
   /*** Resolved from "tablefont" in Style1***/
        font = fonts("Times", 2);
     style header /
         backgroundcolor=white
         color=blue
         fontfamily="arial, helvetica"
         fontweight=bold;
 end;

define style style2;
   parent = style1;
     style fonts from fonts /
        "docfont" = ("Helvetica", 3);
     style table from table /
        borderspacing = 4;
     style header /
         fontstyle=roman
         fontsize=5;
end;
```

Using the FROM Option

The FROM option is used with a style element in order to inherit from another style element. If you omit the FROM option, then you can have an incomplete style element in the child style.

For example, in the following SAS program, the style Concepts.Style2 inherits all of its style elements and style attributes from the style Concepts.Style1. However, the instance of the style element Colors in Concepts.Style2 overrides the instance of Colors in Concepts.Style1. This is because there is no FROM option in the STYLE statement that creates Colors in Concepts.Style2. Therefore, Colors has one style attribute: `"dark"=dark blue`.

When you run the program, the only style references to Color that resolve are references that refer to the **"dark"** style attribute. Style references in Concepts.Style1 and Concepts.Style2 such as **colors("fancy")** and **colors("medium")** do not resolve because they refer to attributes that were not copied into the current instance of the Colors style element. The resulting output is Figure 14.16 on page 452.

To correct this, in the following program, you can add the FROM option to the STYLE statement that creates the Colors style element in Concepts.Style2:

```
style colors from colors /
   "dark"=dark blue;
```

Note: In the following code, Concepts is a folder that is created in **Templates** ⇨ **Sasuser.Templat**. Style1 is a template that is created in the Concepts folder.

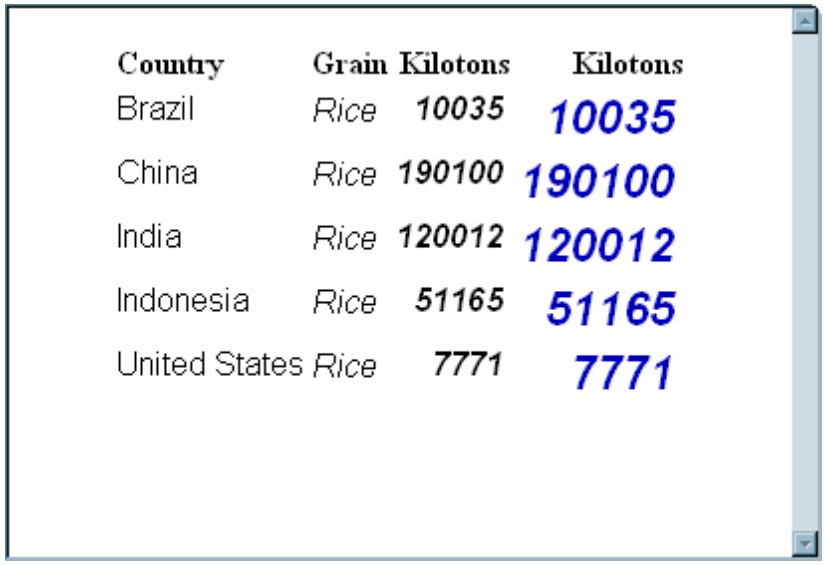
Example Code 14.5 *Creating the Colors Style Element without the FROM Option*

```
proc template;
   define style concepts.style1;
      style colors /
         "default"=white
         "fancy"=very light vivid blue
         "medium"=red ;
      style celldatasimple /
         fontfamily=arial
         backgroundcolor=colors("fancy")
         color=colors("default");
      style celldataemphasis from celldatasimple /
         color=colors("medium")
         fontstyle=italic;
      style celldatalarge from celldataemphasis /
         fontweight=bold
         fontsize=3;
   end;
run;

proc template;
   define style concepts.style2;
      parent=concepts.style1;
      style colors /
         "dark"=dark blue;
      style celldataemphasis from celldataemphasis /
         backgroundcolor=white;
      style celldatasmall from celldatalarge /
         fontsize=5
         color=colors("dark")
         backgroundcolor=colors("medium");
   end;
run;
```

For the complete SAS code that created the following output, see the version of the code without the FROM option in "Using the FROM option" in *SAS Output Delivery System: User's Guide*.

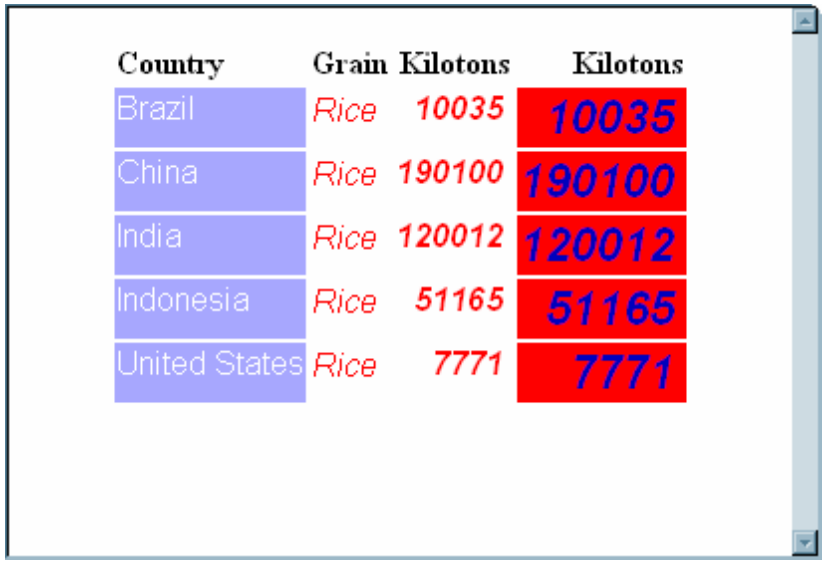
Figure 14.4 Output Created without the FROM Option



Country	Grain	Kilotons	Kilotons
Brazil	Rice	10035	10035
China	Rice	190100	190100
India	Rice	120012	120012
Indonesia	Rice	51165	51165
United States	Rice	7771	7771

For the complete SAS code that created the following output, see the version of the code with the FROM option in "Using the FROM option" in *SAS Output Delivery System: User's Guide*.

Figure 14.5 Output Created with the FROM Option



Country	Grain	Kilotons	Kilotons
Brazil	Rice	10035	10035
China	Rice	190100	190100
India	Rice	120012	120012
Indonesia	Rice	51165	51165
United States	Rice	7771	7771

Syntax: TEMPLATE Procedure: Creating a Style Template

PROC TEMPLATE;

DEFINE STYLE *style-path* | **Base.Template.Style** </ STORE=*libref.template-store*>;
PARENT=*style-path*;
NOTES "*text*";
CLASS *style-element-name(s)* <"*text*">
</ *style-attribute-specification(s)*>;
STYLE *style-element-name(s)* <FROM *style-element-name* | _SELF_ > <"*text*">
</ *style-attribute-specification(s)*>;
END;
END;

Statement	Task	Example
PROC TEMPLATE	Begin a PROC TEMPLATE template	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6
DEFINE STYLE	Create a style for any destination that supports the STYLE= option	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6
CLASS	Create a style element from a like-named style element	Ex. 3, Ex. 5
END	End the style	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6
EDIT	Edit a style	
IMPORT	Import Cascading Style Sheet (CSS) information from a file into the style	Ex. 5
NOTES	Provide information about the style	
PARENT=	Specify the style from which the current style inherits	Ex. 3, Ex. 5, Ex. 6
STYLE	Create or modify one or more style elements	Ex. 1, Ex. 2, Ex. 4, Ex. 6

PROC TEMPLATE Statement

Begins a PROC TEMPLATE template.

Syntax

PROC TEMPLATE;

DEFINE STYLE *style-path* | **Base.Template.Style** </ STORE=*libref.template-store*>;
PARENT=*style-path*;
NOTES "*text*";
CLASS *style-element-name(s)* <"*text*">
</ *style-attribute-specification(s)*>;
STYLE *style-element-name(s)* <FROM *style-element-name* | _SELF_ > <"*text*">
</ *style-attribute-specification(s)*>;
END;
END;

DEFINE STYLE Statement

Creates a style for any destination that supports the STYLE= option. The DEFINE STYLE statement begins a DEFINE STYLE statement block.

Requirement: An END statement must be the last statement in the template.

Supports: The DEFINE STYLE statement supports the following statements: "CLASS Statement" on page 456, "IMPORT Statement" on page 459, "NOTES Statement" on page 391, "PARENT= Statement" on page 461, and "STYLE Statement" on page 462.

Examples:
"Example 1: Creating a Stand-Alone Style" on page 466
"Creating a Cover Page with the LAYOUT_ABSOLUTE Method" in *SAS Output Delivery System: Advanced Topics*
"Example 1: Adding Text to Multiple Destinations " on page 62
"Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570
"Example 7: Table Header and Footer Border Formatting" on page 580
"Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

DEFINE STYLE *style-path* | Base.Template.Style </ STORE=*libref.template-store*>;
PARENT=*style-path*;
NOTES "*text*";
CLASS *style-element-name(s)*<"*text*">
</ *style-attribute-specification(s)*>;
IMPORT *style-specification* <*media-type-1* <, *media-type-2*> …>;
STYLE *style-element-name(s)* <FROM *style-element-name* | _SELF_ > <"*text*">
</ *style-attribute-specification(s)*>;
END;

Required Arguments

style-path
specifies where to store the style. A *style-path* consists of one or more names, separated by periods. Each name represents a directory in a *template store*. PROC TEMPLATE writes the style to the first writable template store in the current path.

Base.Template.Style
creates a style that is the parent of all styles that do not explicitly specify a parent. After this template is created, you do not need to explicitly specify it in your SAS programs. It is automatically applied to all output until you specifically remove it from the item store.

To view the Base.Template.Style using the SAS Windowing Environment:

1. In the Results window, select the **Results** folder. Right-click and select **Templates** to open the Templates window.
2. Double-click **Sashelp.Tmplmst** to view the contents of that directory.
3. Double-click **Base** to view the contents of that directory.
4. Double-click **Template** to view the contents of that directory.

CAUTION
The Base.Template.Style supplied by SAS contains inheritance information used by many styles. If this inheritance information is not retained, some style elements might not appear in the output. To safely create your own Base.Template.Style, you can start with the existing Base.Template.Style template by writing it to an external file and editing the existing template contents.

Restriction If the PARENT= statement is specified, then PARENT= must refer to a style other than Base.Template.Style.

Interaction The Base.Template.Style top-level template attributes are overridden by other style templates.

Tip To view an existing style to base your own Base.Template.Style on, see “Working with Styles ” on page 438.

Optional Argument

STORE=*libref.template-store*
specifies the template store in which to store the style. If the template store does not exist, then it is created.

Restriction The syntax of the STORE= option does not become part of the compiled template.

CLASS Statement

Creates a style element from a like-named style element.

Restriction: The CLASS statement must be used within a DEFINE STYLE template block.

Example: The following statements are equivalent:

```
class fonts;

style fonts from fonts;

style fonts from _self_;
```

Examples: “Example 3: Modifying the Default Style with the CLASS Statement” on page 482
“Example 5: Importing a CSS File” on page 495

Syntax

CLASS *style-element-name(s)* <"*text*"> </ *style-attribute-specification(s)*>;

Required Argument

style–element-name
specifies one or more style elements to be duplicated and modified.

Tip If there are multiple style element names specified within a style and an attribute is specified more than once, then the value of the last attribute specified is used.

See For a complete description of *style–element-name*, see “*style-element-name*” on page 462in the STYLE statement.

For a list of style elements, see Chapter 20, “Style Elements,” on page 777.

Optional Arguments

style-attribute-specification(s)
specifies new style attributes or modifications to existing style attributes for the new style element.

Each *style-attribute-specification* has this general form:

style-attribute-name=< | >*style-attribute-value*

style-attribute-name
is the name of an attribute that is listed in “Style Attributes Tables” on page 808, or it is the name of a user-defined style attribute.

Tip If *style-attribute-name* refers to a user-defined attribute, then enclose the name in quotation marks. If *style-attribute-name* refers to an

attribute that is listed in “Style Attributes Tables” on page 808, then do not enclose the name in quotation marks.

style-attribute-value
assigns the value to the attribute. If an attribute from the table in “Style Attributes Tables” on page 808 is specified, then specify the type of value that the attribute expects.

For more information about style-attribute values, see Chapter 21, “Style Attributes,” on page 807.

|
prevents the style attribute from being inherited by any child style elements.

Restriction If there are multiple style element names specified within a style and an attribute is specified more than once, then the value of the last attribute specified is used.

Tips Override any attribute of the parent style element, whether it is inherited or explicitly defined, by specifying it in the STYLE statement without the FROM option.

If an attribute is defined in a like-named style element in the parent style and it is not explicitly specified in the STYLE statement of the new like-named style element, then the attribute is not inherited, unless you specify the FROM option.

"*text*"
provides information about the STYLE statement. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not become part of the compiled style.

EDIT Statement

Edits an existing template. The EDIT statement replaces the DEFINE statement in a template block when editing. You can use the EDIT statement in place of any DEFINE statement.

Restriction: If you edit a template that is a link, the link is broken and a separate template is created.

Requirement: An END statement must follow the EDIT statement and all of the editing instructions.

Interaction: In some cases, you can use an EDIT statement inside a set of editing instructions. When you edit a table template, you can also edit one or more column, header, or footer templates that are defined in the table. When you edit a column template, you can also edit one or more header templates that are defined for that column.

Example: “Example 1: Editing a Table Template That a SAS Procedure Uses” on page 550

Syntax

EDIT *template-path-1* <AS *template-path-2* > </ STORE=*libref.template-store*>;
template-statements;
END;

Required Argument

template-path-1
specifies a template to edit. *template-path-1* consists of one or more names that are separated by periods. Each name represents a directory in a template store, which is a type of SAS file.

Interaction The STORE= option specifies a particular template store to read from and write to.

Tip To determine the templates that a procedure or DATA step uses, submit the ODS TRACE ON statement before you run the SAS program. (See "ODS TRACE Statement" on page 70.)

Optional Arguments

AS *template-path-2*
specifies the location in which to store the edited template, where *template-path-2* consists of one or more names that are separated by periods. Each name represents a directory in a template store, which is a type of SAS file. By default, PROC TEMPLATE writes the edited template to the first writable template store in the current path.

Default If you omit AS *template-path-2*, PROC TEMPLATE writes the edited template to *template-path-1* in the first writable template store.

Restriction If the current EDIT statement is inside a set of editing instructions, do not use the AS *template-path-2* option.

STORE=*libref.template-store*
specifies the template store from which to read *template-path-1* and in which to store *template-path-2*.

template-statements
template-statements are any statements or attributes that are valid between the DEFINE statement and the END statement.

Editing an Existing Template

When you use the EDIT statement, the following occurs:

- By default, PROC TEMPLATE looks for *template-path-1* in the list of template stores that is defined by the PATH statement. (See "PATH Statement" on page 351.) It opens a copy of the first template path that it finds in a template store that has Read access.
- PROC TEMPLATE writes the modified template to the first template store in the current path with Update access. If you omit a second template path to write to, then PROC TEMPLATE uses *template-path-1*. Therefore, if the template store from which *template-path-1* is read has Update access, you are actually modifying the original template. Otherwise, the modified file is written to a template store to which you do have Update access.

  If you do specify a second template path, then PROC TEMPLATE writes the edited template to the specified path in the first template store to which you have Write access.

END Statement

Ends the style.

Requirement: An END statement must be the last statement in the template.

Syntax

END;

IMPORT Statement

Imports Cascading Style Sheet (CSS) information from a file into the style.

Restriction: The IMPORT statement must be used within a DEFINE STYLE template block.

Requirement: CSS files must be written in the same type of CSS that the ODS HTML statement produces. Only class names that match ODS style element names are supported, with no IDs and no context-based selectors. To view the CSS code that ODS creates, you can specify the STYLESHEET= option, or you can view the source of an HTML file and look at the code between the tags at the top of the file.

Example: “Example 5: Importing a CSS File” on page 495

Syntax

IMPORT *file-specification* <*media-type-1* <, *media-type-2* …>>

Required Argument

file-specification
specifies a file, fileref, or URL that contains CSS code. After you import the CSS code, it is converted to style attributes and style elements that can be used with PROC TEMPLATE.

file-specification is one of the following:

"*external-file*"
is the name of the external CSS file.

Requirement You must enclose *external-file* in quotation marks.

fileref
is a file reference that has been assigned to an external CSS file. Use the FILENAME statement to assign a fileref.

See For information about the FILENAME statement, see *SAS DATA Step Statements: Reference*.

"*URL*"
is a URL to an external CSS file.

Requirement You must enclose *external-file* in quotation marks.

Optional Argument

media-type
specifies one or more media blocks that correspond to the type of media on which your output will be rendered. CSS uses media type blocks to specify how a document is to be presented on different media (for example, on the screen, on paper, with a speech synthesizer, or with a braille device).

The media block is added to your output in addition to the CSS code that is not contained in any media blocks. By using the *media-type* option, in addition to the general CSS code, you can import the section of a CSS file intended only for a specific media type.

Default If no *media-type* is specified in your ODS statement, but you have specified media types in your CSS file, then ODS uses the Screen media type.

Range You can specify up to ten different media types.

Requirement You must separate multiple *media-types* with commas.

Tip If you specify multiple media types, all of the style information in all of the media types is applied to your output. However, if there is duplicate style information in different media blocks, then the styles from the last media block are used.

NOTES Statement

Provides information about the style.

Restriction: The NOTES statement must be used within a DEFINE STYLE template block.

Tip: The NOTES statement becomes part of the compiled style template, which you can view with the SOURCE statement, whereas SAS comments do not.

Syntax

NOTES "*text*";

Required Argument

"*text*"
provides information about the style.

PARENT= Statement

Specifies the style from which the current style inherits.

Restriction: The PARENT= statement must be used within a DEFINE STYLE template block.

Examples:
"Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397
"Creating a Cover Page with the LAYOUT_ABSOLUTE Method" in *SAS Output Delivery System: Advanced Topics*
"Example 1: Adding Text to Multiple Destinations " on page 62
"Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570
"Example 7: Table Header and Footer Border Formatting" on page 580

Syntax

PARENT=*style-path*;

Required Argument

style-path
specifies the style to inherit from.

style-path consists of one or more names, separated by periods. Each name represents a directory in a template store. The current style inherits from the specified style in the first readable template store in the current path.

When you specify a parent, all of the style elements, style attributes, and statements that are specified in the parent's style template are used in the current style template unless the current style template overrides them.

SAS provides some styles. You can specify one of these styles for *style-path*, or you can specify a user-defined style. These are some of the styles that are currently shipped with SAS:

- Styles.Default
- Styles.Journal
- Styles.Grayscaleprinter
- Styles.Statistical
- Styles.Minimal
- Styles.Pearl

- Styles.Statdoc

For information about finding an up-to-date list of the styles and for viewing a style, see “Working with Styles ” on page 438.

Restriction If the PARENT= statement is specified, then PARENT= must refer to a style other than Base.Template.Style.

REPLACE Statement

The REPLACE statement is no longer supported. Use the STYLE statement or the CLASS statement to create and modify style elements.

STYLE Statement

Creates or modifies one or more style elements.

Restriction: The STYLE statement must be used within a DEFINE STYLE template block.

Example: “Example 1: Creating a Stand-Alone Style” on page 466

Syntax

STYLE *style-element-name(s)*
<FROM *existing-style-element-name* | _SELF_ ><"*text*">
</ *style-attribute-specification(s)*>;

Required Argument

style-element-name
specifies one or more style elements to be created or modified. If *style-element-name* is a new style element, then PROC TEMPLATE stores the style element in the current style. If *style-element-name* overrides a style element that is a parent of another element, then all of the descendents of *style-element-name*, including those inherited from parent styles, also inherit the new attributes.

Style element inheritance follows these general guidelines:

- If a like-named style element already exists in the child style and it *is not* created by using the FROM option, then the style element in the child style overrides the style element of the same name in the parent style.
- If a like-named style element already exists in the child style and it *is* created by using the FROM option, then the style attributes from the parent style element are absorbed into the style element in the child style.
- If an attribute is defined in a like-named style element in the parent style and it is not explicitly specified in the STYLE statement of the new like-named

style element, then the attribute is not inherited, unless you specify the FROM option.

- If there are multiple identical style element names specified within a style and an attribute is specified more than once, then the value of the last attribute specified is used.

Requirement Style elements must be separated by commas.

See Chapter 20, "Style Elements," on page 777 for a list of style elements

Example The following STYLE statement uses a style element list:

```
style data, data1, dataempty from _self_ /
      color = red
      backgroundcolor = black;
```

That statement is equivalent to specifying the following STYLE statements together:

```
style data  from data /
      color = red
      backgroundcolor = black;
style data1 from data1/
      color = red
      backgroundcolor = black;
style dataempty from dataempty /
      color = red
      backgroundcolor = black
```

Example "Example 4: Defining Multiple Style Elements in One STYLE Statement" on page 489

Optional Arguments

FROM *existing-style-element-name* | _SELF_
specifies that the preceding *style-element-name* inherit the style attributes from the *existing-style-element-name*.

existing-style-element-name
specifies the existing style element that another style element inherits from. *existing-style-element-name* can have the same name as the preceding *style-element-name*, or it can be the name of another style element. The style element must exist in the current style or in the parent of the current style. Style inheritance using the FROM option follows these general guidelines:

- If a like-named style element already exists in the child style and it *is not* created by using the FROM option, then the style element in the child style overrides the style element of the same name in the parent style.
- If a like-named style element already exists in the child style and it *is* created by using the FROM option, then the style attributes from the parent style element are absorbed into the style element in the child style.
- If an attribute is defined in a like-named style element in the parent style and it is not explicitly specified in the STYLE statement of the new like-named style element, then the attribute is not inherited, unless you specify the FROM option.

- PROC TEMPLATE looks first in the current style for the style element. If PROC TEMPLATE does not find the style element, then it looks in the parent style.

Example The following statement specifies that the style element Data1 be created from the style element Data2, and that the COLOR=BLACK style attribute be added.

```
style data1 from data2 / color=black;
```

SELF
specifies that the parent of the style element should have the same name as the new style element.

Tip The _SELF_ option is most useful when specifying multiple style elements.

See Chapter 20, "Style Elements," on page 777 for a list of style elements

Example The following STYLE statement uses the FROM _SELF_ option:

```
style data, data1, dataempty from _self_ /
      color = red backgroundcolor = black;
```

That statement is equivalent to specifying the following STYLE statements together:

```
style data from  data /
      color = red
      backgroundcolor = black;

style data1 from  data1 /
      color = red
      backgroundcolor = black;

style dataempty from dataempty /
      color = red
      backgroundcolor = black
```

style-attribute-specification(s)
specifies new style attributes or modifications to existing style attributes for the new style element.

Each *style-attribute-specification* has this general form:

style-attribute-name=< | >*style-attribute-value*

style-attribute-name
is the name of an attribute that is listed in "Style Attributes Tables" on page 808, or it is the name of a user-defined style attribute.

Tip If *style-attribute-name* refers to a user-defined attribute, then enclose the name in quotation marks. If *style-attribute-name* refers to an attribute that is listed in "Style Attributes Tables" on page 808, then do not enclose the name in quotation marks.

style-attribute-value
assigns the value to the attribute. If an attribute from the table in "Style Attributes Tables" on page 808 is specified, then specify the type of value that the attribute expects.

For more information about style-attribute values, see Chapter 21, “Style Attributes,” on page 807.

|

prevents the style attribute from being inherited by any child style elements.

Restriction If there are multiple style element names specified within a style and an attribute is specified more than once, then the value of the last attribute specified is used.

Tips Override any attribute of the parent style element, whether it is inherited or explicitly defined, by specifying it in the STYLE statement without the FROM option.

If an attribute is defined in a like-named style element in the parent style and it is not explicitly specified in the STYLE statement of the new like-named style element, then the attribute is not inherited, unless you specify the FROM option.

"*text*"
provides information about the STYLE statement. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not become part of the compiled style.

Usage: TEMPLATE Procedure: Creating a Style Template

About Style Attributes

Overview

Style attributes influence the characteristics of individual cells, tables, documents, graphs, and HTML frames. Style attributes exist within style elements and are specified by the STYLE statement on page 462 or the CLASS statement on page 456. The default value for an attribute depends on the style that is in use. For information about styles, style elements, and style attributes, see “Understanding Styles, Style Elements, and Style Attributes” on page 441. For information about using style attributes with ODS Statistical Graphics, see the chapter on controlling the appearance of your graphics in *SAS Graph Template Language: User’s Guide*.

Style attributes can be supplied by SAS or user-defined. Style attributes can be referenced with a style reference. See “Understanding Style References ” on page 448 and “*style-reference*” on page 879 for more information.

The implementation of an attribute depends on the ODS destination that formats the output. When creating HTML output, the implementation of an attribute depends on the browser that is used.

For a list of the values that style attributes can specify, see Chapter 21, “Style Attributes,” on page 807. For a list of style elements that you can specify style attributes in, see Chapter 20, “Style Elements,” on page 777.

Examples: TEMPLATE Procedure: Creating a Style Template

Example 1: Creating a Stand-Alone Style

Features:
BACKGROUNDCOLOR= in the STYLE statement
BORDERWIDTH= in the STYLE statement
BORDERSPACING= in the STYLE statement
FONTFAMILY= in the STYLE statement
FONTSIZE= in the STYLE statement
FONTSTYLE= in the STYLE statement
FONTWEIGHT= in the STYLE statement
COLOR= in the STYLE statement
CLASSLEVELS= table attribute in the DEFINE TABLE statement
DYNAMIC statement in the DEFINE TABLE statement
MVAR statement in the DEFINE TABLE statement
BLANK_DUPS= in the DEFINE COLUMN statement
GENERIC= in the DEFINE COLUMN statement
HEADER= in the DEFINE COLUMN statement
STYLE= in the DEFINE COLUMN statement
TEXT statement in the DEFINE FOOTER statement
Other ODS features
- ODS HTML statement
- FILE statement with ODS= option
- PUT statement with _ODS_ argument

Data set: Grain_Production

Format: $CNTRY.

Details

This example creates a style that is not based on any other style. When you create a style, you will usually base it on one of the styles that SAS provides (see “Example 3: Modifying the Default Style with the CLASS Statement” on page 482). However, this example is provided to show you some of the basic ways to create a style.

It is important to understand that by default, certain table elements are created with certain style elements. For example, unless you specify a different style element with the STYLE= attribute, ODS produces SAS titles with the SystemTitle style element. Similarly, unless you specify otherwise, ODS produces headers with the Header style element. (For information about each style element, see Chapter 20, “Style Elements,” on page 777.

Program

```
proc template;
   define style newstyle;
      style cellcontents /
         fontfamily="arial, helvetica"
         fontweight=medium
         backgroundcolor=blue
         fontstyle=roman
         fontsize=5
         color=white;

      style header /
         backgroundcolor=very light blue
         fontfamily="arial, helvetica"
         fontweight=medium
         fontstyle=roman
         fontsize=5
         color=white;

     style systemtitle /
         fontfamily="arial, helvetica"
         fontweight=medium
         backgroundcolor=white
         fontstyle=italic
         fontsize=6
         color=red;

     style footer from systemtitle /
         fontsize=3;

     style table /
         borderspacing=5
         borderwidth=10;

   end;
run;

proc template;
```

```
   define table table1;

   mvar sysdate9;

   dynamic colhd;

   classlevels=on;

   define column char_var;
      generic=on;
      blank_dups=on;
      header=colhd;
      style=cellcontents;
   end;

   define column num_var;
      generic=on;
      header=colhd;
      style=cellcontents;
   end;

   define footer table_footer;
      text "Prepared on " sysdate9;
   end;

   end;
run;
ods html close;

ods html file="file-path/newstyle-body.htm" style=newstyle;

   title "Leading Grain Producers";
   title2 "in 1996";

data _null_;
   set grain_production;
   where type  in ("Rice", "Corn") and year=1996;

   file print ods=(
        template="table1"

        columns=(
           char_var=country(generic=on format=$cntry.
                    dynamic=(colhd="Country"))
           char_var=type(generic dynamic=(colhd="Year"))
           num_var=kilotons(generic=on format=comma12.
                   dynamic=(colhd="Kilotons"))
           )
        );

   put _ods_;
run;

ods html close;
ods html;
```

Program Description

Create a new style named NewStyle with the style element CellContents. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style called NewStyle. This STYLE statement

defines the style element CellContents. This style element consists of the style attributes that appear in the STYLE statement. The FONTFAMILY= attribute tells the browser to use the Arial font if it is available, and to look for the Helvetica font if Arial is not available.

```
proc template;
   define style newstyle;
      style cellcontents /
         fontfamily="arial, helvetica"
         fontweight=medium
         backgroundcolor=blue
         fontstyle=roman
         fontsize=5
         color=white;
```

Create the style element Header. This STYLE statement creates the style element Header. By default, ODS uses Header to produce both spanning headers and column headings. This style element uses a different background color from CellContents. It uses the same font (Arial or Helvetica), the same font style (roman), the same font color (white), and the same font size (5) as CellContents.

```
      style header /
         backgroundcolor=very light blue
         fontfamily="arial, helvetica"
         fontweight=medium
         fontstyle=roman
         fontsize=5
         color=white;
```

Create the style element SystemTitle. This STYLE statement creates the style element SystemTitle. By default, ODS uses SystemTitle to produce SAS titles. This style element uses a color scheme of a red foreground on a white background. It uses the same font and font weight as Header and CellContents, but it adds an italic font style and uses a larger font size.

```
      style systemtitle /
         fontfamily="arial, helvetica"
         fontweight=medium
         backgroundcolor=white
         fontstyle=italic
         fontsize=6
         color=red;
```

Create the style element Footer. This STYLE statement creates the style element Footer. This style element inherits all the attributes of SystemTitle. However, the font size that it inherits is overwritten by the FONTSIZE= attribute in its template.

```
      style footer from systemtitle /
         fontsize=3;
```

Create the style element Table. This STYLE statement creates the style element Table. By default, ODS uses this style element to display tables.

```
      style table /
         borderspacing=5
         borderwidth=10;
```

End the style. The END statement ends the style template. The RUN statement executes the TEMPLATE procedure.

```
   end;
run;
```

Create the table template Table1. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE TABLE statement creates a new table template called Table1.

```
proc template;
   define table table1;
```

Specify the symbol that references one macro variable. The MVAR statement defines a symbol, SysDate9, that references a macro variable. ODS will use the value of this macro variable as a string. References to the macro variable are resolved when ODS binds the table template to the data component to produce an output object. SYSDATE9 is an automatic macro variable whose value is always available.

```
   mvar sysdate9;
```

Specify the symbol that references a value to be supplied by the data component. The DYNAMIC statement defines a symbol, Colhd, that references a value that the data component supplies when ODS binds the template and the data component to produce an output object. The values for Colhd are provided in the FILE statement in the DATA step that appears later in the program. Using dynamic column headings gives you more flexibility than does hardcoding the headers in the table template.

```
   dynamic colhd;
```

Control the repetition of values that do not change from one row to the next row. The CLASSLEVELS= attribute suppresses the display of the value in a column that is marked with BLANK_DUPS=ON if the value changes in a previous column that is also marked with BLANK_DUPS=ON. Because BLANK_DUPS= is set in a generic column, set this attribute as well.

```
   classlevels=on;
```

Create the column Char_Var. This DEFINE statement and its attributes create the column template Char_Var. GENERIC= specifies that multiple variables can use the same column template. BLANK_DUPS= suppresses the display of the value in the column if it does not change from one row to the next (and, because CLASSLEVELS=ON for the table, if no values in preceding columns that are marked with BLANK_DUPS=ON changes). HEADER= specifies that the header for the column will be the text of the dynamic variable Colhd, whose value will be set by the data component. The STYLE= attribute specifies that the style element for this column template is CellContents.The END statement ends the template.

```
   define column char_var;
      generic=on;
      blank_dups=on;
      header=colhd;
      style=cellcontents;
   end;
```

Create the column template Num_Var. This DEFINE statement and its attributes create the column template Num_Var. GENERIC= specifies that multiple variables can use the same column template. HEADER= specifies that the header for the column will be the text of the dynamic variable Colhd, whose value will be set by the data component. The STYLE= attribute specifies that the style element for this column template is CellContents.The END statement ends the template.

```
   define column num_var;
      generic=on;
      header=colhd;
      style=cellcontents;
```

```
   end;
```

Create the footer element Table_Footer. The DEFINE statement and its substatement define the table element Table_Footer. The FOOTER argument declares Table_Footer as a footer. The TEXT statement specifies the text of the footer. When ODS binds the data component to the table template (in the DATA step that follows), it will resolve the value of the macro variable SYSDATE9.

```
   define footer table_footer;
      text "Prepared on " sysdate9;
   end;
```

End the table template. This END statement ends the table template. The RUN statement executes the PROC TEMPLATE step.

```
   end;
run;
ods html close;
```

Create HTML output and specify the location for storing the HTML output. Specify the style to use for the output. The HTML destination is open by default. However, to specify a style, you must use the ODS HTML statement with the STYLE= open specified.. The STYLE= option tells ODS to use NewStyle as the style when it formats the output.

```
ods html file="file-path/newstyle-body.htm" style=newstyle;
```

Specify the titles for the report. The TITLE statements provide two titles for the output.

```
   title "Leading Grain Producers";
   title2 "in 1996";
```

Create the data component. This DATA step does not create a data set. Instead, it creates a data component and, eventually, an output object. The SET statement reads the data set Grain_Production. The WHERE statement subsets the data set so that the output object contains information only for rice and corn production in 1996.

```
data _null_;
   set grain_production;
   where type  in ("Rice", "Corn") and year=1996;
```

Route the DATA step results to ODS and use the Table1 table template. The combination of the fileref PRINT and the ODS option in the FILE statement routes the results of the DATA step to ODS. The TEMPLATE= suboption tells ODS to use the table template named Table1, which was previously created with PROC TEMPLATE.

```
   file print ods=(
        template="table1"
```

Specify the column template to use for each variable. The COLUMNS= suboption places DATA step variables into columns that are defined in the table template. For example, the first *column-specification* specifies that the first column of the output object contains the values of the variable COUNTRY and that it uses the column template named Char_Var. GENERIC= must be set to ON in both the table template and each column assignment in order for multiple variables to use the same column template. The FORMAT= suboption specifies a format for the column. The DYNAMIC= suboption provides the value of the dynamic variable Colhd for the current column. Notice that for the first column the column header is Country, and for the second column, which uses the same column template, the column header is Year.

```
          columns=(
             char_var=country(generic=on format=$cntry.
                      dynamic=(colhd="Country"))
             char_var=type(generic dynamic=(colhd="Year"))
             num_var=kilotons(generic=on format=comma12.
                     dynamic=(colhd="Kilotons"))
             )
          );
```

Write the data values to the data component. The _ODS_ option and the PUT statement write the data values for all columns to the data component. The RUN statement executes the DATA step.

```
   put _ods_;
run;
```

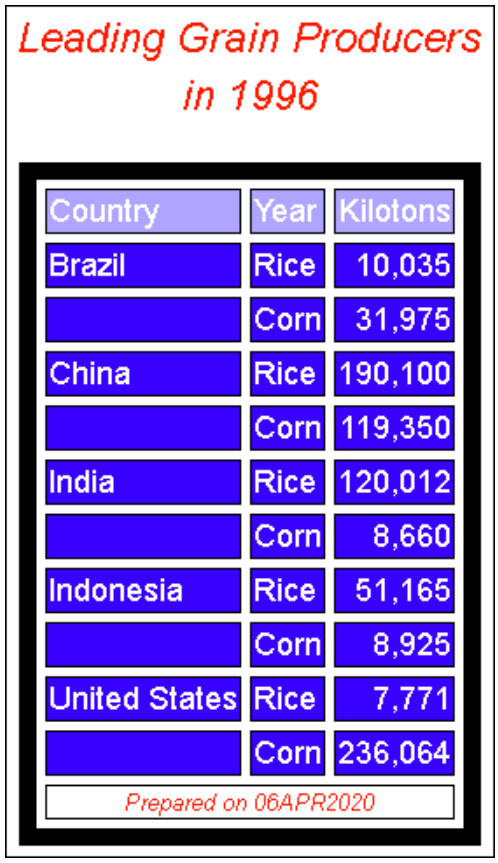
Stop the creation of the HTML output The ODS HTML statement closes the HTML destination and all the files that are associated with it. Close the destination so that you can view the output with a browser. The ODS HTML statement opens the HTML destination to return ODS to its default setup.

```
ods html close;
ods html;
```

HTML Output: Specifying Colors and Fonts with User-Defined Attributes

Use the fonts to confirm that SAS titles use the SystemTitle style element, that column headings use the Header style element, that the footer uses the Table-Footer style element, and that the contents of both character and numeric cells use the CellContents style element. Use the width of the table border and the spacing between cells to confirm that the table itself is produced with the Table style element.

Output 14.1 *HTML Output*



Leading Grain Producers in 1996

Country	Year	Kilotons
Brazil	Rice	10,035
	Corn	31,975
China	Rice	190,100
	Corn	119,350
India	Rice	120,012
	Corn	8,660
Indonesia	Rice	51,165
	Corn	8,925
United States	Rice	7,771
	Corn	236,064

Prepared on 06APR2020

Example 2: Using User-Defined Attributes

Features:

DEFINE STYLE statement:
- STYLE statement with user-defined attributes

DEFINE TABLE statement
- CLASSLEVELS= table attribute
- DYNAMIC statement
- MVAR statement

DEFINE COLUMN statement
- BLANK_DUPS=
- GENERIC=
- HEADER=
- STYLE=

DEFINE FOOTER statement:
- TEXT statement

Other ODS features:
- ODS HTML statement
- FILE statement with ODS= option
- PUT statement with _ODS_ argument

Data set: Grain_Production

Format: $CNTRY.

Program 1: Details

This example creates a style that is equivalent to the style that “Example 1: Creating a Stand-Alone Style” on page 466 creates. However, this style uses user-defined attributes to specify colors and fonts. This technique makes it possible to easily make changes in multiple places in the output.

Program 1: Creating a Custom Style with User-Defined Style Attributes

```
proc template;
   define style newstyle2;
      style fonts /
         "cellfont"=("arial, helvetica", 4, medium roman)
         "headingfont"=("arial, helvetica", 5, bold roman)
         "titlefont"=("arial, helvetica", 6, bold italic);

      style colors /
         "light"=white
         "medium"=cxaaaaff
         "dark"=cx0000ff
         "bright"=red;

      style cellcontents /
         backgroundcolor=colors("dark")
         color=colors("light")
         font=fonts("cellfont");
      style header /
         backgroundcolor=colors("medium")
         color=colors("dark")
         font=fonts("headingfont");
      style systemtitle /
         backgroundcolor=colors("light")
         color=colors("bright")
         font=fonts("titlefont");
      style footer from systemtitle /
         fontsize=3;
      style table /
         borderspacing=5
         borderwidth=10;

   end;
run;

proc template;
   define table table1;

   mvar sysdate9;

   dynamic colhd;

   classlevels=on;
```

```
   define column char_var;
      generic=on;
      blank_dups=on;
      header=colhd;
      style=cellcontents;
   end;

   define column num_var;
      generic=on;
      header=colhd;
      style=cellcontents;
   end;

   define footer table_footer;
      text "Prepared on"  sysdate9;
   end;

   end;
run;

ods html body="file-path/newstyle2-body.htm"
         style=newstyle2;

   title "Leading Grain Producers";
   title2 "in 1996";

data _null_;
   set grain_production;
   where type  in ("Rice", "Corn") and year=1996;

   file print ods=(
        template="table1"

        columns=(
           char_var=country(generic=on format=$cntry.
                    dynamic=(colhd="Country"))
           char_var=type(generic dynamic=(colhd="Year"))
           num_var=kilotons(generic=on format=comma12.
                   dynamic=(colhd="Kilotons"))
           )
        );

   put _ods_;
run;

ods html close;
ods html;
```

Program Description

Create the style NewStyle2. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style called NewStyle2. This STYLE statement defines the style element Fonts. This style element consists of three user-defined attributes: CellFont, HeadingFont, and TitleFont. Each of these attributes describes a font. This style specifies the fontfamily, fontsize, fontweight, and the fontstyle for each of the three attributes. The font and fontwidth attributes are still defined by the default style.

```
proc template;
```

```
define style newstyle2;
   style fonts /
      "cellfont"=("arial, helvetica", 4, medium roman)
      "headingfont"=("arial, helvetica", 5, bold roman)
      "titlefont"=("arial, helvetica", 6, bold italic);
```

Create the style element Colors. This STYLE statement defines the style element Colors. This style element consists of four user-defined attributes: light, medium, dark, and bright. The values for medium and dark are RGB values equivalent to very light blue and blue.

```
   style colors /
      "light"=white
      "medium"=cxaaaaff
      "dark"=cx0000ff
      "bright"=red;
```

Create the three style elements CellContents, Header, and SystemTitle. Create the style element Footer using inheritance. The style attributes are defined in terms of the user-defined attributes that were created earlier in the style. For example, the foreground color in CellContents is set to colors("light"). Looking at the template of Colors, you can see that this is white. However, by setting the colors up in a style element with user-defined attributes, you can change the color of everything that uses a particular color by changing a single value in the style element Colors.

```
   style cellcontents /
      backgroundcolor=colors("dark")
      color=colors("light")
      font=fonts("cellfont");
   style header /
      backgroundcolor=colors("medium")
      color=colors("dark")
      font=fonts("headingfont");
   style systemtitle /
      backgroundcolor=colors("light")
      color=colors("bright")
      font=fonts("titlefont");
   style footer from systemtitle /
      fontsize=3;
   style table /
      borderspacing=5
      borderwidth=10;
```

End the style. The END statement ends the style. The RUN statement executes PROC TEMPLATE.

```
   end;
run;
```

Create the table template Table1. The DEFINE TABLE statement creates a new table template called Table1.

```
proc template;
   define table table1;
```

Specify the symbol that references one macro variable. The MVAR statement defines a symbol, Sysdate9, that references a macro variable. ODS will use the value of this macro variable as a string. References to the macro variable are resolved when ODS binds the table template to the data component to produce an output object. SYSDATE9 is an automatic macro variable whose value is always available.

```
mvar sysdate9;
```

Specify the symbol that references a value to be supplied by the data component. The DYNAMIC statement defines a symbol, Colhd, that references a value that the data component supplies when ODS binds the template and the data component to produce an output object. The values for Colhd are provided in the FILE statement in the DATA step that appears later in the program. Using dynamic column headings gives you more flexibility than hardcoding the headers in the table template does.

```
dynamic colhd;
```

Control the repetition of values that do not change from one row to the next row. The CLASSLEVELS= attribute suppresses the display of the value in a column that is marked with BLANK_DUPS=ON if the value changes in a previous column that is also marked with BLANK_DUPS=ON. Because BLANK_DUPS= is set in a generic column, set this attribute as well.

```
classlevels=on;
```

Create the column Char_Var. This DEFINE statement and its attributes create the column template Char_Var. GENERIC= specifies that multiple variables can use the same column template. BLANK_DUPS= suppresses the display of the value in the column if it does not change from one row to the next (and, because CLASSLEVELS=ON for the table, if no values in preceding columns that are marked with BLANK_DUPS=ON changes). HEADER= specifies that the header for the column will be the text of the dynamic variable Colhd, whose value will be set by the data component. The STYLE= attribute specifies that the style element for this column template is CellContents.The END statement ends the template.

```
define column char_var;
   generic=on;
   blank_dups=on;
   header=colhd;
   style=cellcontents;
end;
```

Create the column Num_Var. This DEFINE statement and its attributes create the column template Num_Var. GENERIC= specifies that multiple variables can use the same column template. HEADER= specifies that the header for the column will be the text of the dynamic variable Colhd, whose value will be set by the data component. The STYLE= attribute specifies that the style element for this column template is CellContents.The END statement ends the template.

```
define column num_var;
   generic=on;
   header=colhd;
   style=cellcontents;
end;
```

Create the footer element Table_Footer. The DEFINE statement and its substatement define the table element Table_Footer. The FOOTER argument declares Table_Footer as a footer. The TEXT statement specifies the text of the footer. When ODS binds the data component to the table template (in the DATA step that follows), it will resolve the value of the macro variable SYSDATE9.

```
define footer table_footer;
   text "Prepared on"  sysdate9;
end;
```

End the table template. This END statement ends the table template. The RUN statement executes the PROC TEMPLATE step.

```
   end;
run;
```

Create HTML output and specify the location for storing the HTML output. Specify the style to use for the output. The HTML destination is open by default. However, to specify a style, you must use the ODS HTML statement with the STYLE= open specified. The STYLE= option tells ODS to use NewStyle2 as the style when it formats the output.

```
ods html body="file-path/newstyle2-body.htm"
         style=newstyle2;
```

Specify the titles for the report. The TITLE statements provide two titles for the output.

```
   title "Leading Grain Producers";
   title2 "in 1996";
```

Create the data component. This DATA step does not create a data set. Instead, it creates a data component and, eventually, an output object. The SET statement reads the data set Grain_Production. The WHERE statement subsets the data set so that the output object contains information only for rice and corn production in 1996.

```
data _null_;
   set grain_production;
   where type  in ("Rice", "Corn") and year=1996;
```

Route the DATA step results to ODS and use the Table1 table template. The combination of the fileref PRINT and the ODS option in the FILE statement routes the results of the DATA step to ODS. The TEMPLATE= suboption tells ODS to use the table template named Table1, which was previously created with PROC TEMPLATE.

For more information about using the DATA step with ODS, see “Using ODS with the DATA Step” in *SAS Output Delivery System: User’s Guide*.

```
   file print ods=(
        template="table1"
```

Specify the column template to use for each variable. The COLUMNS= suboption places DATA step variables into columns that are defined in the table template. For example, the first *column-specification* specifies that the first column of the output object contains the values of the variable COUNTRY and that it uses the column template named Char_Var. GENERIC= must be set to ON in both the table template and each column assignment in order for multiple variables to use the same column template. The FORMAT= suboption specifies a format for the column. The DYNAMIC= suboption provides the value of the dynamic variable Colhd for the current column. Notice that for the first column the column header is **`Country`**, and for the second column, which uses the same column template, the column header is **`Year`**.

```
        columns=(
           char_var=country(generic=on format=$cntry.
                    dynamic=(colhd="Country"))
           char_var=type(generic dynamic=(colhd="Year"))
           num_var=kilotons(generic=on format=comma12.
                   dynamic=(colhd="Kilotons"))
           )
        );
```

Write the data values to the data component. The _ODS_ option and the PUT statement write the data values for all columns to the data component. The RUN statement executes the DATA step.

```
   put _ods_;
run;
```

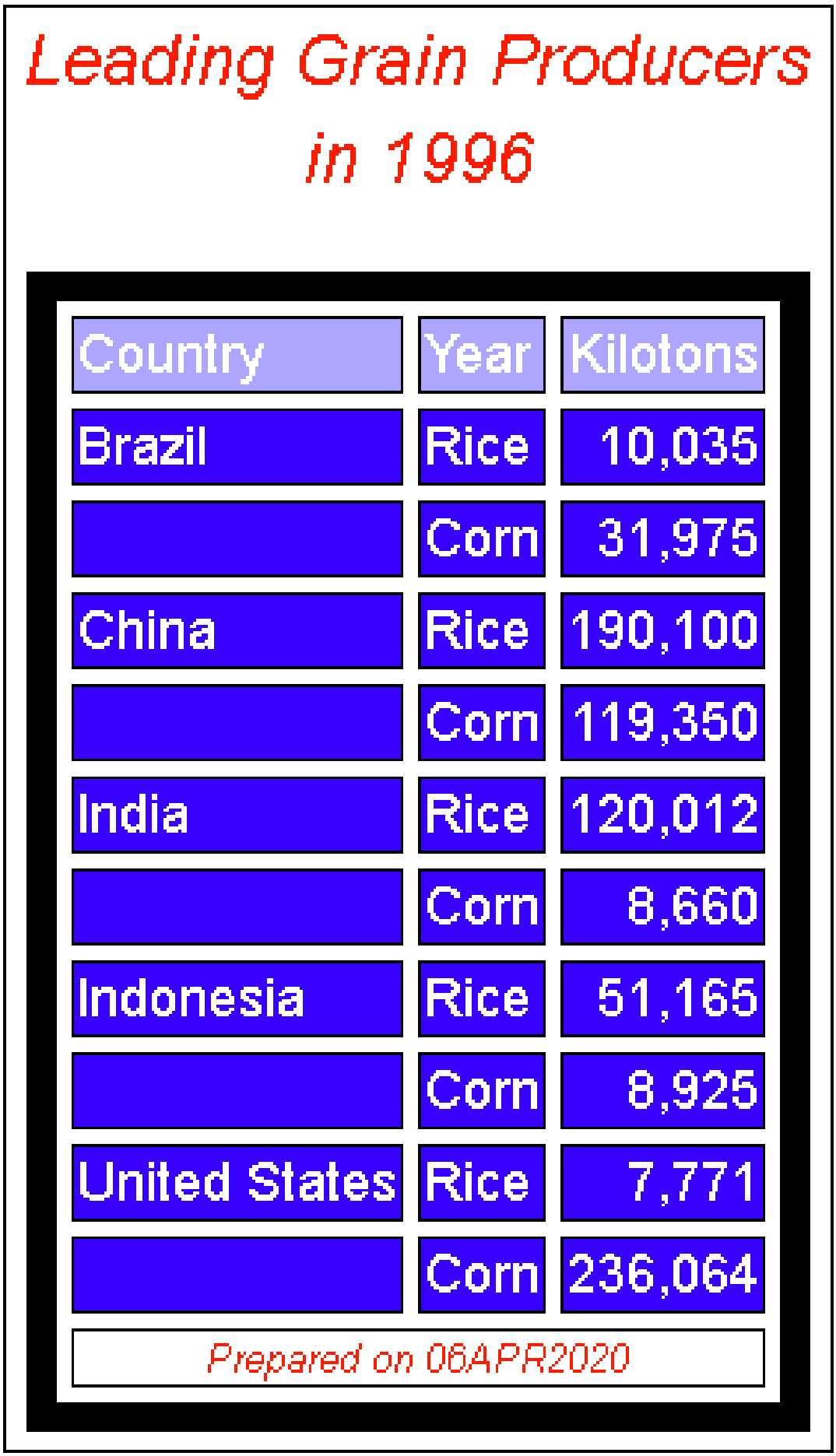
Close the HTML destination. The ODS HTML statement closes the HTML destination and all the files that are associated with it. The ODS HTML statement opens the HTML destination to return ODS to its default setup.

```
ods html close;
ods html;
```

HTML Output

This HTML output is identical to the output in the section "HTML Output: Specifying Colors and Fonts with User-Defined Attributes" on page 472, which was produced with a style that used predefined style attributes. You can use the fonts to confirm that SAS titles use the SystemTitle style element, that column headings use the Header style element, that the footer uses the Table-Footer style element, and that the contents of both character and numeric cells use the CellContents style element. Use the width of the table border and the spacing between cells to confirm that the table produced with the Table style element.

Output 14.2 HTML Output



Program 2: Changing the Color Scheme

```
style colors /
   "light"=white
   "medium"=cxaaaaff
   "dark"=cx0000ff
   "bright"=red;

style colors /
   "light"=white
   "medium"=pink
   "dark"=red
   "bright"=red;

   "cellfont"=("arial, helvetica", 4, medium roman)

   "cellfont"=("courier, arial, helvetica", 4, medium roman)
```

Program Description

In the program “Example 1: Creating a Stand-Alone Style” on page 466, to change the color scheme so that the blues are replaced by pink and red, change each occurrence of "blue" and "very light blue." In this program, because colors are defined as user-defined attributes, make the change only once.

To make the color scheme change, change only this section of code:

The following is the original portion on code from “Program 1: Creating a Custom Style with User-Defined Style Attributes” on page 474.

```
style colors /
   "light"=white
   "medium"=cxaaaaff
   "dark"=cx0000ff
   "bright"=red;
```

Change the attributes as follows:

```
style colors /
   "light"=white
   "medium"=pink
   "dark"=red
   "bright"=red;
```

Similarly, to change the font in any style element that uses CellFont, change this section of code:

```
   "cellfont"=("arial, helvetica", 4, medium roman)
```

Here is one example of how to change the code:

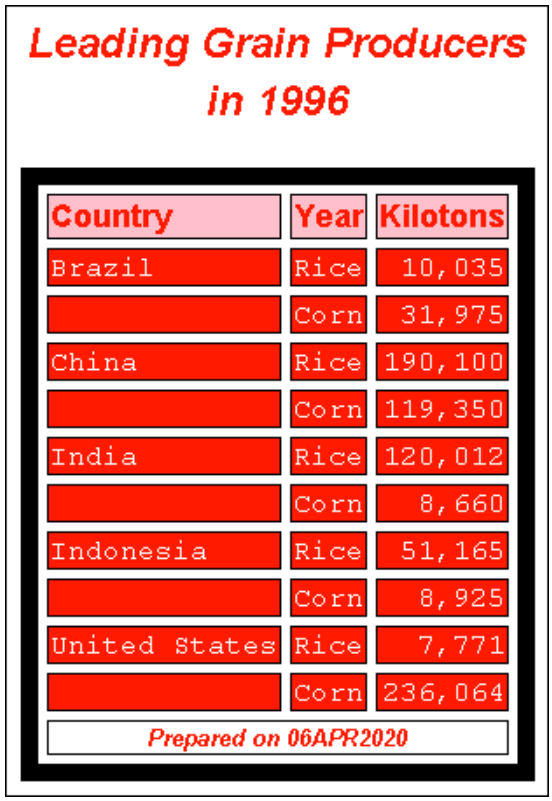
```
   "cellfont"=("courier, arial, helvetica", 4, medium roman)
```

HTML Output: Changing Colors and Fonts of User-Defined Attributes

This HTML output shows the results of running the same program with these changes.

The font in the cells is now Courier. This change occurs in multiple places even though you made only one change to the code for the font.

Output 14.3 *HTML Output with Changed Colors and Fonts*



Leading Grain Producers in 1996

Country	Year	Kilotons
Brazil	Rice	10,035
	Corn	31,975
China	Rice	190,100
	Corn	119,350
India	Rice	120,012
	Corn	8,660
Indonesia	Rice	51,165
	Corn	8,925
United States	Rice	7,771
	Corn	236,064

Prepared on 06APR2020

Example 3: Modifying the Default Style with the CLASS Statement

Features: DEFINE STYLE statement
- User-defined attributes
- BACKGROUNDCOLOR= style attribute
- BORDERWIDTH= style attribute
- CELLPADDING= style attribute
- BORDERSPACING= style attribute
- COLOR= style attribute
- FONT= style attribute
- FONTSTYLE= style attribute
- FRAME= style attribute
- POSTHTML= style attribute

RULES= style attribute
VISITEDLINKCOLOR= style attribute

CLASS statement

PARENT= statement

Other ODS features
ODS HTML statement: STYLE= option
ODS PATH statement

Data set: Energy

Format: DIVFMT. and USETYPE.

Details

When you are working with styles, you are more likely to modify a SAS style than to write a completely new style. This example makes changes to the default style for the HTML destination. The new style affects both the contents file and the body file in the HTML output. In the contents file, the modified style makes changes to the following:

- the text of the header and the text that identifies the procedure that produced the output
- the colors for some parts of the text
- the font size of some parts of the text
- the spacing in the list of entries in the table of contents

In the body file, the modified style makes changes to the following:

- two of the colors in the color list. Style1 of these colors is the foreground color for the table of contents, the BY line, and column headings. The other is the foreground of many parts of the body file, including SAS titles and footnotes.
- the font size for titles and footnotes.
- the font style for headers.
- the presentation of the data in the table by changing attributes such as border spacing, rules, and border width.

When you modify a style element in a new style that has a like-named style element in the parent style, you must use the CLASS statement or the STYLE statement with the FROM option specified. This example uses the CLASS statement to produce a shorter, easier to read program.

Program

```
proc template;
   define style customdefault;

   parent=styles.htmlblue;

      class contents  /
            background=cxffffcc;
```

```
      class contenttitle /
            background=cxffffcc;

       class  data /
             background=cxcccccc;
   style IndexProcName from Index /

         backgroundcolor = cxffffcc;

      class colors /
         'link2' =
cx0000FF
         'link1' =
cx800080
         'docbg' =
cx99ccff
         'contentbg' =
cxFAFBFE
         'systitlebg' =
cx99ccff
         'titlebg' =
cxFAFBFE
         'proctitlebg' =
cxFAFBFE
         'headerbg' =
cxEDF2F9
         'captionbg' =
cxFAFBFE
         'captionfg' =
cx112277
         'bylinebg' =
cx99ccff
         'notebg' =
cxFAFBFE
         'tablebg' =
cxFAFBFE
         'batchbg' =
cxFAFBFE
         'systitlefg' =
cx112277
         'titlefg' =
cx112277
         'proctitlefg' =
cx112277
         'bylinefg' =
cx112277
         'notefg' = cx112277;

   class Header   /
         bordercolor =
cxEDF2F9
         backgroundcolor =
cxEDF2F9
         color = cx112277;

   class text /
      "prefix1" = "PROC "
```

```
      "suffix1" = ":"
      "Content Title" = "Contents"
      "Pages Title" = "Pages"
      ;
   end;
run;
ods html close;

ods html body="file-path/customdefaultstyle-body.htm"
         contents="file-path/customdefaultstyle-content.htm"
         frame="file-path/customdefaultstyle-frame.htm"
         style=customdefault;

title "MSRP by Make and Model";
title2 "(6 Cylinders Only)";

proc print data=sashelp.cars noobs;
   var make model cylinders MSRP;
   format MSRP dollar10.2;
   by make;
   where cylinders=6;
run;

ods html close;
ods html;
```

Program Description

Create the style CustomDefault. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style called CustomDefault.

```
proc template;
   define style customdefault;
```

Specify the parent style from which the CustomDefault style inherits its attributes. The PARENT= attribute specifies Styles.HTMLBlue as the style from which the current style inherits. All the style elements, attributes, and statements that are specified in the parent's style template are used in the child style template unless the child style template overrides them.

```
   parent=styles.htmlblue;
```

Customize the contents style elements and the data cells. By changing the BACKGROUND= style attribute in the style elements Contents, ContentTitle, and IndexProcName, the background of the contents becomes yellow.

```
      class contents  /
            background=cxffffcc;

      class contenttitle /
            background=cxffffcc;

       class  data /
             background=cxcccccc;
   style IndexProcName from Index /

         backgroundcolor = cxffffcc;
```

Change the attributes of the style element Colors This CLASS statement adds to the child style the style element Colors, which also exists in the parent style (HTMLBlue). The CLASS statement adds all of the style attributes that are in the original instance of the Colorsstyle element to the new instance of Colors, except for those that are overridden by the new instance of Colors. By using the CLASS statement, you do not need to specify the FROM option. If you did not use the CLASS statement or the FROM option, then the attributes from the original instance of Colors would not be added to the new instance of Colors. The Colors style element in CustomDefault would contain only the style statements that it specifically specifies. All style elements that use the user-defined attributes that Colors defines (fgB2, fgB1, and so on) use the style attributes that are specified in Custom.Default, not the ones that are specified in Styles.HTMLBlue. Therefore, if you change a color here, then you change every occurrence of the color in the HTML output. This CLASS statement changes the values of three of the user-defined style attributes: Docbg=, Systitlebg=, and Bylinebg=.

```
        class colors /
           'link2' =
   cx0000FF
           'link1' =
   cx800080
           'docbg' =
   cx99ccff
           'contentbg' =
   cxFAFBFE
           'systitlebg' =
   cx99ccff
           'titlebg' =
   cxFAFBFE
           'proctitlebg' =
   cxFAFBFE
           'headerbg' =
   cxEDF2F9
           'captionbg' =
   cxFAFBFE
           'captionfg' =
   cx112277
           'bylinebg' =
   cx99ccff
           'notebg' =
   cxFAFBFE
           'tablebg' =
   cxFAFBFE
           'batchbg' =
   cxFAFBFE
           'systitlefg' =
   cx112277
           'titlefg' =
   cx112277
           'proctitlefg' =
   cx112277
           'bylinefg' =
   cx112277
           'notefg' = cx112277;
```

Change the style attributes in the style element Header. This STYLE statement adds the italic font style to the attributes that Header inherits from the Header style

element that is defined in the parent style. You could have also specified the STYLE statement with the FROM option specified. Because this change occurs after the initial merge of the two styles, the change will effect HeaderFixed and the other style elements that inherit from Header in the parent style.

In the default style, the background color for the BY line differs from the background color for the document, so it appears as a gray stripe in the default output. In this customized style, the stripe disappears because the background color for the BY line and the document are the same.

```
      class Header   /
            bordercolor =
   cxEDF2F9
            backgroundcolor =
   cxEDF2F9
            color = cx112277;
```

Customize the text used in parts of the output. In the customized style, the text that identifies the output reads "1. PROC PRINT". The heading that appears at the top of the contents file has been changed from "Table of Contents" to "Contents", and the heading at the top of the table of pages has been changed from "Table of Pages" to "Pages". The banners have been changed to use mixed case. (Note that neither these banners nor the table of pages is visible in the HTML output from this example, but the attributes are included so that you can use the style in a variety of circumstances.)

This CLASS statement alters the text that is used in parts of the HTML output. In the contents file, the default style uses "The" as the value of prefix1 and "Procedure" as the value of suffix1. Thus, in HTML output that uses the default style, the output from PROC PRINT is identified by "1. The PRINT Procedure".

```
      class text /
         "prefix1" = "PROC "
         "suffix1" = ":"
         "Content Title" = "Contents"
         "Pages Title" = "Pages"
         ;
```

Stop the creation of the customized style. The END statement ends the style. The RUN statement executes the PROC TEMPLATE step.

```
      end;
   run;
   ods html close;
```

Create the HTML output and specify the style to use for the output. The ODS HTML statement opens the HTML destination and creates HTML output. The output from PROC PRINT is sent to the body file. FRAME= and CONTENTS= create a frame that includes a table of contents that links to the contents of the body file. The body file also appears in the frame. The STYLE= option tells ODS to use CustomDefault as the style when it formats the output.

```
   ods html body="file-path/customdefaultstyle-body.htm"
            contents="file-path/customdefaultstyle-content.htm"
            frame="file-path/customdefaultstyle-frame.htm"
            style=customdefault;
```

Specify the titles and footnote for the report. The TITLE and FOOTNOTE statements provide two titles and a footnote for the output.

```
   title "MSRP by Make and Model";
   title2 "(6 Cylinders Only)";
```

Create the PRINT procedure output. This PROC PRINT step is the same one that was used with the default style in the previous program.

```
proc print data=sashelp.cars noobs;
   var make model cylinders MSRP;
   format MSRP dollar10.2;
   by make;
   where cylinders=6;
run;
```

Close the HTML destination. The ODS HTML statement closes the HTML destination and all the files that are associated with it. The ODS HTML statement opens the HTML destination to return ODS to its default setup.

```
ods html close;
ods html;
```

HTML Output

Output 14.4 *HTML Output from PROC PRINT Using the Default Style*

MSRP by Make and Model
(6 Cylinders Only)

Make=Acura

Make	Model	Cylinders	MSRP
Acura	MDX	6	$36,945.00
Acura	TL 4dr	6	$33,195.00
Acura	3.5 RL 4dr	6	$43,755.00
Acura	3.5 RL w/Navigation 4dr	6	$46,100.00
Acura	NSX coupe 2dr manual S	6	$89,765.00

Make=Audi

Make	Model	Cylinders	MSRP
Audi	A4 3.0 4dr	6	$31,840.00
Audi	A4 3.0 Quattro 4dr manual	6	$33,430.00
Audi	A4 3.0 Quattro 4dr auto	6	$34,480.00
Audi	A6 3.0 4dr	6	$36,640.00
Audi	A6 3.0 Quattro 4dr	6	$39,640.00
Audi	A4 3.0 convertible 2dr	6	$42,490.00
Audi	A4 3.0 Quattro convertible 2dr	6	$44,240.00
Audi	A6 2.7 Turbo Quattro 4dr	6	$42,840.00
Audi	TT 3.2 coupe 2dr (convertible)	6	$40,590.00
Audi	A6 3.0 Avant Quattro	6	$40,840.00

Output 14.5 *HTML Output from PROC PRINT with the Customized Style*

MSRP by Make and Model
(6 Cylinders Only)

Make=Acura

Make	Model	Cylinders	MSRP
Acura	MDX	6	$36,945.00
Acura	TL 4dr	6	$33,195.00
Acura	3.5 RL 4dr	6	$43,755.00
Acura	3.5 RL w/Navigation 4dr	6	$46,100.00
Acura	NSX coupe 2dr manual S	6	$89,765.00

Make=Audi

Make	Model	Cylinders	MSRP
Audi	A4 3.0 4dr	6	$31,840.00
Audi	A4 3.0 Quattro 4dr manual	6	$33,430.00
Audi	A4 3.0 Quattro 4dr auto	6	$34,480.00
Audi	A6 3.0 4dr	6	$36,640.00
Audi	A6 3.0 Quattro 4dr	6	$39,640.00
Audi	A4 3.0 convertible 2dr	6	$42,490.00
Audi	A4 3.0 Quattro convertible 2dr	6	$44,240.00
Audi	A6 2.7 Turbo Quattro 4dr	6	$42,840.00
Audi	TT 3.2 coupe 2dr (convertible)	6	$40,590.00
Audi	A6 3.0 Avant Quattro	6	$40,840.00

Example 4: Defining Multiple Style Elements in One STYLE Statement

Features: DEFINE STYLE statement
- FROM option
- Style attribute: BACKGROUNDCOLOR=
- Style attribute: BORDERWIDTH=
- Style attribute: BORDERSPACING=
- Style attribute: FONTFAMILY=
- Style attribute: FONTSIZE=
- Style attribute: FONTSTYLE=
- Style attribute: FONTWEIGHT=
- Style attribute: COLOR=

CLASS statement

Other ODS features
- ODS HTML statement
- FILE statement with ODS= option
- PUT statement with _ODS_ argument

Data set: Grain_Production

Format: $CNTRY.

Details

This example creates a style that defines multiple style elements concurrently. When style element names are specified multiple times, all of the attributes from all instances of that name are collected to create the final set of style attributes. Defining multiple style elements in one STYLE statement makes it possible to create shorter, easier to read programs and to make changes to style attributes in a single STYLE statement rather than in many STYLE statements.

For example, if you wanted to add the style element BorderColor=black to the style elements CellContents, Header, and SystemTitle in the program below, you could add it once, to the first STYLE statement, instead of adding it three times, to each individual STYLE statement.

Program

```
filename odsout "your-directory";
ods html close;
options nodate;

proc template;
   define style Styles.NewStyle;

      style cellcontents, header, systemtitle /
         fontfamily="arial, helvetica"
         fontweight=medium
         backgroundcolor=blue
         fontstyle=roman
         fontsize=5
         color=white;

      class header /
         backgroundcolor=very light blue;

      class systemtitle /
         backgroundcolor=white
         color=red
         fontstyle=italic
         fontsize=6;

      style footer from systemtitle /
         fontsize=3;

      class table /
         borderspacing=5
         borderwidth=10;
      end;
run;

ods html path=odsout file="newstyle-body.htm"
         style=newstyle;
```

```
   title "Leading Grain Producers";
   title2 "in 1996";

data _null_;
   set grain_production;
   where type  in ("Rice", "Corn") and year=1996;

   file print ods=(
        template="table1"

        columns=(
           char_var=country(generic=on format=$cntry.
                    dynamic=(colhd="Country"))
           char_var=type(generic dynamic=(colhd="Year"))
           num_var=kilotons(generic=on format=comma12.
                   dynamic=(colhd="Kilotons"))
           )
        );

   put _ods_;
run;

ods html close;
ods html;

proc template;
delete Styles.NewStyle;
run;
```

Program Description

Create a file reference for the output and set the SAS system options.

```
filename odsout "your-directory";
ods html close;
options nodate;
```

Create a new style Styles.NewStyle. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style called Styles.NewStyle.

```
proc template;
   define style Styles.NewStyle;
```

Create the CellContents, Header, and SystemTitle style elements. This STYLE statement defines three style elements: CellContents, Header, and SystemTitle. They are all composed of the style attributes that appear in the STYLE statement. The FONTFAMILY= attribute tells the browser to use the Arial font if it is available, and to look for the Helvetica font if Arial is not available. These three style elements use a color scheme of a white foreground on a blue background, and the font for all three is medium roman with a size of five.

```
      style cellcontents, header, systemtitle /
         fontfamily="arial, helvetica"
         fontweight=medium
         backgroundcolor=blue
         fontstyle=roman
         fontsize=5
         color=white;
```

Modify the Header style element. The STYLE statement with the FROM option specified creates the new instance of Header from the previous instance of Header, but changes the background color from white to very light blue. By default, ODS uses Header to produce both spanning headers and column headings.

```
        class header /
           backgroundcolor=very light blue;
```

Modify the SystemTitle style element. By default, ODS uses SystemTitle to produce SAS titles.

```
        class systemtitle /
           backgroundcolor=white
           color=red
           fontstyle=italic
           fontsize=6;
```

Create the style element Footer. This STYLE statement creates the style element Footer. This style element inherits all the attributes of SystemTitle. However, the font size that it inherits is overwritten by the FONTSIZE= attribute in its template.

```
        style footer from systemtitle /
           fontsize=3;
```

Create the style element Table. This STYLE statement creates the style element Table. By default, ODS uses this style element to display tables.

```
        class table /
           borderspacing=5
           borderwidth=10;
        end;
run;
```

Create HTML output and specify the location for storing the HTML output. Specify the style to use for the output. The ODS HTML statement opens the HTML destination and creates HTML output. It sends all output objects to the external file NewStyle-Body in the current directory. The STYLE= option tells ODS to use Styles.NewStyle as the style when it formats the output.

```
ods html path=odsout file="newstyle-body.htm"
         style=newstyle;
```

Specify the titles for the report. The TITLE statements provide two titles for the output.

```
    title "Leading Grain Producers";
    title2 "in 1996";
```

Create the data component. This DATA step does not create a data set. Instead, it creates a data component and, eventually, an output object. The SET statement reads the data set Grain_Production. The WHERE statement subsets the data set so that the output object contains information only for rice and corn production in 1996.

```
data _null_;
   set grain_production;
   where type  in ("Rice", "Corn") and year=1996;
```

Route the DATA step results to ODS and use the Table1 table template. The combination of the fileref PRINT and the ODS option in the FILE statement routes the results of the DATA step to ODS. The TEMPLATE= suboption tells ODS to use the table template named Table1, which was previously created with PROC TEMPLATE.

For more information about using the DATA step with ODS, see "Using ODS with the DATA Step" in *SAS Output Delivery System: User's Guide*. For the program that creates the table template Table1, see "Creating the Table1 Table Template" in *SAS Output Delivery System: User's Guide*.

```
file print ods=(
     template="table1"
```

Specify the column template to use for each variable. The COLUMNS= suboption places DATA step variables into columns that are defined in the table template. For example, the first *column-specification* specifies that the first column of the output object contains the values of the variable COUNTRY and that it uses the column template named Char_Var. GENERIC= must be set to ON in both the table template and each column assignment in order for multiple variables to use the same column template. The FORMAT= suboption specifies a format for the column. The DYNAMIC= suboption provides the value of the dynamic variable Colhd for the current column. Notice that for the first column the column header is Country, and for the second column, which uses the same column template, the column header is Year.

```
columns=(
   char_var=country(generic=on format=$cntry.
            dynamic=(colhd="Country"))
   char_var=type(generic dynamic=(colhd="Year"))
   num_var=kilotons(generic=on format=comma12.
           dynamic=(colhd="Kilotons"))
   )
);
```

Write the data values to the data component. The _ODS_ option and the PUT statement write the data values for all columns to the data component. The RUN statement executes the DATA step.

```
   put _ods_;
run;
```

Close the HTML destination. The ODS HTML statement closes the HTML destination and all the files that are associated with it. Specify the ODS HTML statement again to return ODS to its default setup.

```
ods html close;
ods html;
```

Once you have created your output, you can delete the custom styles.

```
proc template;
delete Styles.NewStyle;
run;
```

HTML Output: Specifying Colors and Fonts

You can use the fonts to confirm that SAS titles use the SystemTitle style element, that column headings use the Header style element, that the footer uses the Table-Footer style element, and that the contents of both character and numeric cells use the CellContents style element. Use the width of the table border and the spacing between cells to confirm that the table itself is produced with the Table style element.

Output 14.6 HTML Output

Leading Grain Producers in 1996

Country	Year	Kilotons
Brazil	Rice	10,035
	Corn	31,975
China	Rice	190,100
	Corn	119,350
India	Rice	120,012
	Corn	8,660
Indonesia	Rice	51,165
	Corn	8,925
United States	Rice	7,771
	Corn	236,064

Prepared on 06APR2020

Example 5: Importing a CSS File

Features:

DEFINE STYLE statement
- CLASS statement
- IMPORT statement: *media-type*
- PARENT= statement

Other ODS features
- ODS HTML statement
- ODS PDF statement
- ODS _ALL_ CLOSE statement

Details

The following program imports the external CSS file StyleSheet.css and converts the CSS code into style elements and style attributes. These style elements and attributes then become part of the style.

Your CSS file can contain media blocks that correspond to the type of media that your output will be rendered on. The IMPORT statement enables you to specify one or more media blocks to be imported along with the rest of the CSS code. In this example, the Print media block is included in the style that is applied to the PDF output.

The following code is an example of the external CSS file StyleSheet.css. There are two media type blocks specified in this program, Print and Screen. Copy and paste this code into a text editor and save it as StyleSheet.css.

```
.body {
   background-color: white;
    color: black;
    font-family: times, serif;
}
.header, .rowheader, .footer, .rowfooter, .data {
    border: 1px black solid;
    color: black;
    padding: 5px;
    font-family: times, serif;
}
.header, .rowheader, .footer, .rowfooter {
    background-color: #a0a0a0;
}
.table {
    background-color: #dddddd;
    border-spacing: 0;
    border: 1px black solid;
}
.proctitle {
    font-family: helvetica, sans-serif;
    font-size: x-large;
    font-weight: normal;
```

```
}

@media screen {

   .header, .rowheader, .footer, .rowfooter,{
      color: white;
      background-color: green;}
   .table {
      background-color: yellow;
      border-spacing: 0;
      font-size: small
      border: 1px black solid;

}
}@media print {

  .header, .rowheader, .footer, .rowfooter,{
      color: white;
      background-color: Blue;
      padding: 5px;
}
  .data {
      font-size: small;
}
}
```

Program

```
filename odsout "your-directory";
ods html close;
options nodate;

proc template;
define style styles.mycssstyle;
    import "StyleSheet.css";
    class data /
        color = red;
end;

define style styles.mycssstyleprinter;
       parent=styles.mycssstyle;
       import "StyleSheet.css" print;
end;
run;

ods html path=odsout file="file-path/css.html" style=styles.mycssstyle;
ods pdf file="file-path/css.pdf" style=styles.mycssstyleprinter;

proc contents data=sashelp.class;
run;

ods _all_ close;
ods html;
```

Program Description

Create a file reference for the output and set the SAS system options.

```
filename odsout "your-directory";
ods html close;
options nodate;
```

Define a style that imports a CSS file and defines style elements as well. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style called MyCssStyle. The IMPORT statement imports the CSS file StyleSheet.css, and converts the CSS code into ODS style elements and style attributes. Because no *media-type* option is specified, the Screen media block is imported along with the CSS code that is not in any media blocks. The Print media block is not imported. The CLASS statement specifies a red font color in the Data style element.

Specifying `class data / color=red;` is the same as specifying `style data from data / color=red;`.

```
proc template;
define style styles.mycssstyle;
    import "StyleSheet.css";
    class data /
        color = red;
end;
```

Define a style that imports a CSS file that includes a specific media type templates. The DEFINE STYLE statement creates a new style called MyCssStylePrinter. The IMPORT statement imports the CSS file StyleSheet.css, and converts the CSS code into ODS style elements and style attributes. The Print option specifies that the Print media block be imported along with the CSS code that is not in any media blocks. The code in the Screen media block is not imported.

```
define style styles.mycssstyleprinter;
        parent=styles.mycssstyle;
        import "StyleSheet.css" print;
end;
run;
```

Create HTML and PDF output and view the contents of the SAS data set. The ODS HTML and ODS PDF statements specify the destination to write to, the filename of the output, and the style to use. The CONTENTS procedure shows the contents of the SAS data set Sashelp.Class.

```
ods html path=odsout file="file-path/css.html" style=styles.mycssstyle;
ods pdf file="file-path/css.pdf" style=styles.mycssstyleprinter;

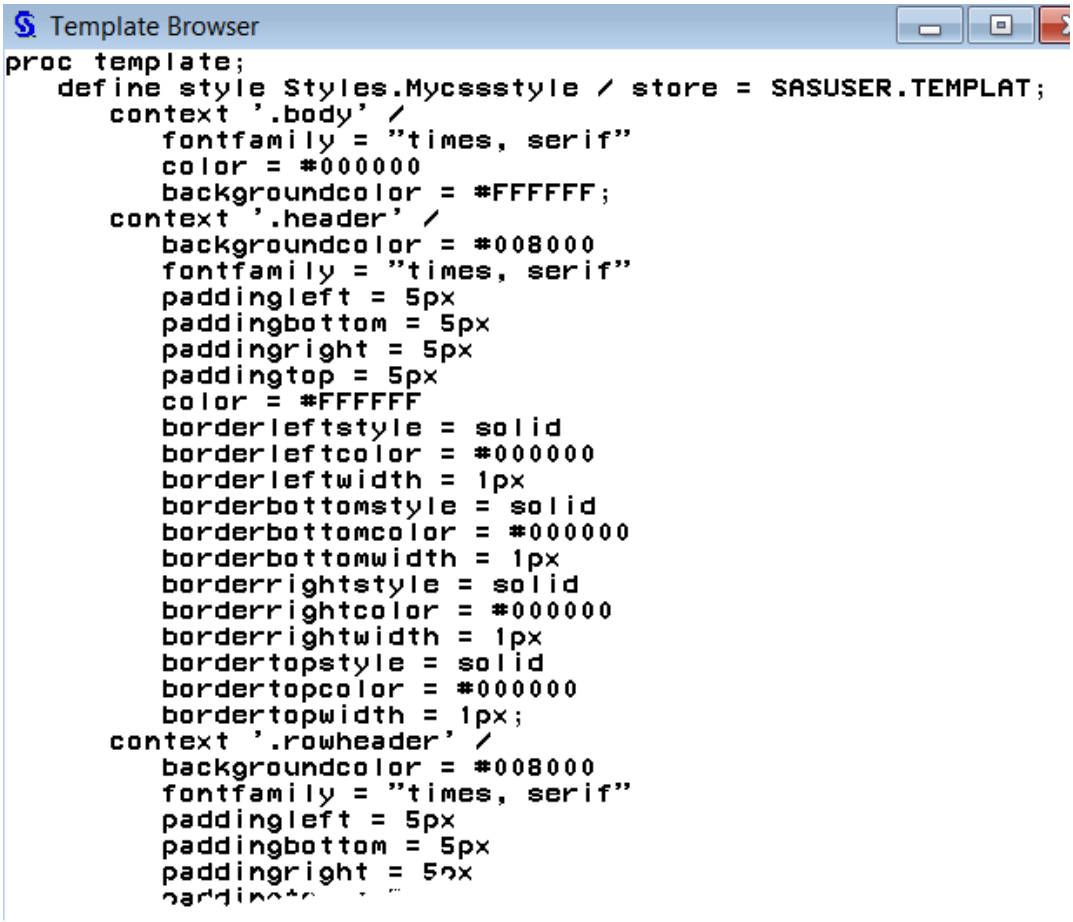
proc contents data=sashelp.class;
run;
```

Close the open destinations. The ODS _ALL_ CLOSE statement closes all open destinations and the files that are associated with them. If you do not close the destinations, then you will not be able to view the files. Specify the ODS HTML statement to return ODS to its default setup.

```
ods _all_ close;
ods html;
```

Output

Output 14.7 *MyCssStyle Style*



The yellow and green background colors, the white font color, the font size and border information all come from the Screen media block. The red font color comes from the CLASS statement. All other style information comes from the code outside of the media blocks. No information from the Print media block is used.

Output 14.8 *MyCssStyle Style Applied to HTML Output*

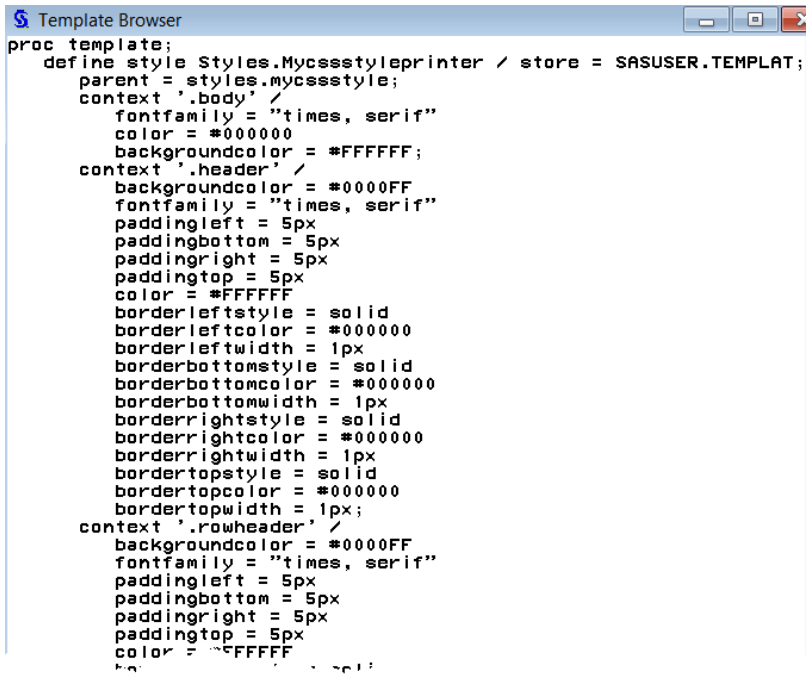
The CONTENTS Procedure

Data Set Name	SASHELP.CLASS	Observations	19
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	03/10/2013 19:14:52	Observation Length	40
Last Modified	03/10/2013 19:14:52	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	Student Data		
Data Representation	WINDOWS_32		
Encoding	us-ascii ASCII (ANSI)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	1632
Obs in First Data Page	19
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	C:\SASv9\sasgen\dev\mva-v940\sas_dvd\sio\dntnd\en\sashelp\class.sas7bdat
Release Created	9.0401B0
Host Created	NET_SRV

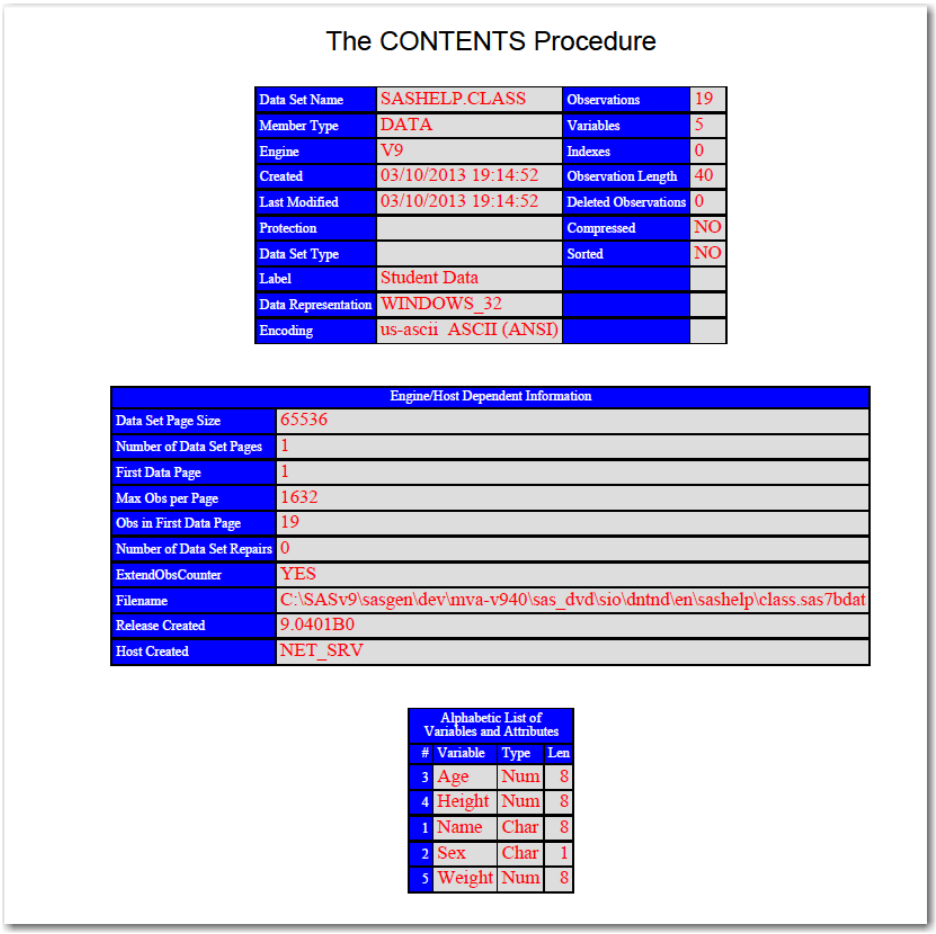
Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

Output 14.9 *MyCssStylePrinter Style*



The white font, small font size, cell padding, and the blue background color all come from the Print media block. All other style information comes from the code outside of the media blocks. No information from the Screen media block is used.

Output 14.10 *MyCssStylePrinter Style Applied to PDF Output*



The CONTENTS Procedure

Data Set Name	SASHELP.CLASS	Observations	19
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	03/10/2013 19:14:52	Observation Length	40
Last Modified	03/10/2013 19:14:52	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	Student Data		
Data Representation	WINDOWS_32		
Encoding	us-ascii ASCII (ANSI)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	1632
Obs in First Data Page	19
Number of Data Set Repairs	0
ExtendObsCounter	YES
Filename	C:\SASv9\sasgen\dev\mva-v940\sas_dvd\sio\dntnd\en\sashelp\class.sas7bdat
Release Created	9.0401B0
Host Created	NET_SRV

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

Example 6: Table Header and Footer Border Formatting

Features:

Border control style attributes
- BORDERBOTTOMCOLOR=
- BORDERBOTTOMSTYLE=
- BORDERBOTTOMWIDTH=
- BORDERTOPCOLOR=
- BORDERTOPSTYLE=
- BORDERTOPWIDTH=

DEFINE statement

DEFINE STYLE statement

EDIT statement

FOOTER statement

HEADER statement

PARENT= statement

PREFORMATTED= header attribute

STYLE statement

WIDTH= header attribute

Other ODS features:

ODS RTF statement
ODS SELECT statement

Data set: Stats and Stats2

Details

You can use the TableHeaderContainer and TableFooterContainer style elements along with the border control style attributes to change the borders of the regions surrounding the table header and footer.

Note: The TableHeaderContainer and TableFooterContainer style elements are valid only in the RTF destination.

Program

```
ods html close;
options nodate;

title  "TableHeaderContainer, TableFooterContainer, and Border Control
Style
     Attributes";
title2 "Allows Control of Borders Between the Header, Body, and Footer
of a
     Table";

proc template;
   define style  HeadersFootersBorders;
   parent=styles.rtf;

   style TableHeaderContainer from TableHeaderContainer /
      borderbottomwidth=12
      borderbottomcolor=blue
      borderbottomstyle=dotted;

   style TableFooterContainer from TableFooterContainer /
      bordertopwidth=6
      bordertopcolor=red
      bordertopstyle=double;

   style table from table /
      borderspacing=0 rules=groups frame=void;
   end;
run;

proc template;
   edit Base.Datasets.Members;
      header hd1;
      footer ft1;
      define hd1;
```

```
         preformatted=on;
         just=l;
         text"     Table Header with Leading and Trailing Blanks    ";
      end;
      define ft1;
         preformatted=on;
         just=l;
         text"     Table Footer with Leading and Trailing Blanks    ";
      end;
      edit name;
       define header myheader;
         just=l;
         preformatted=on;
         text "     My new header";
       end;
       header=myheader;
       width=memname_width width_max=memname_width_max;
       preformatted=on;
      end;
   end;
run;

ods rtf file="file-path/headerfooters.rtf" style=HeadersFootersBorders;
ods select members;
proc datasets lib=work;
run;
quit;

ods rtf close;
ods html;

proc template;
delete HeadersFootersBorders;
delete Base.Datasets.Members;
run;
```

Program Description

Close the HTML destination so that no HTML output is produced. The HTML destination is open by default. The ODS HTML CLOSE statement closes the HTML destination to conserve resources. If the destination were left open, then ODS would produce both HTML and PDF output.

```
ods html close;
options nodate;
```

Specify titles. The TITLE statements specify titles for the output.

```
title  "TableHeaderContainer, TableFooterContainer, and Border Control
Style
     Attributes";
title2 "Allows Control of Borders Between the Header, Body, and Footer
of a
     Table";
```

Create the new style HeadersFootersBorders. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new

style HeadersFootersBorders. The PARENT= statement specifies that the new style inherits all of its style elements and style attributes from the Styles.RTF style.

```
proc template;
   define style  HeadersFootersBorders;
   parent=styles.rtf;
```

Modify the TableHeaderContainer style element. The STYLE statement with the FROM option specified creates the style element TableHeaderContainer which inherits all of its style elements and style attributes from the instance of TableHeaderContainer in the Styles.RTF style. The BORDERBOTTOMWIDTH=, BORDERBOTTOMCOLOR=, and BORDERBOTTOMSTYLE= style attributes specify the width, color, and line style of the bottom border of the table header.

```
      style TableHeaderContainer from TableHeaderContainer /
         borderbottomwidth=12
         borderbottomcolor=blue
         borderbottomstyle=dotted;
```

Modify the TableFooterContainer style element. The STYLE statement with the FROM option specified creates the style element TableFooterContainer which inherits all of its style elements and style attributes from the instance of TableFooterContainer in the Styles.RTF style. The BORDERTOPWIDTH=, BORDERTOPCOLOR=, and BORDERTOPSTYLE= style attributes specify the width, color, and line style of the top border of the table footer.

```
      style TableFooterContainer from TableFooterContainer /
         bordertopwidth=6
         bordertopcolor=red
         bordertopstyle=double;
```

Modify the Table style element. The STYLE statement with the FROM option specified creates the style element Table which inherits all of its style elements and style attributes from the instance of Table in the Styles.RTF style. The BORDERSPACING=, RULES=, and FRAME= attributes modify the border spacing, rules, and frame of the table.

```
      style table from table /
         borderspacing=0 rules=groups frame=void;
      end;
   run;
```

Edit the Base.Datasets.Members table template. The EDIT statement, along with the table template DEFINE statements and attributes, modifies the Base.Datasets.Members table template.

For more information about creating and modifying table templates, see Chapter 15, “TEMPLATE Procedure,” on page 515.

```
   proc template;
      edit Base.Datasets.Members;
         header hd1;
         footer ft1;
         define hd1;
            preformatted=on;
            just=l;
            text"     Table Header with Leading and Trailing Blanks    ";
         end;
         define ft1;
            preformatted=on;
            just=l;
```

```
          text"     Table Footer with Leading and Trailing Blanks    ";
        end;
        edit name;
         define header myheader;
           just=l;
           preformatted=on;
           text "     My new header";
         end;
         header=myheader;
         width=memname_width width_max=memname_width_max;
         preformatted=on;
        end;
    end;
run;
```

Create the RTF file, select the output object and run PROC DATASETS. The ODS RTF statement specifies the file that will contain the RTF output. The STYLE= option specifies the style to apply to the output. The ODS SELECT statement selects the output object Members to be sent to the open destinations.

```
ods rtf file="file-path/headerfooters.rtf" style=HeadersFootersBorders;
ods select members;
proc datasets lib=work;
run;
quit;
```

Close the RTF destination and open the HTML destination. The ODS RTF CLOSE statement closes the RTF destination and the files that are associated with it. If you do not close the destination, then you will not be able to view the files. Specify the ODS HTML statement to return ODS to its default setup.

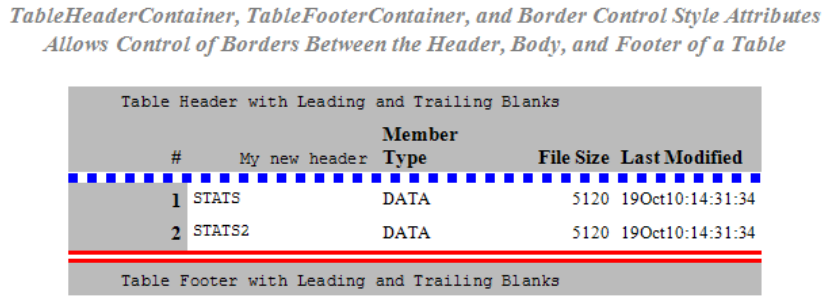
```
ods rtf close;
ods html;
```

Once you have created your output, you can delete the custom styles.

```
proc template;
delete HeadersFootersBorders;
delete Base.Datasets.Members;
run;
```

RTF Output

Output 14.11 *RTF Output with Custom Headers and Footers*



Example 7: Enhancing Titles and Footnotes in PDF Output

Features:

CLASS statement features
- COLOR= style attribute
- FONT_WEIGHT= style attribute
- SystemFooter style element
- User-defined style element FONTS
- User-defined style attributes

DEFINE STYLE statement

PARENT= statement

OPTIONS statement

ODS PDF statement

PROC PRINT

PROC TEMPLATE DELETE statement

Details

The text of titles and footnotes created by tabular (non-graphical) output is controlled using the user-defined style attribute TitleFont in the style element Fonts. In Styles.Pearl (the default style for PDF output), the source code shows that for

TitleFont, the font family is "<MTsans-serif>, Albany", the font size is 11pt, and the font weight is bold:

```
source styles.pearl;
define style Styles.Pearl;
   parent = styles.printer;
   class fonts /
      'TitleFont2' = ("<MTsans-serif>, Albany",10pt,bold)
      'TitleFont' = ("<MTsans-serif>, Albany",11pt,bold)
      'StrongFont' = ("<MTsans-serif>, Albany",8pt,bold)
      'EmphasisFont' = ("<MTsans-serif>, Albany",8pt,italic)
      'FixedEmphasisFont' = ("<MTmonospace>, Courier",7pt)
      'FixedStrongFont' = ("<MTmonospace>, Courier",7pt,bold)
      'FixedHeadingFont' = ("<MTmonospace>, Courier",7pt,bold)
      'BatchFixedFont' = ("SAS Monospace, <MTmonospace>, Courier",6pt)
      'FixedFont' = ("<MTmonospace>, Courier",7pt)
      'headingEmphasisFont' = ("<MTsans-serif>, Albany",9pt,bold
italic)
      'headingFont' = ("<MTsans-serif>, Albany",8pt,bold)
      'docFont' = ("<MTsans-serif>, Albany",8pt);
```

The following example uses PROC TEMPLATE to edit the style attribute TitleFont to make the following changes:

- font size 14pt
- font weight medium
- font style italic

For separate control of the footnotes, the style element SystemFooter can be modified using PROC TEMPLATE to make the footnotes green and medium weight.

Program

```
ods html close;

proc template;
   define style styles.MyPDFstyle;
   parent=styles.pearl;

   class fonts /
      'TitleFont2' = ("<MTsans-serif>, Albany",10pt,bold)
      'TitleFont' = ("<MTsans-serif>, Albany",14pt, medium italic)
      'StrongFont' = ("<MTsans-serif>, Albany",8pt,bold)
      'EmphasisFont' = ("<MTsans-serif>, Albany",8pt,italic)
      'FixedEmphasisFont' = ("<MTmonospace>, Courier",7pt)
      'FixedStrongFont' = ("<MTmonospace>, Courier",7pt,bold)
      'FixedHeadingFont' = ("<MTmonospace>, Courier",7pt,bold)
      'BatchFixedFont' = ("SAS Monospace, <MTmonospace>, Courier",6pt)
      'FixedFont' = ("<MTmonospace>, Courier",7pt)
      'headingEmphasisFont' = ("<MTsans-serif>, Albany",9pt,bold
italic)
      'headingFont' = ("<MTsans-serif>, Albany",8pt,bold)
      'docFont' = ("<MTsans-serif>, Albany",8pt);

    class SystemFooter /
        color = green
```

```
        font_weight=medium;
end;
run;

options nodate number;

ods pdf file="file-path/file.pdf" style=styles.MyPDFstyle;

proc print data=sashelp.cars(obs=5);
title "First 5 observations from SASHELP.CARS";
footnote "Created by SAS &sysver";
run;


ods pdf close;
ods html;

proc template;
delete Styles.MyPDFstyle;
end;
title;
```

Program Description

Close the HTML destination so that no HTML output is produced. The HTML destination is open by default. The ODS HTML CLOSE statement closes the HTML destination to conserve resources. If the destination were left open, then ODS would produce both HTML and PDF output.

```
ods html close;
```

Create a new style named Styles.MyPDFstyle that inherits from Styles.Pearl. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style called Styles.MyPDFstyle. The PARENT= attribute specifies Styles.Pearl. as the style from which Styles.MyPDFstyle inherits. All the style elements, attributes, and statements that are specified in the parent's style template (Pearl) are used in the child style template (MyPDFstyle) unless the child style template overrides them.

```
proc template;
   define style styles.MyPDFstyle;
   parent=styles.pearl;
```

Customize the title font by changing the attributes of the style element Fonts. The style element Fonts contains the style attribute that controls the appearance of titles. This CLASS statement adds the style element Fonts to the custom style Styles.MyPDFstyle. The Fonts style element also exists in the parent style (Styles.Pearl). The CLASS statement also changes the values of the user-defined style attribute TitleFont2. The change is that the title font will now be 14pt, medium weight, and italic.

```
   class fonts /
      'TitleFont2' = ("<MTsans-serif>, Albany",10pt,bold)
      'TitleFont' = ("<MTsans-serif>, Albany",14pt, medium italic)
      'StrongFont' = ("<MTsans-serif>, Albany",8pt,bold)
      'EmphasisFont' = ("<MTsans-serif>, Albany",8pt,italic)
      'FixedEmphasisFont' = ("<MTmonospace>, Courier",7pt)
      'FixedStrongFont' = ("<MTmonospace>, Courier",7pt,bold)
      'FixedHeadingFont' = ("<MTmonospace>, Courier",7pt,bold)
```

```
      'BatchFixedFont' = ("SAS Monospace, <MTmonospace>, Courier",6pt)
      'FixedFont' = ("<MTmonospace>, Courier",7pt)
      'headingEmphasisFont' = ("<MTsans-serif>, Albany",9pt,bold
italic)
      'headingFont' = ("<MTsans-serif>, Albany",8pt,bold)
      'docFont' = ("<MTsans-serif>, Albany",8pt);
```

Customize the footer by adding the SystemFooter style element. The SystemFooter style element controls the appearance of footers. It is not a default style element that is present in the parent style (Styles.Pearl). The CLASS statement adds SystemFooter, which contains the style attributes COLOR= and FONT_WEIGHT=, to Styles.MyPDFstyle. This statement results in the font of the footnote being changed to a medium, green font.

```
    class SystemFooter /
        color = green
        font_weight=medium;
end;
run;
```

Set the SAS system options.

```
options nodate number;
```

Create the PDF file and specify the custom style. The ODS PDF statement creates the PDF file. The STYLE= option specifies that the custom style template Styles.MyPDFstyle is applied to the PDF output.

```
ods pdf file="file-path/file.pdf" style=styles.MyPDFstyle;
```

Create the PRINT procedure output and specify a title and footnote. The appearance of the title and footnote reflects the changes made in the style MyPDFstyle.

```
proc print data=sashelp.cars(obs=5);
title "First 5 observations from SASHELP.CARS";
footnote "Created by SAS &sysver";
run;
```

Close the PDF destination and open the HTML destination. The ODS PDF CLOSE statement closes the PDF destination and all of the files that are associated with it. You must close the destinations before you can view the output with a browser or before you can send the output to a physical printer. The ODS HTML statement opens the HTML destination and returns ODS to its default setting.

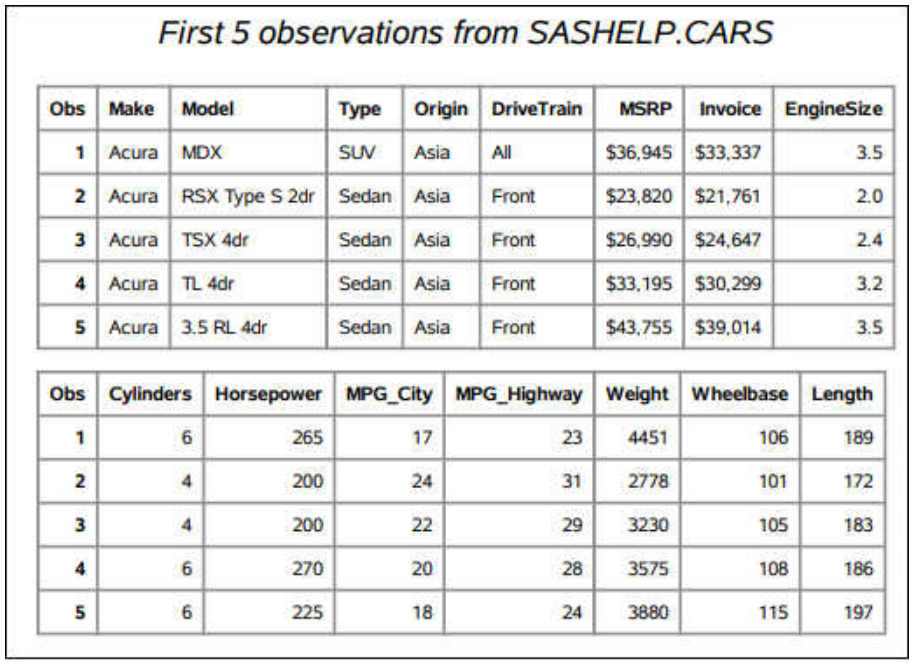
```
ods pdf close;
ods html;
```

Remove the customized style template. The DELETE statement removes the customized style that was created in this example. When you use the DELETE statement, ODS looks for the Styles.MyPDF style in Sasuser.Templat first. If it is there, it deletes it. If not, it searches Sashelp.Tmplmst.

```
proc template;
delete Styles.MyPDFstyle;
end;
title;
```

PDF Output

Output 14.12 *PDF Output with Custom Title and Footnote*



First 5 observations from SASHELP.CARS

Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5

Obs	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	6	265	17	23	4451	106	189
2	4	200	24	31	2778	101	172
3	4	200	22	29	3230	105	183
4	6	270	20	28	3575	108	186
5	6	225	18	24	3880	115	197

Created by SAS V.04.00

Example 8: Customizing Graphic and Tabular Titles in PDF Output

Features:

STYLE statement features
- COLOR= style attribute
- FROM option
- SystemTitle style element
- SystemTitle2 style element
- SystemTitle3 style element
- TitlesandFooters style element

DEFINE STYLE statement

PARENT= statement

OPTIONS statement

ODS PDF statement options
- NOGTITLE
- NOTOC

PROC PRINT

PROC SGPANEL

TITLE statement

Details

The style elements and attributes that control the appearance of titles in tabular output are in the Base.Template.Style template. In the BASE.TEMPLATE.STYLE definition, there is an inheritance defined for each title (and footnote) as shown by this code snippet:

```
style SystemTitle from TitlesAndFooters
   "Controls system title text.";
style SystemTitle2 from SystemTitle
   "Controls system title2 text";
style SystemTitle3 from SystemTitle2
   "Controls system title3 text";
```

When you use a style template to customize titles in PDF output, you must be careful if you have graphical output in your PDF file. By default, titles and footnotes are embedded in graphical output created by ODS graphics and SAS/Graph. The style definitions contain separate definitions for the titles and footnotes for these procedures.

The fonts controlling graphicical output are defined under the Graphfonts collection in the default PDF style Styles.Pearl, as shown in this code snippet:

```
  ..more style information...
class GraphFonts /
      'GraphAnnoFont' = ("<MTsans-serif>, Albany",10pt)
      'GraphTitle1Font' = ("<MTsans-serif>, Albany",14pt,bold)
      'GraphTitleFont' = ("<MTsans-serif>, Albany",11pt,bold)
      'GraphFootnoteFont' = ("<MTsans-serif>, Albany",10pt)
      'GraphLabelFont' = ("<MTsans-serif>, Albany",10pt)
      'GraphLabel2Font' = ("<MTsans-serif>, Albany",10pt)
      'GraphValueFont' = ("<MTsans-serif>, Albany",9pt)
      'GraphUnicodeFont' = ("<MTsans-serif-unicode>",9pt)
      'GraphDataFont' = ("<MTsans-serif>, Albany",7pt);
. .more style information...
```

This example illustrates the following points

- You can use the ODS PDF option NOGTITLE to specify that the titles for any graphical output are not embedded in the resulting image. This enables the titles to be under the control of Systemtitle and Systemtitle*n*, which are the style elements that control titles in tabular output. This is also true for footnotes in PDF output.
- Most SAS styles are written with a large degree of inheritance. There are very few stand-alone styles. After all of your output titles are controlled by Systemtitle and Systemtitl*n*, you can use inheritance to create a custom style that customizes the titles for your PDF output.

Program

```
ods html close;

proc template;
```

```
   define style Styles.CustomTitles;
   parent=styles.pearl;
   style SystemTitle from TitlesAndFooters
       "Controls system title text." / color=blue;
   style SystemTitle2 from SystemTitle
       "Controls system title2 text" / color=orange;
   style SystemTitle3 from SystemTitle2
       "Controls system title3 text" / color=purple;
   end;
run;

ods pdf file="file-path/PDFStyle.pdf" notoc style=Styles.CustomTitles
nogtitle;

title "Predicted and Actual Sales";
title2 "For Sofas";
title3 "By Region";

options nodate nonumber obs=50000;
proc sort data=sashelp.prdsale out=prdsale;
    by Country;
run;

proc sgpanel data=prdsale;
    where quarter=1;
    panelby product / novarname;
    vbar region / response=predict;
    vline region / response=actual lineattrs=GraphFit;
    colaxis fitpolicy=thin;
    rowaxis label='Sales';
run;
options nodate nonumber obs=8;

proc print data=prdsale;
 var product region actual predict;
run;


ods pdf close;
ods html;
```

Program Description

Close the HTML destination so that no HTML output is produced. The HTML destination is open by default. The ODS HTML CLOSE statement closes the HTML destination to conserve resources. If the destination were left open, then ODS would produce both HTML and PDF output.

```
ods html close;
```

Create a custom style template that modifies the appearance of titles. This PROC TEMPLATE step creates a new style named Styles.CustomTitles that inherits all of its style elements and attributes from Styles.Pearl. The STYLE statements use the COLOR= style attribute to change the color of the system options in tabular output.

```
proc template;
```

```
    define style Styles.CustomTitles;
    parent=styles.pearl;
    style SystemTitle from TitlesAndFooters
        "Controls system title text." / color=blue;
    style SystemTitle2 from SystemTitle
        "Controls system title2 text" / color=orange;
    style SystemTitle3 from SystemTitle2
        "Controls system title3 text" / color=purple;
    end;
run;
```

Open the PDF destination and specify the ODS PDF statement options. The NOTOC option specifies that no table of contents is created. The NOGTITLE option specifies that titles are not embedded in graphical output. This enables the titles to be controlled by the style elements specified in the previous STYLE statements.

```
ods pdf file="file-path/PDFStyle.pdf" notoc style=Styles.CustomTitles
nogtitle;
```

Specify the titles.

```
title "Predicted and Actual Sales";
title2 "For Sofas";
title3 "By Region";
```

Create the procedure output.

```
options nodate nonumber obs=50000;
proc sort data=sashelp.prdsale out=prdsale;
    by Country;
run;

proc sgpanel data=prdsale;
    where quarter=1;
    panelby product / novarname;
    vbar region / response=predict;
    vline region / response=actual lineattrs=GraphFit;
    colaxis fitpolicy=thin;
    rowaxis label='Sales';
run;
options nodate nonumber obs=8;

proc print data=prdsale;
 var product region actual predict;
run;
```

Close the PDF destination and open the HTML destination. The ODS PDF CLOSE statement closes the PDF destination and all of the files that are associated with it. You must close the destinations before you can view the output with a browser or before you can send the output to a physical printer. The ODS HTML statement opens the HTML destination and returns ODS to its default setting.

```
ods pdf close;
ods html;
```

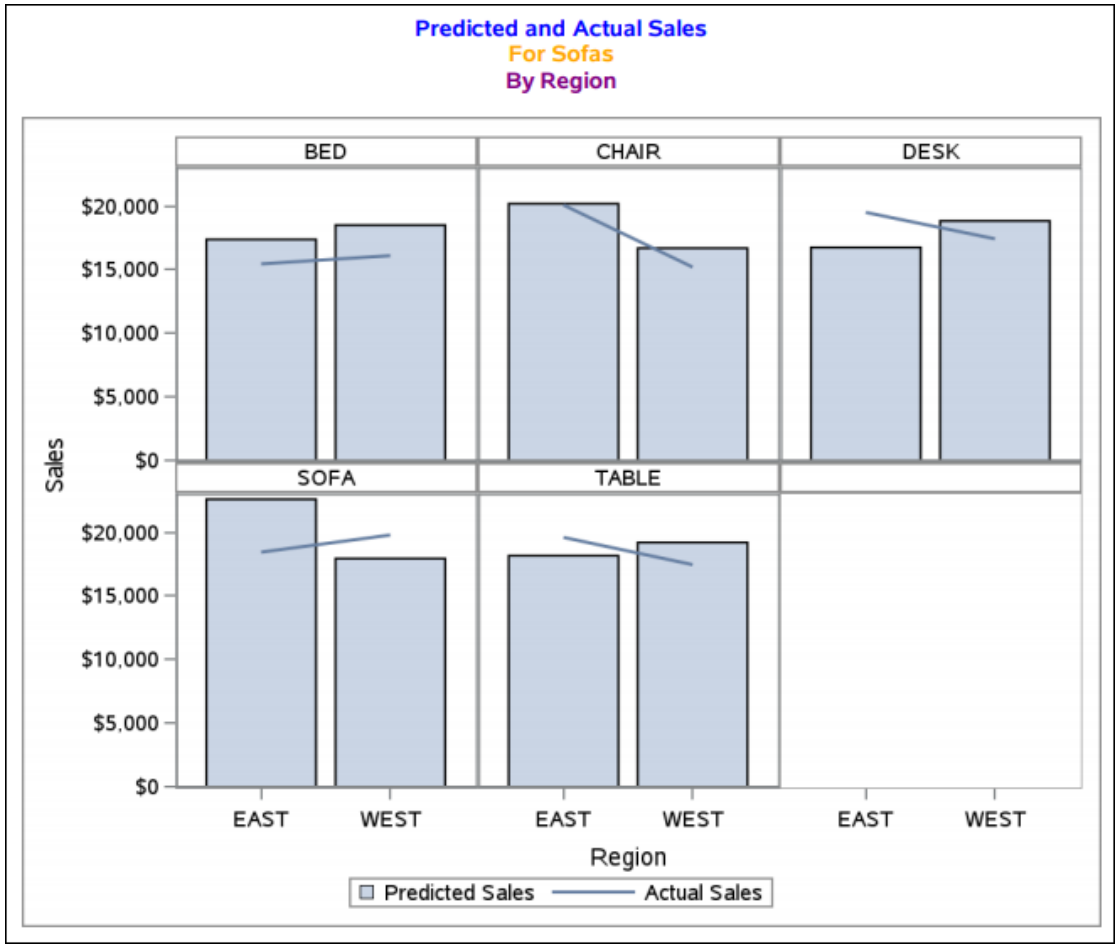
PDF Output

The following output shows that the custom style is applied to all of our output. If you omit the NOGTITLE option from the ODS PDF statement, then the titles are embedded within the graph and the custom style is not applied to them.

Output 14.13 *PDF with Customized Titles for Tabular and Graphical Output*

Predicted and Actual Sales
For Sofas
By Region

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00



Chapter 15

TEMPLATE Procedure: Creating Tabular Templates

Overview: TEMPLATE Procedure: Creating Tabular Templates ***515***
Using the TEMPLATE Procedure to Create or Customize Tabular Output 516
What You Can Do with Table Templates 517

Concepts: TEMPLATE Procedure: Creating Tabular Templates ***519***
Comparing the Edit of an Existing Table Template with Creating a New Table Template 519
Viewing the Contents of a Table Template 520

Syntax: TEMPLATE Procedure: Creating Tabular Templates ***520***
CELLSTYLE AS Statement 522
COLUMN Statement 525
COMPUTE AS Statement 526
DEFINE Statement 528
DEFINE COLUMN Statement 530
DEFINE FOOTER Statement 531
DEFINE HEADER Statement 532
DEFINE TABLE Statement 533
DYNAMIC Statement 535
EDIT Statement 536
END Statement 537
FOOTER Statement 537
HEADER Statement 538
MVAR Statement 539
NMVAR Statement 540
NOTES Statement 541
TEXT Statement 542
TEXT2 Statement 543
TEXT3 Statement 543
TRANSLATE INTO Statement 544

Usage: TEMPLATE Procedure: Creating Tabular Templates ***547***
Values in Table Columns and How They Are Justified 547
Formatting Values in Table Columns 548
Stacking Values for Two or More Variables 549

Examples: TEMPLATE Procedure: Creating Tabular Templates ***550***

Example 1: Editing a Table Template That a SAS Procedure Uses 550
Example 2: Comparing the EDIT Statement to the DEFINE TABLE Statement . . 553
Example 3: Creating a New Table Template 557
Example 4: Setting the Style Element for Cells Based on Their Values 565
Example 5: Setting the Style Element for a Specific Column, Row, and Cell 570
Example 6: Creating Top-Level Templates 577
Example 7: Table Header and Footer Border Formatting 580

Overview: TEMPLATE Procedure: Creating Tabular Templates

Using the TEMPLATE Procedure to Create or Customize Tabular Output

The TEMPLATE procedure enables you to customize the tabular appearance of your SAS output.

Tabular templates describe how tables should be constructed. This includes the content and placement of headers and footers, the content and placement of columns, and style overrides. All SAS procedures, except PROC PRINT, PROC REPORT, and PROC TABULATE, use tabular templates to describe how their tabular output should look. This means that you can change the structure of tables that are generated by SAS procedures by using tabular templates.

With the TEMPLATE procedure, you can create and modify tabular templates. Tabular templates include the following types:

- column templates
- header templates
- footer templates
- table templates

The Output Delivery System then uses these templates to produce customized tabular output for better data presentations and reports than what you get with the default SAS output. You can also create your own top-level tables using templates.

By default, ODS output is formatted according to the various definitions or templates that the procedure or DATA step specify. However, you can customize existing tabular output templates, or create your own new tabular output templates, by using the TEMPLATE procedure with these statements.

Table 15.1 PROC TEMPLATE Statements

Customization	Element Modified	Statement
Column presentation	Column template	"DEFINE COLUMN Statement" on page 530
Table footer	Footer template	"DEFINE FOOTER Statement" on page 531
Table header	Header template	"DEFINE HEADER Statement" on page 532
Single output object	Table template	"DEFINE TABLE Statement" on page 533
An existing template for a table, column, header, or footer	Table, column, header, footer	"EDIT Statement" on page 536

You can find additional tips and tricks in the SAS press book excerpt, ODS Techniques: Tips for Enhancing Your SAS Output, by Kevin Smith. This SAS press book is a cookbook-style collection of Kevin's top ODS tips and techniques to teach you how to bring your reports to a new level and inspire you to see ODS in a new light.

What You Can Do with Table Templates

Default Listing and RTF Display of an Output Object

By default, ODS uses the table template specified by the procedure or DATA step to create ODS output. For example, the following display shows the default RTF output of the Moments output object created by PROC UNIVARIATE.

Output 15.1 *RTF Output of Sales Statistics from PROC UNIVARIATE (Default Moments Table)*

Default Moments Table

The UNIVARIATE Procedure
Variable:
Quantity

Moments			
N	48	Sum Weights	48
Mean	20.5208333	Sum Observations	985
Std Deviation	14.4177633	Variance	207.871897
Skewness	3.6558946	Kurtosis	19.528114
Uncorrected SS	29983	Corrected SS	9769.97917
Coeff Variation	70.2591509	Std Error Mean	2.08102487

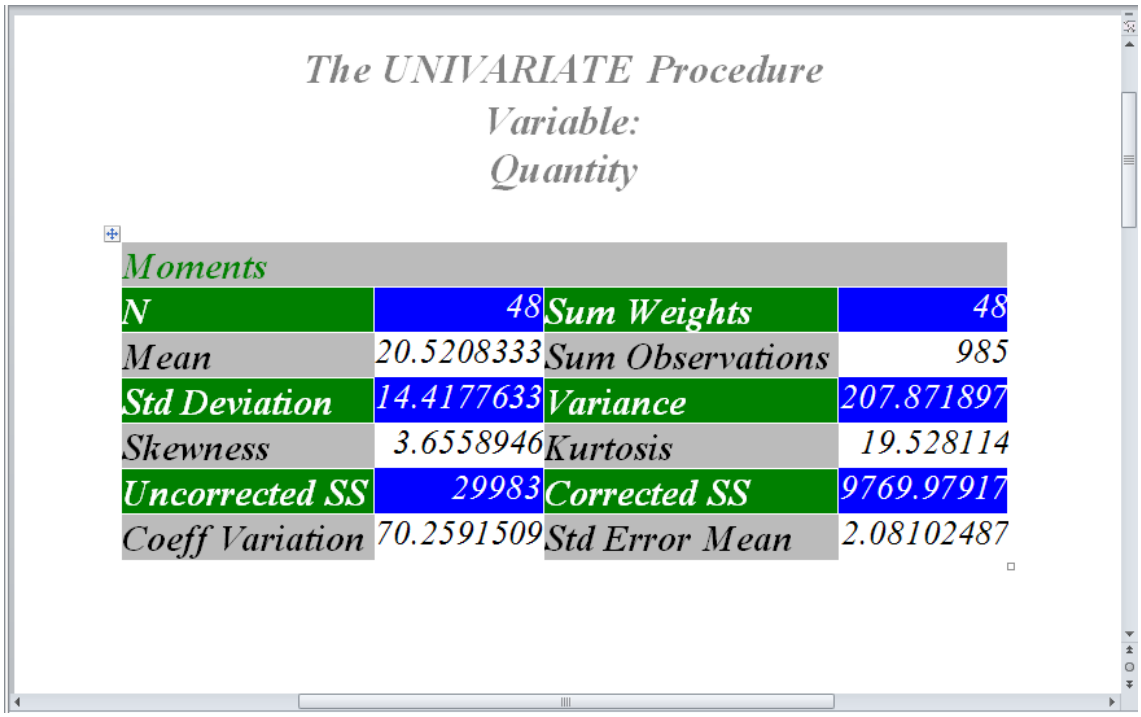
Customized Version of the RTF Display of an Output Object

With PROC TEMPLATE, you can change many of the table elements and obtain a customized format for the output objects. Here are some of the elements that you can change:

- the color and the font of the text of the first table header
- the justification of the first table header
- the setting of the table attributes UNDERLINE and OVERLINE
- the line spacing between the rows

Note: Not all table template changes affect all destinations.

Figure 15.1 *RTF Output of Sales Statistics from PROC UNIVARIATE (Customized Moments Table)*



The UNIVARIATE Procedure
Variable:
Quantity

Moments			
N	48	Sum Weights	48
Mean	20.5208333	Sum Observations	985
Std Deviation	14.4177633	Variance	207.871897
Skewness	3.6558946	Kurtosis	19.528114
Uncorrected SS	29983	Corrected SS	9769.97917
Coeff Variation	70.2591509	Std Error Mean	2.08102487

Concepts: TEMPLATE Procedure: Creating Tabular Templates

Comparing the Edit of an Existing Table Template with Creating a New Table Template

To change a table template without completely redefining it, use an EDIT statement. Using the EDIT statement keeps all of the templates and attributes that already exist in the table template, and changes only the templates or attributes specified in the EDIT statement. By default, the modified table template is stored in Sasuser.Templat with the same name as the table template specified in the EDIT statement.

To create a new table template, use the DEFINE TABLE statement. A table template cannot be a parent to itself because creating a table through inheritance causes an error, and then the template must be deleted. When you create a new table template, only the columns, headers, footers, and table attributes that you define exist in the new table template.

Note: Assuming the default ODS path, if you edit an existing table or define a new table with the same name as an existing table, then the table template is stored in the Sasuser.Templat item store. This table template is used, by default, unless you specify that the Sashelp.Tmplmst path is searched first. However, you can use the ODS PATH statement to store the template elsewhere and access it differently. See the "ODS PATH Statement" in *SAS Output Delivery System: User's Guide* for more information.

Viewing the Contents of a Table Template

To view the contents of a table template, use the TEMPLATE procedure.

The SOURCE statement writes the source code for the specified template to the SAS log. For example, if to view the source code for all the objects in Base SAS, submit this code.

```
proc template;
source base;
run;
```

Syntax: TEMPLATE Procedure: Creating Tabular Templates

PROC TEMPLATE;
EDIT *template-path-1* <AS *template-path-2*> < / STORE=*libref.template-store* > ;
statements-and-attributes
END;
DEFINE COLUMN *column-path* | **Base.Template.Column**
< / STORE=*libref.template-store*>;
statements-and-attributes
END;
DEFINE FOOTER *footer-path* | **Base.Template.Footer**
< / STORE=*libref.template-store*>;
statements-and-attributes
END;
DEFINE HEADER *template-name* | **Base.Template.Header**;
statements-and-attributes
END;
DEFINE TABLE *table-path* | **Base.Template.Table**
</ STORE=*libref.template-store*>;
statements-and-attributes

END;

Statement	Task	Example
CELLSTYLE AS	Set the style element of the cells in the table or column according to the values of the variables	Ex. 5, Ex. 6
COLUMN	Declare a symbol as a column in the table and specify the order of the columns	Ex. 2
COMPUTE AS	Compute values for a column that is not in the data component, or modify the values of a column that is in the data component	
DEFINE	Create a template inside a table template	Ex. 2, Ex. 3, Ex. 7
DEFINE COLUMN	Create a template for a column	Ex. 4, Ex. 5, Ex. 6
DEFINE FOOTER	Create a template for a table footer	Ex. 3
DEFINE HEADER	Create a template for a table header or a header inside a column template	Ex. 5, Ex. 6
DEFINE TABLE	Create a table template	Ex. 4, Ex. 5, Ex. 6
DYNAMIC	Define a symbol that references a value that the data component supplies from the procedure or DATA step	Ex. 2
EDIT	Edit an existing template for a table, column, header, or footer	Ex. 1, Ex. 2, Ex. 7
END	End the table template, header template, column template, or footer template	
FOOTER	Declare a symbol as a footer in the table and specify the order of the footers	Ex. 7
HEADER	Declare a symbol as a header in the table and specify the order of the headers	Ex. 3, Ex. 7
MVAR	Define a symbol that references a macro variable, the value of which is treated as a string	Ex. 3
NMVAR	Define a symbol that references a macro variable, the value of which is treated as a number	
NOTES	Provide information about the table, header, column, or footer	Ex. 2
TEXT	Specify the text of a header, footer, or the label of a variable in an output data set	Ex. 5

Statement	Task	Example
TEXT2	Provide an alternative header or footer to use in the LISTING output if the header or footer that is provided by the TEXT statement is too long	
TEXT3	Provide an alternative header or footer to use in the LISTING output if the header or footer that is provided by the TEXT2 statement is too long	
TRANSLATE INTO	Translate the specified numeric values to other values	

CELLSTYLE AS Statement

For tables, sets the style element of the cells in the table or column according to the values of the variables. For text, sets the style attributes of the list items or paragraphs. Use this statement to set the presentation characteristics (such as foreground color and font face) of individual cells or text.

Restriction: The CELLSTYLE AS statement can be used only within a column template, an ODS list, an ODS textblock, or a table template.

Syntax

CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*]> <, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*] >>;

Required Arguments

expression
is an expression that is evaluated for each list item, paragraph, or table cell.

If *expression* resolves to TRUE (a nonzero value), the style element that is specified is used for the current cell. If *expression* is FALSE, the next expression in the statement is evaluated. Thus, you can string multiple expressions together to format cells conditionally.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands.

An operator is a symbol that requests a string, a comparison, logical operation, or arithmetic calculation. An operand is one of the following:

constant
is a fixed value such as the name of a column or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in-variable
is a special type of WHERE expression operand that helps you find common values in table or column templates.

Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

DATANAME
is a data column name.

DATATYPE
is the data type of the column variable. The data type is either numeric ("num") or character ("char").

Example The following CELLSTYLE AS statement specifies that numeric column variables have a red font color and character column variables have a blue font color:

```
cellstyle   _datatype_ = "num" as {color=red},
            _datatype_ = "char" as {color=blue};
```

LABEL
is a column label.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

ROW
is a row number. Row numbering begins with 1.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

STYLE
is a style element name.

See For a table of style element names, see Chapter 20, "Style Elements," on page 777.

Example "Example 6: Creating Top-Level Templates" on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example "Example 6: Creating Top-Level Templates" on page 577

comparison-operator
compares a variable with a value or with another variable.

The following table lists the comparison operators:

Table 15.2 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one or more from a list of values

Tip Using an expression of 1 as the last expression in the CELLSTYLE AS statement sets the style element for any cells that did not meet an earlier condition. For a table of style element names, see Chapter 20, "Style Elements," on page 777.

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

style-attribute-specification
describes a style attribute to set.

Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

For information about the style attributes that you can set in a table template, see "About Style Attributes" on page 465.

Optional Argument

style-element-name
is the name of a style element that is part of a style that is registered with the Output Delivery System.

SAS provides some styles. You can create customized styles and style elements with PROC TEMPLATE by using the "DEFINE STYLE Statement" on page 454. For a table of style element names, see Chapter 20, "Style Elements," on page 777.

The following style elements are frequently used with the CELLSTYLE AS statement:

- Data
- DataFixed
- DataEmpty
- DataEmphasis
- DataEmphasisFixed
- DataStrong
- DataStrongFixed
- ListItem
- ListItem2
- Paragraph

The style element provides the basis for displaying the cell. Additional style attributes modify the display.

Default Data

See Chapter 14, "TEMPLATE Procedure," on page 433

For a table of style element names, see Chapter 20, "Style Elements," on page 777.

COLUMN Statement

Declares a symbol as a column in the table and specifies the order of the columns.

Restriction: The COLUMN statement can be used only within a table template.

Examples: "Example 3: Creating a New Table Template " on page 557

"Example 2: Defining Variables with the COLUMN Statement" on page 271

Syntax

COLUMN *column(s)*;

Required Argument

column

is one or more columns. If the column is defined outside the current table template, reference it by its path in the template store. Columns in the template

are laid out from left to right in the same order that they are specified in the COLUMN statement.

Defaults If you omit a COLUMN statement, ODS makes a column for each column template (DEFINE COLUMN statement), and places the columns in the same order that the column templates have in the table template.

If you use a COLUMN statement but omit a DEFINE COLUMN statement for any of the columns, ODS uses a default column template that is based on the type of data in the column.

Interaction If you specify the column attribute PRINT=OFF, then the value of a column is turned off if the column is part of a stacked column. If all columns in a stacked column have PRINT=OFF set, then the entire column is removed from the table.

Tip Use a list of variable names, such as DAY1–DAY10, to specify multiple variables.

See “Stacking Values for Two or More Variables ” on page 549

COMPUTE AS Statement

Computes values for a column that is not in the data component, or modifies the values of a column that is in the data component.

Restriction: The COMPUTE AS statement can be used only within a column template.

Syntax

COMPUTE AS *expression*;

Required Argument

expression
is an expression that assigns a value to each table cell in the column.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical sequence of operators and operands. An operator is a symbol that requests a comparison, a logical operation, or an arithmetic calculation. An operand is one of the following:

constant
is a fixed value, such as the name of a column, or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

To reference another column in a COMPUTE AS statement, use the name of the column. In addition, if the column has values in the data component, you can reference the column itself in the expression.

For example, this DEFINE COLUMN block defines a column that contains the square root of the value in the column called Source:

```
define column sqroot;
   compute as sqrt(source);
   header="Square Root";
   format=6.4;
end;
```

function
specifies a SAS function. For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in variable
is a special type of WHERE expression operand that helps you find common values in column templates. Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

DATANAME
is a data-column name.

LABEL
is a column label.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

ROW
is a row number. Row numbering begins with 1.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

STYLE
is a style-element name.

Example "Example 6: Creating Top-Level Templates" on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example "Example 6: Creating Top-Level Templates" on page 577

comparison-operator
compares a variable with a value or another variable.

Table 15.3 Comparison Operators

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one from a list of values

Tip The COMPUTE AS statement can alter values in an output object. None of the templates that SAS provides modifies any values. To determine whether a template was provided by SAS, use the "ODS VERIFY Statement" in *SAS Output Delivery System: User's Guide*. If the template is not from SAS, the ODS VERIFY statement returns a warning when it runs the SAS program that uses the template. If you receive such a warning, use the SOURCE statement to look at the template and determine whether the COMPUTE AS statement alters values. (See "SOURCE Statement" on page 353.)

See For more information about SAS expressions and WHERE statement processing, see *SAS Programmer's Guide: Essentials*.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

DEFINE Statement

Creates a template inside of a table template.

Restriction: The DEFINE statement can be used only inside a table template.

See: DEFINE COLUMN Statement on page 530
DEFINE FOOTER Statement on page 531
DEFINE HEADER Statement on page 532

Example: "Example 2: Defining Variables with the COLUMN Statement" on page 271

Syntax

DEFINE <COLUMN | FOOTER | HEADER> *template-name* < / NOLIST>;
statements-and-attributes;
END;

Required Argument

template-name
specifies the name of the new template.

Restriction *template-name* must be a single-level name.

Tip To reference the template that you are creating from another template, create it outside of the table template.

Optional Arguments

COLUMN
specifies to create a column template. For statements and attributes that can be used within a column template, see “DEFINE COLUMN Statement” on page 530.

FOOTER
specifies to create a footer template. For statements and attributes that can be used within a footer template, see “DEFINE FOOTER Statement” on page 531.

HEADER
specifies to create a header template. For statements and attributes that can be used within a header template, see “DEFINE HEADER Statement” on page 532.

Options to Use after a Slash Delimiter

NOLIST
preserves the *template-type* when inheriting it from another table template.

Tip If you specify an existing *template-name* without using the NOLIST option, then the template is overwritten.

Details

The template type, HEADER, FOOTER, or COLUMN, is optional only if you specify an existing template name. For example, if a column named Make exists in the template, the following DEFINE statement is valid:

```
define make;
```

For a complete example, see “Example 2: Defining Variables with the COLUMN Statement” on page 271.

DEFINE COLUMN Statement

Creates a template for a column.

Requirement: An END statement must be the last statement in the template.

Interaction: A column template can include one or more header templates.

See: "DEFINE HEADER Statement" on page 532

Examples: "Example 4: Setting the Style Element for Cells Based on Their Values" on page 565
"Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570
"Example 6: Creating Top-Level Templates" on page 577

Syntax

DEFINE COLUMN *column-path* | Base.Template.Column
< / STORE=*libref.template-store*>;
<*column-attribute-1* < *column-attribute-2*>…>;
CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*] >
<, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*]>>;
COMPUTE AS *expression*;
DEFINE HEADER | **Base.Template.Header** *template-path*;
statements-and-attributes
END;
DYNAMIC *variable-1* <=*default-variable-1*>
<'*text-1*'>
<… *variable-n* <=*default-variable-n*><'*text-n*'>>;
MVAR *variable-1* <=*default-variable-1*><'*text-1*'>
<… *variable-n* <=*default-variable-n*><'*text-n*'>>;
NMVAR *variable-1* <=*default-variable-1*><'*text-1*'>
<… *variable-n* <=*default-variable-n*><'*text-n*'>>;
NOTES "*text*";
TRANSLATE *expression-1* **INTO** *expression-2*
<, *expression-n* INTO *expression-m*>;
END;

Required Arguments

column-path
specifies where to store the column template. A *column-path* consists of one or more names that are separated by periods. Each name represents a directory in a template store, which is a type of SAS file. PROC TEMPLATE writes the template to the first writable template store in the current path.

Restrictions If the template is nested inside another template, *template-path* must be a single-level name because the nested template is stored in the same location as the original template.

To reference the template that you are creating from another template, do not nest the template inside another one. For example, to reference a column template from multiple tables, do not define the column inside a table template.

Base.Template.Column
creates a top-level column template that is globally applied to all of your tabular output. After you create this template, you do not need to specify it explicitly in your SAS programs. It is automatically applied to all tabular output until you specifically remove the template from the item store.

Interaction The Base.Template.Column top-level template attributes are overridden by other tabular templates.

Example "Example 6: Creating Top-Level Templates" on page 577

Optional Argument

STORE=*libref.template-store*
specifies the template store in which to store the template. If the template store does not exist, it is created.

Restrictions If the template is nested inside another template, do not use the STORE= option for the nested template because it is stored in the same location as the original template.

The STORE= option does not become part of the template.

DEFINE FOOTER Statement

Creates a template for a table footer.

Requirement: An END statement must be the last statement in the template.

See: "DEFINE HEADER Statement" on page 532

Examples: "Example 3: Creating a New Table Template " on page 557
"Example 1: Creating a Stand-Alone Style" on page 466

Syntax

DEFINE FOOTER *footer-path* | **Base.Template.Footer**
< / STORE=*libref.template-store*>;
<*header/footer-attribute-1* < *header/footer-attribute-2*>…>;
DYNAMIC *variable-1* <=*default-variable-1*>

```
    <'text-1'>
        <… variable-n <=default-variable-n><'text-n'>>;
MVAR variable-1 <=default-variable-1><'text-1'>
    <… variable-n <=default-variable-n><'text-n'>>;
NMVAR variable-1 <=default-variable-1><'text-1'>
    <… variable-n <=default-variable-n><'text-n'>>;
NOTES "text";
TEXT footer-specification;
TEXT2 footer-specification;
TEXT3 footer-specification;
END;
```

Substatements and Attributes

The substatements in the DEFINE FOOTER statements and the footer attributes are the same as the substatements in the “DEFINE HEADER Statement” on page 532 and the “Header and Footer Attributes” on page 603.

DEFINE HEADER Statement

Creates a template for a table header or a header inside a column template.

Restriction: The DEFINE HEADER statement can be used only within a column template or a table template.

Requirement: An END statement must be the last statement in the template.

See: “Example 3: Creating a New Table Template ” on page 557

Syntax

```
DEFINE HEADER header-path | Base.Template.Header
    </ STORE=libref.template-store>;
    <header/footer-attribute-1 < header/footer-attribute-2>…>;
    DYNAMIC variable-1 <=default-variable-1>
        <'text-1'>
        <… variable-n <=default-variable-n><'text-n'>>;
    MVAR variable-1 <=default-variable-1><'text-1'>
        <… variable-n <=default-variable-n><'text-n'>>;
    NMVAR variable-1 <=default-variable-1><'text-1'>
        <… variable-n <=default-variable-n><'text-n'>>;
    NOTES "text";
    TEXT header-specification;
    TEXT2 header-specification;
    TEXT3 header-specification;
```

END;

Required Arguments

header-path
specifies where to store the header template. A *header-path* consists of one or more names, separated by periods. Each name represents a directory in a template store. (A template store is a type of SAS file.) PROC TEMPLATE writes the template to the first writable template store in the current path.

Restrictions If the template is nested inside another template, *header-path* must be a single-level name.

To reference the template that you are creating from another template, do not nest the template inside another template. For example, to reference a header template from multiple columns, do not define the header inside a column template.

Base.Template.Header | Base.Template.Footer
creates a top-level header template that is globally applied to all of your tabular output. After this template is created, you do not need to explicitly specify it in your SAS programs. It is automatically applied to all tabular output until you specifically remove it from the item store.

Interaction The Base.Template.Header or Base.Template.Footer top-level template attributes are overridden by other tabular templates.

Example "Example 6: Creating Top-Level Templates" on page 577

Optional Argument

STORE=*libref.template-store*
specifies the template store in which to store the template. If the template store does not exist, it is created.

Restrictions If the template is nested inside another template, do not use the STORE= option for the nested template because it is stored where the original template is stored.

The STORE= option does not become part of the template.

DEFINE TABLE Statement

Creates a table template.

Requirement: An END statement must be the last statement in the template.

Interaction: A table template can contain one or more column, header, or footer templates.

Examples: "Example 3: Creating a New Table Template " on page 557
"Example 4: Setting the Style Element for Cells Based on Their Values" on page 565

Syntax

DEFINE TABLE *table-path* | Base.Template.Table
</ STORE=*libref.template-store*>;
<*table-attribute-1* < *table-attribute-2*>…>;
CELLSTYLE *expression-1* **AS** <*style-element-name*><[*style-attribute-specification(s)*] >
<, *expression-n* AS <*style-element-name*><[*style-attribute-specification(s)*]>>;
COLUMN *column(s)*;
DEFINE *template-type template-name* </ *options*>;
statements-and-attributes
END;
DYNAMIC *variable-1* <=*default-variable-1*>
<'*text-1*'> <… *variable-n* <=*default-variable-n*><'*text-n*'>>;
FOOTER *footer-name(s)*;
HEADER *header-name(s)*;
MVAR *variable-1* <=*default-variable-1*><'*text-1*'>
<… *variable-n* <=*default-variable-n*><'*text-n*'>>;
NMVAR *variable-1* <=*default-variable-1*><'*text-1*'>
<… *variable-n* <=*default-variable-n*><'*text-n*'>>;
NOTES "*text*";
TRANSLATE *expression-1* **INTO** *expression-2* < , *expression-n* INTO *expression-m*;>
END;

Required Arguments

table-path
specifies where to store the table template. A *table-path* consists of one or more names that are separated by periods. Each name represents a directory in a template store, which is a type of SAS file. PROC TEMPLATE writes the template to the first writable template store in the current path.

Base.Template.Table
creates a top-level table template that is globally applied to all of your tabular output. Once this template is created, you do not need to explicitly specify it in your SAS programs. It is automatically applied to all tabular output until you specifically remove it from the item store.

Interaction The Base.Template.Table top-level template attributes are overridden by other tabular templates.

Tip The Base.Template.Table top-level template is most useful when used with the CELLSTYLE AS statements to create alternating colors in your tabular output.

Example “Example 6: Creating Top-Level Templates” on page 577

Optional Argument

STORE=*libref.template-store*
specifies the template store in which to store the template. If the template store does not exist, it is created.

Restriction The STORE= option does not become part of the template.

DYNAMIC Statement

Defines a symbol that references a value that the data component supplies from the procedure or DATA step.

Restriction: The DYNAMIC statement can be used only in the template of an ODS textblock, ODS list, table, column, header, footer, or statistical graph template. A dynamic variable that is defined in a template is available to that template and to all the templates that it contains.

Syntax

DYNAMIC *variable-name-1* <=*value–1*<'*text-1*'>>
< *variable-name-2* <=*value-2*><'*text-2*'…>>;

Required Argument

variable-name
names a variable that the data component supplies. ODS resolves the value of the variable when it binds the template and the data component.

Tip Dynamic variables are most useful to the authors of SAS procedures and to DATA step programmers.

Optional Arguments

value
sets the value of the variable.

text
is text that is placed in the template to explain the dynamic variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

EDIT Statement

Edits an existing template. The EDIT statement replaces the DEFINE statement in a template block when editing. You can use the EDIT statement in place of any DEFINE statement.

Restriction: If you edit a template that is a link, the link is broken and a separate template is created.

Requirement: An END statement must follow the EDIT statement and all of the editing instructions.

Interaction: In some cases, you can use an EDIT statement inside a set of editing instructions. When you edit a table template, you can also edit one or more column, header, or footer templates that are defined in the table. When you edit a column template, you can also edit one or more header templates that are defined for that column.

Example: "Example 1: Editing a Table Template That a SAS Procedure Uses" on page 550

Syntax

EDIT *template-path-1* <AS *template-path-2* > </ STORE=*libref.template-store*>;
template-statements;
END;

Required Argument

template-path-1
specifies a template to edit. *template-path-1* consists of one or more names that are separated by periods. Each name represents a directory in a template store, which is a type of SAS file.

Interaction The STORE= option specifies a particular template store to read from and write to.

Tip To determine the templates that a procedure or DATA step uses, submit the ODS TRACE ON statement before you run the SAS program. (See "ODS TRACE Statement" on page 70.)

Optional Arguments

AS *template-path-2*
specifies the location in which to store the edited template, where *template-path-2* consists of one or more names that are separated by periods. Each name represents a directory in a template store, which is a type of SAS file. By default, PROC TEMPLATE writes the edited template to the first writable template store in the current path.

Default If you omit AS *template-path-2*, PROC TEMPLATE writes the edited template to *template-path-1* in the first writable template store.

Restriction If the current EDIT statement is inside a set of editing instructions, do not use the AS *template-path-2* option.

STORE=*libref.template-store*
specifies the template store from which to read *template-path-1* and in which to store *template-path-2*.

template-statements
template-statements are any statements or attributes that are valid between the DEFINE statement and the END statement.

Editing an Existing Template

When you use the EDIT statement, the following occurs:

- By default, PROC TEMPLATE looks for *template-path-1* in the list of template stores that is defined by the PATH statement. (See "PATH Statement" on page 351.) It opens a copy of the first template path that it finds in a template store that has Read access.
- PROC TEMPLATE writes the modified template to the first template store in the current path with Update access. If you omit a second template path to write to, then PROC TEMPLATE uses *template-path-1*. Therefore, if the template store from which *template-path-1* is read has Update access, you are actually modifying the original template. Otherwise, the modified file is written to a template store to which you do have Update access.

  If you do specify a second template path, then PROC TEMPLATE writes the edited template to the specified path in the first template store to which you have Write access.

END Statement

Ends the table template, header template, column template, or footer template.

Syntax

END;

FOOTER Statement

Declares a symbol as a footer in the table and specifies the order of the footers.

Example: "Example 1: Creating a Customized Crosstabulation Table Template with No Legend" on page 397

Syntax

FOOTER *footer-specification(s)*;

Required Argument

footer-specification
is one or more footers. If the footer is defined outside the current table template, reference it by its path in the template store. Footers in the template are laid out from top to bottom in the same order that they are specified in the FOOTER statement.

Each *footer-specification* is one of the following:

"*string*"
specifies the text to use for the footer. If you specify a string, you do not need to specify a DEFINE FOOTER statement. However, you cannot specify any footer attributes except for a split character. If the SPLIT= attribute is not in effect and if the first character of the footer that you specify is neither a blank character nor an alphanumeric character, PROC TEMPLATE and PROC ODSTABLE treat it as the split character.

See "SPLIT= "*character*" | *variable*;" on page 611

footer-path
is the path of the footer template to use. A footer-path consists of one or more names, separated by periods. Each name represents a directory in a template store, which is a type of SAS file.

LABEL
uses the label of the output object as the footer. Each SAS procedure specifies a label for each output object that it creates. The DATA step uses the value of the OBJECTLABEL= option as the label of the output object. If OBJECTLABEL= is not specified, it uses the text of the first TITLE statement as the label.

Default If you omit a FOOTER statement, ODS makes a footer for each footer template (DEFINE FOOTER statement), and places the footers in the same order that the footer templates have in the table template.

HEADER Statement

Declares a symbol as a header in the table and specifies the order of the headers.

Syntax

HEADER *header-specification(s)*;

Required Argument

header-specification
is one or more headers. If the header is defined outside the current table template, reference it by its path in the template store. Headers in the template are laid out from top to bottom in the same order that they are specified in the HEADER statement.

Each *header-specification* is one of the following:

"*string*"
specifies the text to use for the header. If you specify a string, you do not need to use a DEFINE HEADER statement. However, you cannot specify any header attributes except for a split character. If the SPLIT= header attribute is not in effect and if the first character of the header that you specify is neither a blank character nor an alphanumeric character, PROC TEMPLATE and PROC ODSTABLE treat it as the split character.

See “SPLIT= "*character*" | *variable*;” on page 611

header-path
is the path of the header template to use. A header-path consists of one or more names, separated by periods. Each name represents a directory in a template store. (A template store is a type of SAS file.)

LABEL
uses the label of the output object as the header. Each SAS procedure specifies a label for each output object that it creates. The DATA step uses the value of the OBJECTLABEL= option as the label of the output object. If OBJECTLABEL= is not specified, it uses the text of the first TITLE statement as the label.

Default If you omit a HEADER statement, then ODS makes a header for each header template (DEFINE HEADER statement), and places the headers in the same order that the header templates have in the table template.

Example “Example 3: Creating a New Table Template ” on page 557

MVAR Statement

Defines a symbol that references a macro variable. ODS uses the value of the variable as a string. References to the macro variable are resolved when ODS binds the template and the data component to produce an output object.

Restriction: When replaying an ODS document with PROC DOCUMENT, values created by the MVAR statement must be re-created in the same session that is replaying the document.

Tip: You can use the MVAR statement in the template of an ODS list, ODS textblock, table, column, header, or footer. A macro variable that is defined in a template is available to that template and to all the templates that it contains.

See: “Example 3: Creating a New Table Template ” on page 557 and “Example 1: Creating a Stand-Alone Style” on page 466

Syntax

MVAR *variable-name-1* <='*value-1*' <'*text-1*'>>
< *variable-name-2* <='*value-2*'> <'*text-2*'…>>;

Required Argument

variable-name
names a macro variable to reference in the template. ODS uses the value of the macro variable as a string. ODS does not resolve the value of the macro variable until it binds the template and the data component.

Tip Declare macro variables this way in a template. For example, to use the automatic macro variable SYSDATE9 in a template, declare it in an MVAR statement and reference it as SYSDATE9, without an ampersand, in the PROC TEMPLATE or PROC ODSTABLE step. If you use the ampersand, the macro variable resolves when the template is compiled instead of when ODS binds the template to the data component.

Optional Arguments

value
sets the default variable value.

text
is text that is placed in the template to explain the macro variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

NMVAR Statement

Defines a symbol that references a macro variable. ODS converts the variable's value to a number (stored as a double) before using it. References to the macro variable are resolved when ODS binds the template and the data component to produce an output object.

Restriction: The NMVAR statement can be used only in the template of an ODS list, ODS textblock, table, column, header, or footer. A macro variable that is defined in a template is available to that template and to all the templates that it contains.

See: "Example 4: Setting the Style Element for Cells Based on Their Values" on page 565

Syntax

NMVAR *variable-name-1* <='*value–1*' <'*text-1*'>>
< *variable-name-2* <='*value-2*'> <'*text-2*'…>>;

Required Argument

variable-name
names a macro variable to reference in the template. ODS converts the variable's value to a number (stored as a double) before using it. ODS does not resolve the macro variable until it binds the template and the data component.

Tip Declare macro variables this way in a template. For example, to use a macro variable as a number, declare it in an NMVAR statement and reference it without an ampersand. If you use the ampersand, the macro variable resolves when the template is compiled instead of when ODS binds the template to the data component.

Optional Arguments

value
sets the value of the variable.

text
is text that is placed in the template to explain the macro variable's use. Text of this type becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

NOTES Statement

Provides information about the table, header, column, or footer.

Restriction: The NOTES statement can be used only in the template of a table, column, header, or footer.

Tip: The NOTES statement becomes part of the compiled template, which you can view with the SOURCE statement, whereas SAS comments do not.

See: “Example 4: Setting the Style Element for Cells Based on Their Values” on page 565

Example: “Example 1: Creating a Customized Crosstabulation Table Template with No Legend” on page 397

Syntax

NOTES '*text*';

Required Argument

text
provides information about the table.

TEXT Statement

Specifies the text of a header, footer, or the label of a variable in an output data set.

Restriction: The TEXT statement can be used only within a header or footer template.

See: "Example 3: Creating a New Table Template " on page 557

Syntax

TEXT *header/footer-specification(s)*;

Required Argument

header/footer-specification(s)

specifies the text of the header or footer. Each *header/footer-specification* is one of the following:

LABEL

uses the label of the object that the header applies to as the text of the header. For example, if the header or footer is for a column, _LABEL_ specifies the label for the variable that is associated with the column. If the header or footer is for a table, _LABEL_ specifies the label for the data set that is associated with the table.

text-specification(s)

specifies the text to use in the header or footer. Each *text-specification* is one of the following:

- a quoted string
- a variable, followed by an optional format. The variable is any variable that is declared in a DYNAMIC, MVAR, or NMVAR statement.

Note: If the first character in a quoted string is neither a blank character nor an alphanumeric character, and SPLIT is not in effect, the TEXT statement treats that character as the split character. See the discussion of the SPLIT= option in the "DEFINE HEADER Statement" on page 532.

Default If you omit a TEXT statement, the text of the header is the label of the object that the header applies to.

Tip If the quoted string is a blank and it is the only item in the header or footer specification, the header or footers a blank line.

Example "Example 3: Creating a New Table Template " on page 557

TEXT2 Statement

Provides an alternative header or footer to use in the LISTING output if the header or footer that is provided by the TEXT statement is too long.

Restriction: The TEXT2 statement can be used only within a header or footer template.

See: "TEXT Statement" on page 542

Syntax

TEXT2 *header/footer-specification(s)*

Required Argument

header/footer-specification(s)
specifies the text of the header or footer. Each *header/footer-specification* is one of the following:

LABEL
uses the label of the object that the header applies to as the text of the header. For example, if the header or footer is for a column, _LABEL_ specifies the label for the variable that is associated with the column. If the header or footer is for a table, _LABEL_ specifies the label for the data set that is associated with the table.

text-specification(s)
specifies the text to use in the header or footer. Each *text-specification* is one of the following:

- a quoted string
- a variable, followed by an optional format. The variable is any variable that is declared in a DYNAMIC, MVAR, or NMVAR statement.

Note: If the first character in a quoted string is neither a blank character nor an alphanumeric character, and SPLIT is not in effect, the TEXT statement treats that character as the split character. See the discussion of the SPLIT= option in the "DEFINE HEADER Statement" on page 532.

TEXT3 Statement

Provides an alternative header or footer to use in the LISTING output if the header or footer that is provided by the TEXT2 statement is too long.

Restriction: The TEXT3 statement can be used only within a header or footer template.

See: "TEXT Statement" on page 542

Syntax

TEXT3 *header/footer-specification(s)*

Required Argument

header/footer-specification(s)
specifies the text of the header or footer. Each *header/footer-specification* is one of the following:

LABEL
uses the label of the object that the header applies to as the text of the header. For example, if the header or footer is for a column, _LABEL_ specifies the label for the variable that is associated with the column. If the header or footer is for a table, _LABEL_ specifies the label for the data set that is associated with the table.

text-specification(s)
specifies the text to use in the header or footer. Each *text-specification* is one of the following:

- a quoted string
- a variable, followed by an optional format. The variable is any variable that is declared in a DYNAMIC, MVAR, or NMVAR statement.

Note: If the first character in a quoted string is neither a blank character nor an alphanumeric character, and SPLIT is not in effect, the TEXT statement treats that character as the split character. See the discussion of the SPLIT= option in the "DEFINE HEADER Statement" on page 532.

TRANSLATE INTO Statement

Translates the specified numeric values to other values.

Restrictions: The TRANSLATE INTO statement can be used only in a column template, an ODS list, an ODS textblock, or a table template.

The TRANSLATE INTO statement in a table template applies only to numeric variables. To translate the values of a character variable, use TRANSLATE INTO in the template of that column.

Syntax

TRANSLATE *expression-1* **INTO** *expression-2* <, *expression-n* INTO *expression-m*>;

Required Argument

expression
is an expression that is evaluated for each list item, paragraph, table, or column cell that contains a numeric variable.

If *expression-1* resolves to TRUE (a nonzero value), the translation that is specified is used for the current cell. If *expression-1* is FALSE (zero), the next expression in the statement is evaluated. Thus, you can string multiple expressions together to format cells conditionally.

expression has this form:

expression-1 <comparison-operator expression-n>

expression
is an arithmetic or logical expression that consists of a sequence of operators and operands. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation.

An operand is one of the following:

constant
is a fixed value such as the name of a column or symbols that are declared in a DYNAMIC, MVAR, or NMVAR statement in the current template.

function
specifies a SAS function.

For information about SAS functions, see *SAS Functions and CALL Routines: Reference*.

built-in-variable
is a special type of WHERE expression operand that helps you find common values in table or column templates.

Built-in variables are one or more of the following:

COLUMN
is a column number. Column numbering begins with 1.

Alias _COL_

Example “Example 5: Setting the Style Element for a Specific Column, Row, and Cell” on page 570

DATANAME
is a data column name.

DATATYPE
is the data type of the column variable. The data type is either numeric ("num") or character ("char").

LABEL
is a column label.

Example “Example 5: Setting the Style Element for a Specific Column, Row, and Cell” on page 570

ROW
is a row number. Row numbering begins with 1.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

STYLE
is a style element name.

Example "Example 6: Creating Top-Level Templates" on page 577

VAL
is the data value of a cell.

Tip Use _VAL_ to represent the value of the current column.

Example "Example 6: Creating Top-Level Templates" on page 577

comparison-operator
compares a variable with a value or with another variable.

The following table lists the comparison operators:

Table 15.4 *Comparison Operators*

Symbol	Mnemonic Equivalent	Definition
=	EQ	Equal to
^= or ~= or ¬= or <>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to
	IN	Equal to one or more from a list of values

Restriction You cannot reference the values of other columns in *expression-1*.

Tip Using an expression of 1 as the last expression in the TRANSLATE–INTO statement specifies a translation for any cells that did not meet an earlier condition.

See You can use any expression that can be used in the WHERE= data set option. For information about expressions that you can use in the WHERE data set option, see the section on WHERE-Expression Processing in *SAS Programmer's Guide: Essentials*.

Example "Example 5: Setting the Style Element for a Specific Column, Row, and Cell" on page 570

Usage: TEMPLATE Procedure: Creating Tabular Templates

Values in Table Columns and How They Are Justified

The process of justifying the values in columns in a LISTING output is determined by the format of the variable and the values of two attributes: JUST= and JUSTIFY=. It is a three-step process:

1. ODS puts the value into the format for the column. Character variables are left-justified within their format fields; numeric variables are right-justified.
2. ODS justifies the entire format field within the column width according to the value of the JUST= attribute for the column, or, if that attribute is not set, JUST= for the table. For example, if you right-justify the column, the format field is placed as far to the right as possible. However, the placement of the individual numbers and characters within the field does not change. Thus, decimal points remain aligned. If the column and the format field have the same width, then JUST= has no apparent effect because the format field occupies the entire column.
3. If you specify JUSTIFY=ON for the column or the table, ODS justifies the values within the column without regard to the format field. By default, JUSTIFY=OFF.

For example, consider this set of values:

```
123.45
234.5
.
987.654
```

If the values are formatted with a 6.2 format and displayed in a column with a width of 6, they appear this way, regardless of the value of JUST= (asterisks indicate the width of the column):

```
******
123.45
234.50
   .
987.65
```

If the width of the column increases to 8, then the value of JUST= does affect the placement of the values, because the format field has room to move within the column. Notice that the decimal points remain aligned but that the numbers shift in relation to the column width.

```
just=left                just=center              just=right

********                 ********                 ********
123.45                    123.45                    123.45
234.50                    234.50                    234.50
   .                         .                         .
987.65                    987.65                    987.65
```

Now, if you add JUSTIFY=ON, then the values are formatted within the column without regard to the format width. The results are as follows:

```
justify=on               justify=on               justify=on
just=left                just=center              just=right

********                 ********                 ********
123.45                    123.45                    123.45
234.50                    234.50                    234.50
.                           .                           .
987.65                    987.65                    987.65
```

All destinations except LISTING justify the values in columns as if JUSTIFY=ON.

Formatting Values in Table Columns

The process of formatting the values in columns in a LISTING output is determined by the format of the variable and the values of three options: FORMAT=, FORMAT_WIDTH=, and FORMAT_NDEC=. It is a four-step process:

1. If you omit a FORMAT= option, then the format that the data component provides is used. If the data component does not provide a format, then ODS uses one of the following:
   - best8. for integers
   - D12.3 for doubles
   - the length of the variable for character variables
2. If a format width is specified in the FORMAT= option, then it takes precedence over the FORMAT_WIDTH= and FORMAT_NDEC= options.
3. If you specify a decimal width with the FORMAT= and FORMAT_NDEC= options, then the format that is specified with the FORMAT= option is used.
4. If you specify a format width with the FORMAT= and FORMAT_WIDTH= options, then the format that is specified with FORMAT= option is used.

The formatting attributes of a column are determined by the data component or the column template. This table summarizes the behavior of the column formatting attributes based on which attributes the column template provides.

Table 15.5 Summary of Column Formatting Attributes

Specifications Provided by the Column Template	Result
Nothing	Format name, width, and number of decimal places are determined by the data component.
Format name	Format name and width are determined by the column template; number of decimal places is determined by the data component.
Format name and width	Format name and width are determined by the column template.
Format name, width, and number of decimal places	All three are determined by the column template.
Width	No name is specified; width is determined by the column template; number of decimal places is determined by the data component.
Number of decimal places	No name is specified; width is determined by the data component; number of decimal places is determined by the column template.

Stacking Values for Two or More Variables

To stack values for two or more variables in the same column, put parentheses around the stacked variables. In such a case, the column header for the first column inside the parentheses becomes the header for the column that contains all the variables inside parentheses. For example, this COLUMN statement produces a template with the following characteristics:

- The value of NAME is in the first column by itself.
- The values of CITY and STATE appear in the second column with CITY above STATE. The header for this column is the header that is associated with CITY.
- The values HOMEPHONE and WORKPHONE appear in the third column with HOMEPHONE above WORKPHONE. The header for this column is the header that is associated with HOMEPHONE.

```
column name (city state) (homephone workphone);
```

Use the asterisk (*) in the COLUMN statement to change the layout of stacking variables. An asterisk between groups of variables in parentheses stacks the first item in the first set of parentheses above the first item in the next set of parentheses, and so on, until the last group of parentheses is reached. Then, the second item in the first group is stacked above the second item in the second group, and so on. For example, this COLUMN statement produces a report with the following characteristics:

- The value of NAME is in the first column by itself.
- The values of CITY and HOMEPHONE appear in the second column with CITY above HOMEPHONE. The header for this column is the header that is associated with CITY.
- The values STATE and WORKPHONE appear in the third column with STATE above WORKPHONE. The header for this column is the header that is associated with STATE.

```
column name (city state) * (homephone workphone);
```

Examples: TEMPLATE Procedure: Creating Tabular Templates

Example 1: Editing a Table Template That a SAS Procedure Uses

Features: EDIT statement

Header attributes
JUST=
STYLE=

Other ODS features
ODS SELECT statement
DELETE statement

Data set: Exprev

Details

This example customizes the table template for the Moments output object from PROC UNIVARIATE.

The following tasks are demonstrated:

- creates and edits a copy of the default table template
- edits a header within the table template
- sets column attributes to enhance the appearance of the HTML output

For information about viewing a style, see Chapter 19, “Style Templates,” on page 723. For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

Select the output objects for the report. The ODS SELECT statement sends one output object, Moments, to the HTML destination, which is open by default.

To learn the names of the output objects, run the procedure with the ODS TRACE ON statement in effect. For more information see "ODS TRACE Statement" on page 70.

```
ods html5 file="moments.html";
ods select moments;
```

Compute the descriptive statistics for one variable. PROC UNIVARIATE computes the univariate statistics for one variable, Quantity. It uses the default table template, Base.Univariate.Moments from the template store Sashelp.Tmplmst.

```
proc univariate data=exprev mu0=3.5;
   var Quantity;
title "Default Moments Table";
run;
```

Specify the search path in order to locate the table template. The ODS PATH statement specifies which locations to search for definitions or templates that were created by PROC TEMPLATE, as well as the order in which to search for them. The statement is included to ensure that the example works correctly. However, if you have not changed the path, you do not need to include this statement because it specifies the default path.

```
ods path work.templat(update) sashelp.tmplmst(read);
```

Create a modified table template named Base.Univariate.Moments. The EDIT statement looks in the available template stores for a table template named Base.Univariate.Moments. By default, it first looks in Work.Templat, but it finds nothing. Next, it looks in Sashelp.Tmplmst, which contains the table templates that SAS provides. Because the EDIT statement can read this template, this is the one that it uses. The program does not specify a destination for the edited template, so PROC TEMPLATE writes to the first template store in the path that it can write to, which is Sasuser.Templat. It creates a table template of the same name as the original one in Sashelp.tmplmst.

To learn the name of the table template that a procedure uses, run the procedure with the ODS TRACE ON statement in effect. For more information, see "ODS TRACE Statement" on page 70.

```
proc template;
   edit base.univariate.moments;
```

Specify changes to the Moments output object for the HTML5 destination. The LABEL= table attribute specifies a label for the table. The CELLSTYLE AS statement sets the style of the data cell based on the value of the cell.

```
   label="Custom Moments";
   cellstyle 1 as data{color=orange fontstyle=italic};
```

Modify a table element. The following EDIT statement edits the table element Head within the table template. The STYLE= attribute alters the style element that produces the Head table element. The style element Header is defined in the default style. All other attributes that are included in Header remain in effect. The JUST= attribute left-justifies the text of the header.

```
   edit head;
         style=header{color=green fontstyle=italic};
         just=left;
```

Stop the editing of the table element and the table template. The first END statement ends the editing of the table element Head. The second END statement ends the editing of the table Base.Univariate.Moments.

```
      end;
   end;
run;
```

Select the output objects for the report. The ODS SELECT statement sends one output object, Moments, to the HTML5 destination, which is open by default.

```
ods select moments;
```

Compute the descriptive statistics for one variable. PROC UNIVARIATE computes the univariate statistics for one variable, Quantity. The actual results of the procedure step are the same in this case, but they are presented differently because the procedure uses the edited table template. It does so because when it looks for Base.Univariate.Moments, it looks in the first template store in the path, Sasuser.Templat. If you wanted to use the table template that is supplied by SAS, you would have to change the path with the ODS PATH statement.

See also: "ODS PATH Statement" in *SAS Output Delivery System: User's Guide*.

```
proc univariate data=exprev mu0=3.5;
   var Quantity;
title "Custom Moments Table";
run;
ods html5 close;
```

Remove the customized moments table template from Sasuser.Templat. The DELETE statement removes the customized moments table that was created in this example. When using the DELETE statement, ODS looks for **`base.univariate.moments`** in Sasuser.Templat first. If it is there, it deletes it. If not, it searches Sashelp.Tmplmst.

```
proc template;
delete base.univariate.moments;
end;
title;
```

Customized Moments Table

Custom Moments Table

The UNIVARIATE Procedure
Variable: Quantity

Moments			
N	48	Sum Weights	48
Mean	20.5208333	Sum Observations	985
Std Deviation	14.4177633	Variance	207.871897
Skewness	3.6558946	Kurtosis	19.528114
Uncorrected SS	29983	Corrected SS	9769.97917
Coeff Variation	70.2591509	Std Error Mean	2.08102487

Output 15.2 *Default Moments Table*

Default Moments Table

The UNIVARIATE Procedure
Variable: Quantity

Moments			
N	48	Sum Weights	48
Mean	20.5208333	Sum Observations	985
Std Deviation	14.4177633	Variance	207.871897
Skewness	3.6558946	Kurtosis	19.528114
Uncorrected SS	29983	Corrected SS	9769.97917
Coeff Variation	70.2591509	Std Error Mean	2.08102487

Example 2: Comparing the EDIT Statement to the DEFINE TABLE Statement

Features:
EDIT statement
COLUMN statement
DEFINE statement
    STYLE= attribute
NOTES statement
DYNAMIC statement
Other ODS features
    ODS PATH statement
    ODS HTML statement
    DELETE statement

Data set: Exprev

Details

This example compares the use of an EDIT statement with a DEFINE TABLE statement for the same table template. The first program uses the EDIT statement to change the Base.Summary table template. The foreground color of the NOBS column is changed to orange. The other templates and attributes of the Base.Summary table template remain the same. The second program uses the DEFINE TABLE statement to define a new table using the same name, Base.Summary. The NOBS column is the only column defined in the new table template. When the PROC SUMMARY step executes, only the NOBS column is printed. The only style attribute that formats the column is the `color=orange` attribute.

Program 1: Edit an Existing Table Template

```
ods path work.templat (update) sashelp.tmplmst (read);
proc template;
    edit Base.Summary;
    edit nobs;
    style={color=orange background=white};
end;

end;
run;

proc summary data=exprev print;
    class Sale_Type;
run;

proc template;
delete base.Summary;
run;
```

Program Description

Edit the existing table template Base.Summary. The ODS PATH statement specifies which item store to search first for the table template. The EDIT statement edits the table template Base.Summary. The modified table template Base.Summary is written to the Work.Templat item store.

```
ods path work.templat (update) sashelp.tmplmst (read);
proc template;
    edit Base.Summary;
    edit nobs;
    style={color=orange background=white};
end;

end;
run;

proc summary data=exprev print;
    class Sale_Type;
run;
```

Remove the customized summary table template from Work.Templat. The DELETE statement removes the customized summary table that was created in this example. When using the DELETE statement, ODS looks for **`base.summary`** in Sasuser.Templat and Work.Templat first. If it is there, it deletes it. If not, it searches Sashelp.Tmplmst.

```
proc template;
delete base.Summary;
run;
```

Output for Program 1: Edit an Existing Table Template

The column labeled Sale_Type remains in the output because Sale_Type is defined as a dynamic variable, which is passed to the original Base.Summary table template, and Sale_Type is specified as the CLASS variable. The attributes of the NOBS column are modified in the EDIT statement where the NOBS column is defined.

The modified Base.Summary table template changes the foreground color of the NOBS column to orange. The vertical alignment and heading of the NOBS column, and the other table attributes, are retained from the default table template and stay the same.

Output 15.3 *HTML Output Using an Edited Table Template for Base.Summary*

The SUMMARY Procedure

Sale_Type	N Obs
Catalog	12
In Store	18
Internet	18

Program 2: Define a New Table Template

```
ods path work.templat (update) sashelp.tmplmst (read);
proc template;
   define table Base.Summary;
      notes "Summary table for MEANS and SUMMARY";
      dynamic clmpct one_var_name one_var_label one_var;
      column class nobs id type ways (varname) (label) (min) (max)
(range)
          (n          ) (nmiss) (sumwgt) (sum) (mean) (uss) (css) (var)
(stddev) (cv)
          (         stderr) (t) (probt) (lclm) (uclm) (skew) (kurt)
(median) (mode) (q1)
          (q3) (qrange) (p1) (p5) (p10) (p25) (p50) (p75) (p90) (p95)
(p99);


      define nobs;
        style={color=orange backgroundcolor=white};

      end;

   end;
run;

proc summary data=exprev print;
class Sale_Type;
```

```
run;

proc template;
delete base.Summary;
run;
```

Program Description

Define the table Base.Summary. The ODS PATH statement specifies which item store to search first for the table template. The DEFINE TABLE statement creates a new table template Base.Summary. The new table template Base.Summary is written to the Work.Templat item store.

```
ods path work.templat (update) sashelp.tmplmst (read);
proc template;
   define table Base.Summary;
      notes "Summary table for MEANS and SUMMARY";
      dynamic clmpct one_var_name one_var_label one_var;
      column class nobs id type ways (varname) (label) (min) (max)
(range)
         (n         ) (nmiss) (sumwgt) (sum) (mean) (uss) (css) (var)
(stddev) (cv)
         (        stderr) (t) (probt) (lclm) (uclm) (skew) (kurt)
(median) (mode) (q1)
         (q3) (qrange) (p1) (p5) (p10) (p25) (p50) (p75) (p90) (p95)
(p99);


      define nobs;
        style={color=orange backgroundcolor=white};

      end;

   end;
run;

proc summary data=exprev print;
class Sale_Type;
run;
```

Remove the customized summary table template from Work.Templat. The DELETE statement removes the customized summary table that was created in this example. When using the DELETE statement, ODS looks for **`base.summary`** in Sasuser.Templat and Work.Templat first. If it is there, it deletes it. If not, it searches Sashelp.Tmplmst.

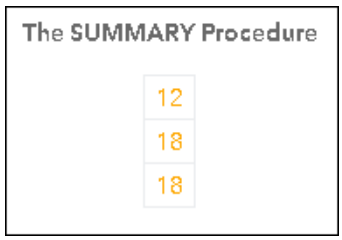
```
proc template;
delete base.Summary;
run;
```

Output for Program 2: Define a New Table Template

The column labeled Sale_Type is missing because it was not defined in the new table template Base.Summary. The new table template only defined the NOBS column with an orange foreground and no column headers.

The Base.Summary table template defines the foreground color of the NOBS column as orange. Because the vertical alignment and header of the NOBS column and the other table attributes are not defined, they are no longer part of the Base.Summary table template.

Output 15.4 *HTML Output That Uses the Table Template Base.Summary.*



Example 3: Creating a New Table Template

Features:

Table attributes
- DOUBLE_SPACE=
- OVERLINE=
- UNDERLINE=

DEFINE statement (for columns)
- GENERIC= attribute
- HEADER= attribute
- ID= attribute
- STYLE= attribute
- VJUST= attribute

DEFINE statement (for headers)
- TEXT statement
- STYLE= attribute
- SPACE= attribute

DEFINE FOOTER statement

HEADER statement

MVAR statement

Other ODS features
- FILE statement with ODS= option
- PUT statement with _ODS_ argument

Data set: Charity

Details

This example creates a custom table template for an output data set that PROC MEANS produces.

Program 1: Producing an Output Data Set with PROC MEANS

```
proc means data=Charity descendTypes charType noprint;
   class School Year;
   var moneyRaised;
   types () School year;
   output out=top3list sum= mean=
      idgroup ( max(moneyRaised) out[3](moneyRaised name school
year)= )
      / autoname;
   run;

proc print data=top3list noobs;
   title "Simple PROC PRINT of the Output Data Set";
run;
```

Program Description

Compute the descriptive statistics, and specify the options and subgroups for analysis. This PROC MEANS step analyzes the data for the one-way combination of the class variables and across all observations. It creates an output data set that includes variables for the total and average amount of money raised. The data set also includes new variables for the top three amounts of money raised, the names of the three students who raised the money, the years when the students raised the money, and the schools that the students attended.

```
proc means data=Charity descendTypes charType noprint;
   class School Year;
   var moneyRaised;
   types () School year;
   output out=top3list sum= mean=
      idgroup ( max(moneyRaised) out[3](moneyRaised name school
year)= )
      / autoname;
   run;
```

Print the report. This PROC PRINT step generates HTML output of the output data set that PROC MEANS created.

```
proc print data=top3list noobs;
   title "Simple PROC PRINT of the Output Data Set";
```

```
run;
```

Output for Program 1: Producing an Output Data Set with PROC MEANS

Output 15.5 *Default PROC PRINT Output*

Simple PROC PRINT of the Output Data Set

School	Year	_TYPE_	_FREQ_	moneyRaised_Sum	moneyRaised_Mean	moneyRaised_1	moneyRaised_2	moneyRaised_3	Name_1	Name_2	Name_3	School_1	School_2	School_3	Year_1	Year_2	Year_3
Kennedy	All	10	53	$1575.95	$29.73	$72.22	$52.63	$43.89	Luther	Thelma	Jenny	Kennedy	Kennedy	Kennedy	1994	1992	1992
Monroe	All	10	56	$1606.80	$28.69	$78.65	$65.44	$56.87	Willard	Cameron	L.T.	Monroe	Monroe	Monroe	1994	1993	1994
All	1992	01	31	$882.92	$28.48	$55.16	$53.76	$52.63	Tonya	Edward	Thelma	Monroe	Monroe	Kennedy	1992	1992	1992
All	1993	01	32	$907.92	$28.37	$65.44	$47.33	$42.23	Cameron	Myrtle	Bill	Monroe	Monroe	Kennedy	1993	1993	1993
All	1994	01	46	$1391.91	$30.26	$78.65	$72.22	$56.87	Willard	Luther	L.T.	Monroe	Kennedy	Monroe	1994	1994	1994
All	All	00	109	$3182.75	$29.20	$78.65	$72.22	$65.44	Willard	Luther	Cameron	Monroe	Kennedy	Monroe	1994	1994	1993

Program 2: Building a Custom Table Template for the TopN Report

```
proc template;
   define table means.topn;

   mvar first_year last_year sysdate9;

   column class sum mean (raised) (name) (school) (year);

    double_space=on;
    overline=on;
    underline=on;

   header table_header_1 table_header_2;


   define table_header_1;
      text "Top Three Fund Raisers";
      style=header{fontsize=6};
   end;

  define table_header_2;
      text "from " first_year " to " last_year;
      space=1;
  end;

   define footer table_footer;
      text "(report generated on " sysdate9 ")";
      split="*";
      style=header{fontsize=2};
   end;

   define class;
      generic=on;
      id=on;
      vjust=top;
      style=data;
   end;
```

```
   define sum;
      generic=on;
      header="Total Dollars Raised";
      vjust=top;
   end;

   define mean;
      generic=on;
      header="Average Dollars per Student";
      vjust=top;
   end;

   define raised;
      generic=on;
      header="Individual Dollars";
   end;

   define name;
      generic=on;
      header="Student";
   end;

   define school;
      generic=on;
      header="School";
   end;

   define year;
      generic=on;
      header="Year";
   end;

   end;
run;

data _null_;
   set top3list;

file print ods = (
          template="means.topn"

columns=(
         class=school(generic=on)
         class=year(generic=on)
         sum=moneyRaised_sum(generic=on)
         mean=moneyRaised_mean(generic=on)
         raised=moneyRaised_1(generic=on)
         raised=moneyRaised_2(generic=on)
         raised=moneyRaised_3(generic=on)
         name=name_1(generic=on)
         name=name_2(generic=on)
         name=name_3(generic=on)
         school=school_1(generic=on)
         school=school_2(generic=on)
         school=school_3(generic=on)
         year=year_1(generic=on)
         year=year_2(generic=on)
         year=year_3(generic=on)
```

```
          )
       );
put _ods_;
run;

proc template;
delete means.topn;
run;
```

Program Description

Create the table template Means.Topn The DEFINE statement creates the table template Means.Topn in the first template store in the path for which you have Write access. By default, this template store is Sasuser.Templat.

```
proc template;
   define table means.topn;
```

Specify the symbols that reference three macro variables. The MVAR statement defines three symbols that reference macro variables. ODS will use the values of these variables as strings. References to the macro variables are resolved when ODS binds the template and the data component to produce an output object. First_Year and Last_Year will contain the values of the first and last years for which there are data. Their values are assigned by the SYMPUT statements in the DATA step. SYSDATE9 is an automatic macro variable whose value is always available.

```
   mvar first_year last_year sysdate9;
```

Specify the column names and the order in which they appear in the report. The COLUMN statement declares these variables as columns in the table and specifies their order in the table. If a column name appears in parentheses, then PROC TEMPLATE stacks the values of all variables that use that column template one below the other in the output object. Variables are assigned a column template in the DATA step that appears later in the program.

```
   column class sum mean (raised) (name) (school) (year);
```

Specify three customized changes to the table template. These three table attributes affect the presentation of the output object in the LISTING output. They have no effect on its presentation in the HTML output. DOUBLE_SPACE= creates double spaces between the rows of the output object. OVERLINE= and UNDERLINE= draw a continuous line before the first row of the table and after the last row of the table.

```
    double_space=on;
    overline=on;
    underline=on;
```

Specify the two table headers and the order in which they appear in the report. The HEADER statement declares Table_Header_1 and Table_Header_2 as headers in the table and specifies the order in which the headers appear in the output object.

```
   header table_header_1 table_header_2;
```

Create the table element Table_Header_1. The DEFINE statement and its substatement and attribute define Table_Header_1. The TEXT statement specifies the text of the header. The STYLE= attribute alters the style element that displays

the table header. The style element Header is defined in the default style, Styles.HTMLBlue. In this case, the STYLE= attribute specifies a large font size. All other attributes that are included in Header remain in effect. This attribute affects only the HTML output. The END statement ends the header template.

```
define table_header_1;
   text "Top Three Fund Raisers";
   style=header{fontsize=6};
end;
```

Create the table element Table_Header_2. The DEFINE statement and its substatement and attribute define Table_Header_2. The TEXT statement uses text and the macro variables First_Year and Last_Year to specify the contents of the header. When ODS binds the data component to the table template (in the DATA step that follows), it will resolve the values of the macro variables First_Year and Last_Year. The table template itself contains references to the macro variables. The SPACE= attribute inserts a blank line after the header (in the LISTING output only). The END statement ends the header template.

```
define table_header_2;
    text "from " first_year " to " last_year;
    space=1;
end;
```

Create the table element Table_Footer. The DEFINE statement and its substatement and attribute define Table_Footer. The FOOTER argument declares Table_Footer as a footer. (Compare this approach with the creation of the headers. You could use a FOOTER statement instead of the FOOTER argument in the DEFINE statement.) The TEXT statement specifies the text of the footer. When ODS binds the data component to the table template (in the DATA step that follows), it will resolve the value of the macro variable SYSDATE9. The table template itself contains a reference to the macro variable. The SPLIT= attribute specifies the asterisk as the split character. This prevents the header from splitting at the open parenthesis. If no split character is specified, then ODS interprets the nonalphabetic, leading character as the split character. Alternatively, place a space character before the open parenthesis. The STYLE= attribute alters the style element that displays the table footer. The style element Header is defined in the default style, Styles.Default. In this case, the STYLE= attribute specifies a small font size. All other attributes that are included in Footer remain in effect. This attribute affects only the HTML output. The END statement ends the footer template.

```
define footer table_footer;
   text "(report generated on " sysdate9 ")";
   split="*";
   style=header{fontsize=2};
end;
```

Create the column template Class. The DEFINE statement and its attributes create the column template Class. (The COLUMN statement earlier in the program declared Class as a column.) GENERIC= specifies that multiple variables can use the same column template. GENERIC= is not specific to a destination.ID= specifies that this column should be repeated on every data panel if the report uses multiple data panels. ID= affects only the LISTING output. VJUST= specifies that the text appear at the top of the HTML table cell that it is in. VJUST= affects only the HTML output. STYLE= specifies that the column uses the DATA table element. This table element is defined in the default style, which is the style that is being used. STYLE= affects only the HTML output. The END statement ends the template. Notice that, unlike subsequent column templates, this column template does not include a header. This is because the same header is not appropriate for all the variables that

use this column template. Because there is no header specified here or in the FILE statement, the header comes from the label that was assigned to the variable in the DATA step.

```
define class;
   generic=on;
   id=on;
   vjust=top;
   style=data;
end;
```

Create six additional columns. Each of these DEFINE statements and its attributes creates a column template. GENERIC= specifies that multiple variables can use a column template (although in the case of Sum and Mean, only one variable uses the template). HEADER= specifies the text for the column header. VJUST= specifies that the text appear at the top of the HTML table cell that it is in. The END statement ends the template.

```
define sum;
   generic=on;
   header="Total Dollars Raised";
   vjust=top;
end;

define mean;
   generic=on;
   header="Average Dollars per Student";
   vjust=top;
end;

define raised;
   generic=on;
   header="Individual Dollars";
end;

define name;
   generic=on;
   header="Student";
end;

define school;
   generic=on;
   header="School";
end;

define year;
   generic=on;
   header="Year";
end;
```

End the table template. This END statement ends the table template. The RUN statement ends the PROC TEMPLATE step.

```
   end;
run;
```

Create the data component. This DATA step does not create a data set. Instead, it creates a data component and, eventually, an output object. The SET statement reads the data set TOP3LIST that was created with PROC MEANS.

```
data _null_;
   set top3list;
```

Route the DATA step results to ODS and use the Means.Topn table template. The combination of the fileref PRINT and the ODS option in the FILE statement routes the results of the DATA step to ODS. The TEMPLATE= suboption tells ODS to use the table template named Means.Topn, which was previously created with PROC TEMPLATE.

```
file print ods = (
          template="means.topn"
```

Specify the column template to use for each variable. The COLUMNS= suboption places DATA step variables into columns that are defined in the table template. For example, the first *column-specification* specifies that the first column of the output object contains the values of the variable SCHOOL and that it uses the column template named Class. GENERIC= must be set to ON in both the table template and each column assignment in order for multiple variables to use the same column template.

```
columns=(
         class=school(generic=on)
         class=year(generic=on)
         sum=moneyRaised_sum(generic=on)
         mean=moneyRaised_mean(generic=on)
         raised=moneyRaised_1(generic=on)
         raised=moneyRaised_2(generic=on)
         raised=moneyRaised_3(generic=on)
         name=name_1(generic=on)
         name=name_2(generic=on)
         name=name_3(generic=on)
         school=school_1(generic=on)
         school=school_2(generic=on)
         school=school_3(generic=on)
         year=year_1(generic=on)
         year=year_2(generic=on)
         year=year_3(generic=on)
         )
      );
```

Write the data values to the data component. The _ODS_ option and the PUT statement write the data values for all columns to the data component.

```
put _ods_;
run;
```

Remove the customized means table template. The DELETE statement removes the customized means table that was created in this example. When using the DELETE statement, ODS looks for **means.topn** in Sasuser.Templat and Work.Templat first. If it is there, it will delete it. If not, it will search Sashelp.Tmplmst.

```
proc template;
delete means.topn;
run;
```

HTML Output: Using a Customized Table for the TopN Report

Output 15.6 *HTML Output for the TopN Report*

Top Three Fund Raisers

from 1992 to 1994

Schools	Years	Total Dollars Raised	Average Dollars per Student	Individual Dollars	Student	School	Year
Kennedy	All	$1575.95	$29.73	$72.22 $52.63 $43.89	Luther Thelma Jenny	Kennedy Kennedy Kennedy	1994 1992 1992
Monroe	All	$1606.80	$28.69	$78.65 $65.44 $56.87	Willard Cameron L.T.	Monroe Monroe Monroe	1994 1993 1994
All	1992	$882.92	$28.48	$55.16 $53.76 $52.63	Tonya Edward Thelma	Monroe Monroe Kennedy	1992 1992 1992
All	1993	$907.92	$28.37	$65.44 $47.33 $42.23	Cameron Myrtle Bill	Monroe Monroe Kennedy	1993 1993 1993
All	1994	$1391.91	$30.26	$78.65 $72.22 $56.87	Willard Luther L.T.	Monroe Kennedy Monroe	1994 1994 1994
All	All	$3182.75	$29.20	$78.65 $72.22 $65.44	Willard Luther Cameron	Monroe Kennedy Monroe	1994 1994 1993

(report generated on 13APR2020)

Example 4: Setting the Style Element for Cells Based on Their Values

Features:

DEFINE TABLE statement
- NMVAR statement
- NOTES statement
- TRANSLATE INTO statement

DEFINE COLUMN statement
- BLANK_DUPS= attribute
- CELLSTYLE AS statement
- GENERIC= attribute

Other ODS features
- DELETE statement
- FILE statement with ODS= option
- PUT statement with _ODS_ argument

Data set: Grain_Production

Format: $CNTRY.

Details

This example creates a template that uses different colors and font attributes for the text inside cells, depending on their values.

Program

```
title "Leading Grain Producers";

proc template;
   define table shared.cellstyle;

   translate _val_=. into "No data";

   notes "NMVAR defines symbols that will be used to determine the
colors
 of the cells.";

  nmvar low "Use default style."
        medium "Use yellow foreground color and bold font weight"
        high "Use red foreground color and a bold, italic font.";

   classlevels=on;

   define column char_var;
      generic=on;
      blank_dups=on;
   end;

   define column num_var;
      generic=on;

      justify=on;

%let low=10000;
%let medium=50000;
%let high=100000;

      cellstyle _val_ <= &low as data,
                _val_ <= &medium as data
                         {color=green fontstyle=italic},
                _val_ <= &high as data
                         {color=yellow fontweight=bold},
                    1 as data
                         {color=red fontstyle=italic
                          fontweight=bold};
   end;


   end;
run;

data _null_;
   set grain_production;

   file print ods=(
```

```
        template="shared.cellstyle"

        columns=(
           char_var=year(generic=on)
           char_var=country(generic=on format=$cntry.)
           char_var=type(generic=on)
           num_var=kilotons(generic=on format=comma12.)
           )
        );

   put _ods_;
run;

proc template;
    delete shared.cellstyle;
run;
```

Program Description

Specify a title. The TITLE statement specifies a title.

```
title "Leading Grain Producers";
```

Create the table template Shared.Cellstyle. The DEFINE statement creates the table template Shared.Cellstyle in the first template store in the path that is available to write to. By default, this template store is Sasuser.Templat.

```
proc template;
   define table shared.cellstyle;
```

Specify that missing values show the text "No data" in the report. The TRANSLATE INTO statement translates missing values (.) into the string **`No data`**.

```
    translate _val_=. into "No data";
```

Store the information about the table in the table template. The NOTES statement provides information about the table. NOTES statements remain a part of the compiled table template whereas SAS comments do not.

```
    notes "NMVAR defines symbols that will be used to determine the
colors
 of the cells.";
```

Specify the symbols that reference three macro variables. The NMVAR statement defines three symbols that reference macro variables. ODS will convert the variable's values to numbers (stored as doubles) before using them. References to the macro variables are resolved when ODS binds the template and the data component to produce an output object. The text inside quotation marks provides information about the symbols. This information becomes a part of the compiled table template whereas SAS comments do not. LOW, MEDIUM, and HIGH will contain the values to use as the determinants of the style element that displays the cell. The values are provided just before the DATA step that produces the report.

```
  nmvar low "Use default style."
        medium "Use yellow foreground color and bold font weight"
        high "Use red foreground color and a bold, italic font.";
```

Control the repetition of values that do not change from one row to the next row. The CLASSLEVELS= attribute suppresses the display of the value in a column that is marked with BLANK_DUPS=ON if the value changes in a previous column

that is also marked with BLANK_DUPS=ON. Because BLANK_DUPS= is set in a generic column, set this attribute as well.

```
classlevels=on;
```

Create the column template Char_Var. The DEFINE statement and its attributes create the column template Char_Var. GENERIC= specifies that multiple variables can use the same column template. BLANK_DUPS= suppresses the display of the value in the column if it does not change from one row to the next (and, because CLASSLEVELS=ON for the table, if no value changes in a preceding column that is marked with BLANK_DUPS=ON changes).The END statement ends the template.

```
define column char_var;
   generic=on;
   blank_dups=on;
end;
```

Create the column template Num_Var. The DEFINE statement and its attributes create the column template Num_Var. GENERIC= specifies that multiple variables can use the same column template.

```
define column num_var;
   generic=on;
```

Align the values in the column without regard to the format field. JUSTIFY= justifies the values in the column without regard to the format field. For numeric variables, the default justification is RIGHT, so even the translated character value **No data** that is used for missing values is right-justified. Without JUSTIFY=ON in this column template, the value **No data** is formatted as a character variable (left-justified) within a format field that has the same width as the column.

```
   justify=on;
```

Assign values to three macro variables. The %LET statements assign values to the macro variables LOW, MEDIUM, and HIGH.

```
%let low=10000;
%let medium=50000;
%let high=100000;
```

Specify which style element and style attributes to use for different values in the column. The CELLSTYLE AS statement specifies the style element and style attributes to use for different values in this column. If a value is less than or equal to the value of the variable LOW, the cell uses the unaltered Data style element. If a value is greater than LOW but less than or equal to the value of MEDIUM, the cell uses the style element Data with a foreground color of green and an italic font. Similarly, other values use a foreground color of yellow or red and combinations of a bold font weight and an italic font style. The CELLSTYLE AS statement affects only the HTML destination END statement ends the column template.

```
   cellstyle _val_ <= &low as data,
             _val_ <= &medium as data
                      {color=green fontstyle=italic},
             _val_ <= &high as data
                      {color=yellow fontweight=bold},
                 1 as data
                      {color=red fontstyle=italic
                       fontweight=bold};
end;
```

End the table template. This END statement ends the table template. The RUN statement ends the PROC TEMPLATE step.

```
   end;
run;
```

Create the data component. This DATA step does not create a data set. Instead, it creates a data component, and, eventually, an output object. The SET statement reads the data set Grain_Production.

```
data _null_;
   set grain_production;
```

Route the DATA step results to ODS and use the Shared.CellStyle table template. The combination of the fileref PRINT and the ODS option in the FILE statement routes the results of the DATA step to ODS. The TEMPLATE= suboption tells ODS to use the table template named Shared.CellStyle, which was previously created with PROC TEMPLATE.

```
   file print ods=(
        template="shared.cellstyle"
```

Specify the column template to use for each variable. The COLUMNS= suboption places DATA step variables into columns that are defined in the table template. For example, the first *column-specification* specifies that the first column of the output object contains the values of the variable YEAR and that it uses the column template named Char_Var. GENERIC= must be set to ON, both in the table template and in each column assignment, in order for multiple variables to use the same column template.

```
        columns=(
           char_var=year(generic=on)
           char_var=country(generic=on format=$cntry.)
           char_var=type(generic=on)
           num_var=kilotons(generic=on format=comma12.)
           )
        );
```

Write the data values to the data component. The _ODS_ option and the PUT statement write the data values for all columns to the data component.

```
   put _ods_;
run;
```

Remove the customized table template. The DELETE statement removes the customized table that was created in this example. When using the DELETE statement, ODS looks for **`shared.cellstyle`** in Sasuser.Templat and Work.Templat first. If it is there, it will delete it. If not, it will search Sashelp.Tmplmst.

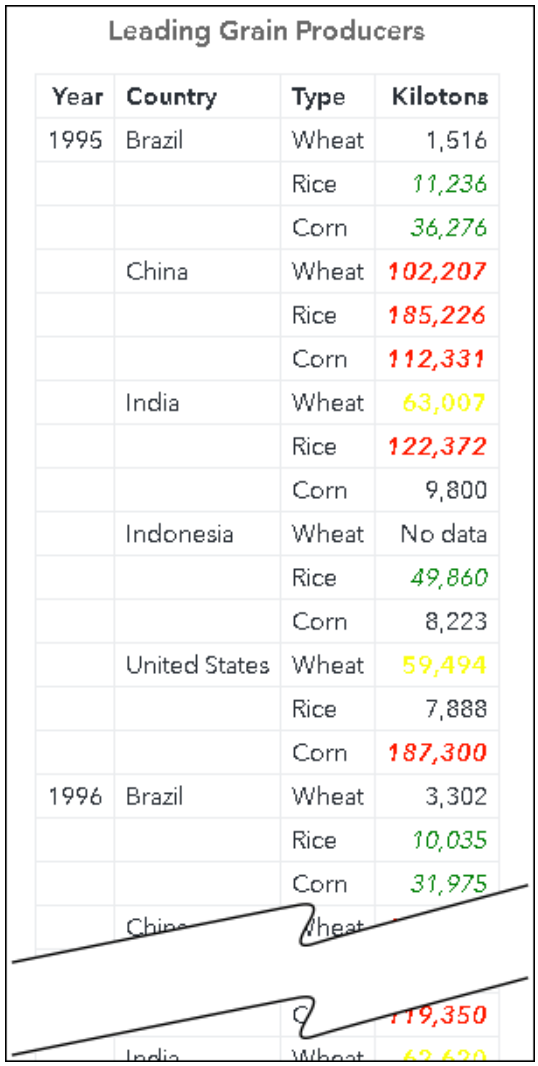
```
proc template;
    delete shared.cellstyle;
run;
```

HTML Output of a Customized Table

Both the table customizations and the style customizations appear in the HTML output. Table customizations include the suppression of values that do not change from one row to the next, and the translation of missing values to **`No data`**. The style

customizations include the colors and font styles that are specified in the CELLSTYLE AS statement.

Output 15.7 *Customized HTML Output*



Leading Grain Producers

Year	Country	Type	Kilotons
1995	Brazil	Wheat	1,516
		Rice	11,236
		Corn	36,276
	China	Wheat	102,207
		Rice	185,226
		Corn	112,331
	India	Wheat	63,007
		Rice	122,372
		Corn	9,800
	Indonesia	Wheat	No data
		Rice	49,860
		Corn	8,223
	United States	Wheat	59,494
		Rice	7,888
		Corn	187,300
1996	Brazil	Wheat	3,302
		Rice	10,035
		Corn	31,975
	Chin	Wheat	
		C	119,350
	India	Wheat	62,620

Example 5: Setting the Style Element for a Specific Column, Row, and Cell

Features:

DEFINE STYLE statement
  REPLACE statement

DEFINE TABLE statement
  CELLSTYLE AS statement

DEFINE COLUMN statement
  DEFINE HEADER statement: TEXT statement

DEFINE HEADER statement
  TEXT statement

Other ODS features

FILE statement with ODS= option
ODS HTML statement: STYLE= option
ODS PDF statement: STYLE= option
PUT statement: _ODS_ argument
ODS TRACE statement

Data set: Exprev

Details

This example combines a customized style with a customized table template to produce output with a checkerboard pattern of table cells.

Program

```
proc template;
   define style greenbar;

   parent=styles.printer;

   replace headersandfooters from cell /
         backgroundcolor=light green
         color=black
         fontsize=3
         fontweight=bold
       ;

    end;
 run;

ods html body="greenbar.html" style=greenbar;
ods pdf file="greenbar.pdf" style=greenbar;

ods trace on;

proc template;
   define table Checkerboard;

cellstyle mod(_row_,2) && mod(_col_,2) as
data{backgroundcolor=yellow fontweight=bold },
               not(mod(_row_,2)) && not(mod(_col_,2)) as
data{backgroundcolor=yellow fontweight=bold },
               1 as data;

define header top;
   text "Checkerboard Table Template";
end;

      define column country;
         dataname=country;
         define header bar;
            text "Country";
         end;
         header=bar;
      end;
```

```
      define column OrderDate;
         dataname=Order_Date;
         define header bar;
            text "Order Date";
         end;
         header=bar;
      end;

      define column ShipDate;
         dataname=Ship_Date;
         define header bar;
            text "Ship Date";
         end;
         header=bar;
      end;

      define column SaleType;
         dataname=Sale_Type;
         define header bar;
            text "Sale Type";
         end;
         header=bar;
      end;

end;
run;

data _null_;
   set work.exprev;

file print ods=(template="Checkerboard");
  put _ods_;
run;

ods html close;
ods pdf close;
ods html;
```

Program Description

Create the new style Greenbar. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style Greenbar.

```
proc template;
   define style greenbar;
```

Specify the parent style from which the Greenbar style inherits its attributes. The PARENT= attribute specifies the style from which the Greenbar definition inherits its style elements and attributes. All the style elements and their attributes that are specified in the parent's definition are used in the current definition unless the current definition overrides them.

```
      parent=styles.printer;
```

Change the colors used in the headers and footers. The REPLACE statement adds a style element to the Greenbar style from the parent style, but the background is light green, the foreground is black, and the font is bold and has a size of 3.

```
   replace headersandfooters from cell /
         backgroundcolor=light green
         color=black
         fontsize=3
         fontweight=bold
       ;
```

End the style. The END statement ends the style. The RUN statement executes the PROC TEMPLATE step.

```
    end;
run;
```

Create the PDF output and specify the style that you want to use for the output. The ODS HTML statement sends all output objects to the file greenbar.html. The STYLE= option tells ODS to use Greenbar as the style when it formats the output. The ODS PDF statement opens the PDF destination and creates PDF output. It sends all output objects to the file greenbar.pdf in the current directory. The STYLE= option tells ODS to use Greenbar as the style when it formats the output.

```
ods html body="greenbar.html" style=greenbar;
ods pdf file="greenbar.pdf" style=greenbar;

ods trace on;
```

Create the table template Checkerboard. The DEFINE statement creates the table template Checkerboard in the first template store in the path that is available to write to. By default, this template store is Sasuser.Templat.

```
proc template;
   define table Checkerboard;
```

Specify which style element and style attributes to use for different cells. The CELLSTYLE-AS statement specifies the style element and style attributes to use for cells in each of the rows and columns. The CELLSTYLE-AS statement creates the checkerboard effect in the output. If both the row and column are odd numbered, then the cell is yellow. Similarly, if both the row and column are even numbered, then the cell is yellow.

```
cellstyle mod(_row_,2) && mod(_col_,2) as
data{backgroundcolor=yellow fontweight=bold },
                not(mod(_row_,2)) && not(mod(_col_,2)) as
data{backgroundcolor=yellow fontweight=bold },
                1 as data;
```

Create the header template Top. The DEFINE HEADER statement defines the table header Top. The TEXT statement specifies the text of the header "Checkerboard Table Template". The END statement ends the header template.

```
define header top;
   text "Checkerboard Table Template";
end;
```

Create the column template Country. The DEFINE COLUMN statement creates the column template Country. The DEFINE HEADER statement creates the header template Bar. The DATANAME= column attribute specifies the name of the column Country in the data component to associate with the column template Country. The TEXT statement specifies the text to use in the header. The first END statement ends the header template. The HEADER statement declares Bar as the header in the table. The second END statement ends the column template.

```
        define column country;
           dataname=country;
```

```
      define header bar;
         text "Country";
      end;
      header=bar;
   end;
```

Create the column template OrderDate. The DEFINE COLUMN statement creates the column template OrderDate. The DATANAME= column attribute specifies the name of the column OrderDate in the data component to associate with the column template OrderDate. The DEFINE HEADER statement creates the header template Bar. The TEXT statement specifies the text "Order Date" to use in the header. The first END statement ends the header template. The HEADER statement declares Bar as the header in the table. The second END statement ends the column template.

```
   define column OrderDate;
      dataname=Order_Date;
      define header bar;
         text "Order Date";
      end;
      header=bar;
   end;
```

Create the column template ShipDate. The DEFINE COLUMN statement creates the column template ShipDate. The DATANAME= column attribute specifies the name of the column template ShipDate in the data component to associate with the column template ShipDate. The DEFINE HEADER statement creates the header template Bar. The TEXT statement specifies the text "Ship Date" to use in the header. The first END statement ends the header template. The HEADER statement declares Bar as the header in the table. The second END statement ends the column template.

```
   define column ShipDate;
      dataname=Ship_Date;
      define header bar;
         text "Ship Date";
      end;
      header=bar;
   end;
```

Create the column template SaleType. The DEFINE COLUMN statement creates the column template SaleType. The DATANAME= column attribute specifies the name of the column template SaleType in the data component to associate with the column template SaleType. The DEFINE HEADER statement creates the header template Bar. The TEXT statement specifies the text "Sale Type" to use in the header. The first END statement ends the header template. The HEADER statement declares Bar as the header in the table. The second END statement ends the column template.

```
   define column SaleType;
      dataname=Sale_Type;
      define header bar;
         text "Sale Type";
      end;
      header=bar;
   end;
```

End the table template. The END statement ends the table template. The RUN statement executes the TEMPLATE procedure.

```
end;
run;
```

Create the data component. This DATA step does not create a data set. Instead, it creates a data component that is used to produce an output object. The SET statement reads the data set Work.Exprev.

```
data _null_;
   set work.exprev;
```

Route the DATA step results to ODS and use the Checkerboard table template. The combination of the fileref PRINT and the ODS option in the FILE statement routes the results of the DATA step to ODS. The TEMPLATE= suboption tells ODS to use the table template named Checkerboard.

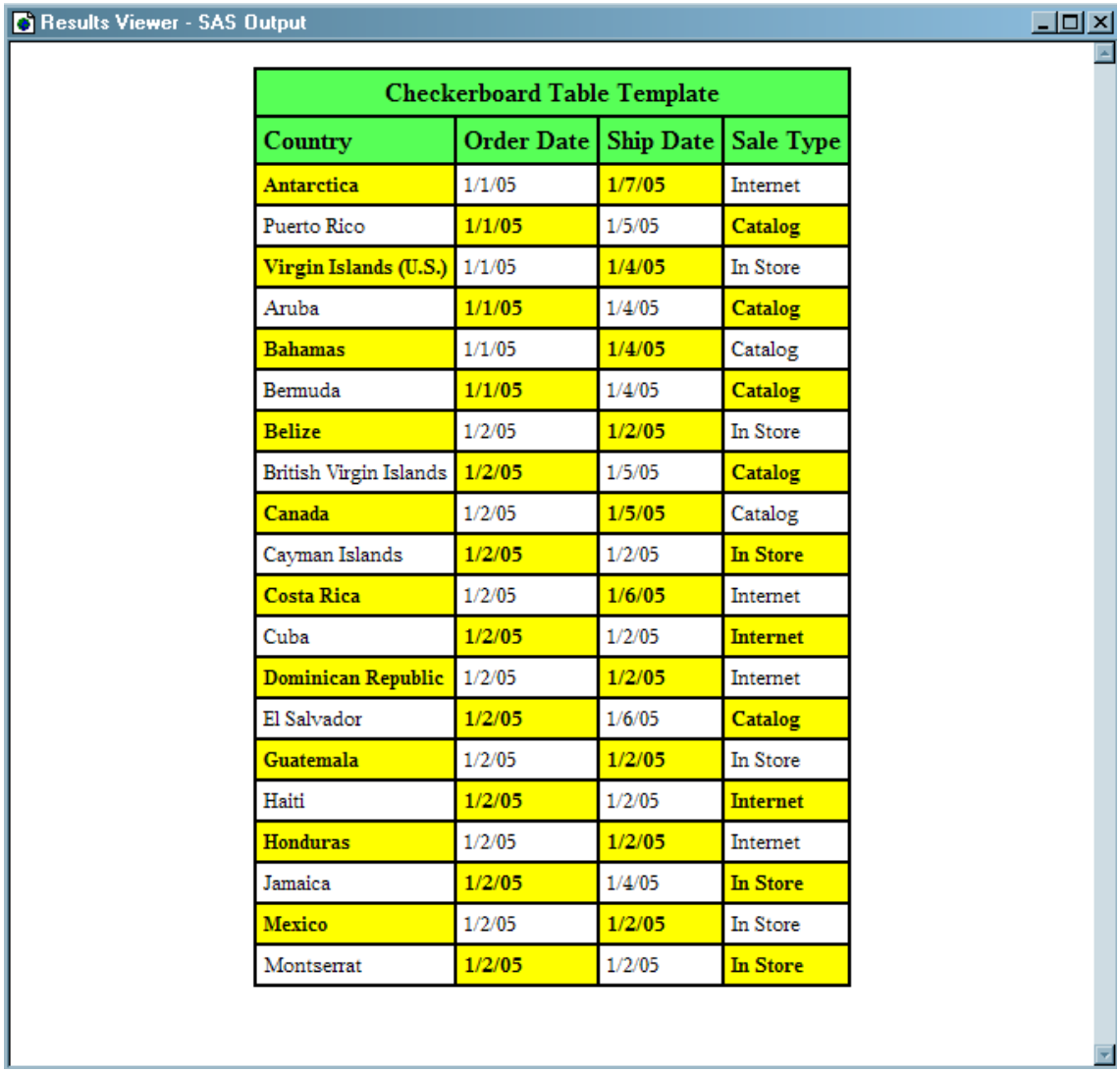
```
file print ods=(template="Checkerboard");
  put _ods_;
run;
```

Stop the creation of PDF output. The ODS HTML CLOSE statement closes the HTML destination and all the files that are associated with it. The ODS PDF CLOSE statement closes the PDF destination and all the files that are associated with it. You must close the PDF destination before you can view the output.

```
ods html close;
ods pdf close;
ods html;
```

Output

Output 15.8 *HTML Output*



Checkerboard Table Template			
Country	**Order Date**	**Ship Date**	**Sale Type**
Antarctica	1/1/05	**1/7/05**	Internet
Puerto Rico	**1/1/05**	1/5/05	**Catalog**
Virgin Islands (U.S.)	1/1/05	**1/4/05**	In Store
Aruba	**1/1/05**	1/4/05	**Catalog**
Bahamas	1/1/05	**1/4/05**	Catalog
Bermuda	**1/1/05**	1/4/05	**Catalog**
Belize	1/2/05	**1/2/05**	In Store
British Virgin Islands	**1/2/05**	1/5/05	**Catalog**
Canada	1/2/05	**1/5/05**	Catalog
Cayman Islands	**1/2/05**	1/2/05	**In Store**
Costa Rica	1/2/05	**1/6/05**	Internet
Cuba	**1/2/05**	1/2/05	**Internet**
Dominican Republic	1/2/05	**1/2/05**	Internet
El Salvador	**1/2/05**	1/6/05	**Catalog**
Guatemala	1/2/05	**1/2/05**	In Store
Haiti	**1/2/05**	1/2/05	**Internet**
Honduras	1/2/05	**1/2/05**	Internet
Jamaica	**1/2/05**	1/4/05	**In Store**
Mexico	1/2/05	**1/2/05**	In Store
Montserrat	**1/2/05**	1/2/05	**In Store**

Output 15.9 *PDF Output (Viewed with Acrobat Reader 5.0)*



Checkerboard Table Template			
Country	Order Date	Ship Date	Sale Type
Antarctica	1/1/05	1/7/05	Internet
Puerto Rico	1/1/05	1/5/05	Catalog
Virgin Islands (U.S.)	1/1/05	1/4/05	In Store
Aruba	1/1/05	1/4/05	Catalog
Bahamas	1/1/05	1/4/05	Catalog
Bermuda	1/1/05	1/4/05	Catalog
Belize	1/2/05	1/2/05	In Store
British Virgin Islands	1/2/05	1/5/05	Catalog
Canada	1/2/05	1/5/05	Catalog
Cayman Islands	1/2/05	1/2/05	In Store
Costa Rica	1/2/05	1/6/05	Internet
Cuba	1/2/05	1/2/05	Internet
Dominican Republic	1/2/05	1/2/05	Internet
El Salvador	1/2/05	1/6/05	Catalog
Guatemala	1/2/05	1/2/05	In Store
Haiti	1/2/05	1/2/05	Internet
Honduras	1/2/05	1/2/05	Internet
Jamaica	1/2/05	1/4/05	In Store
Mexico	1/2/05	1/2/05	In Store
Montserrat	1/2/05	1/2/05	In Store

Example 6: Creating Top-Level Templates

Features: DEFINE TABLE statement
CELLSTYLE AS statement: STYLE_ variable
CELLSTYLE AS statement: _ROW_ variable
DEFINE COLUMN statement
CELLSTYLE AS statement:_VAL_ variable
STYLE= column attribute

DEFINE HEADER statement
    STYLE= column attribute
LINK statement

Details

The following program creates four top-level templates for tables: Base.Template.Table, Base.Template.Column, Base.Template.Header, and Base.Template.Footer. These templates contain style information that creates alternating blue and green row colors and specific styles for headers and footers. Once they are created, top-level templates are applied to every table created by SAS until you specifically remove the top-level template or it is overridden by another table template created by PROC TEMPLATE.

Program

```
proc template;
define table base.template.table;
   cellstyle mod(_row_, 2) and _style_ ^= "RowHeader" as
{background=blue
color=white},
             mod(_row_, 2) and _style_ = "RowHeader" as
{background=green
color=white};
end;

define column base.template.column;
   style={fontstyle=italic};
   cellstyle _val_ > 5 as {fontsize=15pt},
             _val_ = "Num" as {fontsize=20pt};
end;


define header base.template.header;
   style={fontsize=20pt color=purple};
end;
link base.template.footer to base.template.header;
run;

ods select variables;
proc contents data=sashelp.class;
run;

proc template;
delete base.template.table;
delete base.template.column;
delete base.template.header;
delete base.template.footer;

run;
```

Program Description

Create the top-level parent Base.Template.Table and specify which style element and style attributes to use for different cells in a row. The DEFINE TABLE statement creates the top-level parent Base.Template.Table. This template will be applied to every table created by SAS, unless it is overridden by another template created by PROC TEMPLATE, removed with the DELETE statement, or manually removed from the item store. The CELLSTYLE-AS statement specifies the style element and style attributes to use for cells in each of the rows in a table, which creates the alternating row colors in the output. If the row is even numbered and does not contain a style element named RowHeader, then the cell has a green background color and white font color. Similarly, if the row is even numbered and does contain a style element named RowHeader, then the cell has a blue background color and white font color.

```
proc template;
define table base.template.table;
   cellstyle mod(_row_, 2) and _style_ ^= "RowHeader" as
{background=blue
color=white},
             mod(_row_, 2) and _style_ = "RowHeader" as
{background=green
color=white};
end;
```

Create the top-level parent Base.Template.Column and specify which style element and style attributes to use for different cells in a column. The DEFINE TABLE statement creates the top-level parent Base.Template.Column. This template will be applied to every table created by SAS, unless it is overridden by another template created by PROC TEMPLATE, removed with the DELETE statement, or manually removed from the item store. The STYLE= column attribute specifies that column fonts are italicized. The first CELLSTYLE-AS statement specifies that if the value of the cell is greater than five, then the font size is 15pt; and if the value of the cell is equal to "Num", then the font size is 20pt.

```
define column base.template.column;
   style={fontstyle=italic};
   cellstyle _val_ > 5 as {fontsize=15pt},
             _val_ = "Num" as {fontsize=20pt};
end;
```

Create the top-level parent Base.Template.Header and specify the font size and font color for the headers and footers. The DEFINE TABLE statement creates the top-level parent Base.Template.Header. The STYLE= header attribute specifies that the header font is 20pt and purple. The LINK statement creates the Base.Template.Footer top-level template and links it to the Base.Template.Header template, which it inherits its characteristics from. Base.Template.Header and Base.Template.Footer will be applied to every table created by SAS, unless they are overridden by another template created by PROC TEMPLATE, removed with the DELETE statement, or manually removed from the item store.

```
define header base.template.header;
   style={fontsize=20pt color=purple};
end;
link base.template.footer to base.template.header;
run;
```

View the contents of the SAS data set. The CONTENTS procedure shows the contents of the SAS data set Sashelp.Class.

```
ods select variables;
proc contents data=sashelp.class;
run;
```

Delete the top-level templates. The DELETE statement deletes each top-level template. If you do not delete them, they will be applied to all of your tabular output until you do delete them.

```
proc template;
delete base.template.table;
delete base.template.column;
delete base.template.header;
delete base.template.footer;

run;
```

Output

Output 15.10 *Using Top-Level Templates for HTML Output*

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

Example 7: Table Header and Footer Border Formatting

Features: Border control style attributes
BORDERBOTTOMCOLOR=
BORDERBOTTOMSTYLE=
BORDERBOTTOMWIDTH=
BORDERTOPCOLOR=
BORDERTOPSTYLE=
BORDERTOPWIDTH=

DEFINE statement

DEFINE STYLE statement
EDIT statement
FOOTER statement
HEADER statement
PARENT= statement
PREFORMATTED= header attribute
STYLE statement
WIDTH= header attribute
Other ODS features
ODS RTF
ODS SELECT

Data set: Stats and Stats2

Details

You can use the TableHeaderContainer and TableFooterContainer style elements along with the border control style attributes to change the borders of the regions surrounding the table header and footer.

Note: The TableHeaderContainer and TableFooterContainer style elements are valid only in the RTF destination.

Program

```
options nodate nonumber;
title "TableHeaderContainer, TableFooterContainer, and Border Control
Style"
   " Attributes";
title2 "Allows Control of Borders Between the Header, Body, and Footer
of a"
   " Table";

ods html close;

proc template;
   define style  HeadersFootersBorders;
   parent=styles.rtf;

   style TableHeaderContainer from TableHeaderContainer /
      borderbottomwidth=12
      borderbottomcolor=blue
      borderbottomstyle=dotted;

   style TableFooterContainer from TableFooterContainer /
      bordertopwidth=6
      bordertopcolor=red
      bordertopstyle=double;
```

```
   style table from table /
      cellspacing=0 rules=groups frame=void;
   end;
run;

proc template;
   edit Base.Datasets.Members;
      header hd1;
      footer ft1;
      define hd1;
         preformatted=on;
         just=l;
         text"     Table Header with Leading and Trailing Blanks    ";
      end;
      define ft1;
         preformatted=on;
         just=l;
         text"     Table Footer with Leading and Trailing Blanks    ";
      end;
      edit name;
       define header myheader;
         just=l;
         preformatted=on;
         text "     My new header";
       end;
       header=myheader;
       width=memname_width width_max=memname_width_max;
       preformatted=on;
      end;
   end;
run;

ods rtf file="headerfooters.rtf" style=HeadersFootersBorders;
ods select members;
proc datasets lib=work;
run;
quit;

ods rtf close;
```

Program Description

Specify titles. The TITLE statements specify titles for the output.

```
options nodate nonumber;
title "TableHeaderContainer, TableFooterContainer, and Border Control
Style"
   " Attributes";
title2 "Allows Control of Borders Between the Header, Body, and Footer
of a"
   " Table";
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination to conserve system resources.

```
ods html close;
```

Create the new style HeadersFootersBorders. The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style HeadersFootersBorders. The PARENT= statement specifies that the new style inherits all of its style elements and style attributes from the Styles.RTF style.

```
proc template;
   define style  HeadersFootersBorders;
   parent=styles.rtf;
```

Modify the TableHeaderContainer style element. The STYLE statement with the FROM option specified creates the style element TableHeaderContainer, which inherits all of its style elements and style attributes from the instance of TableHeaderContainer in the Styles.RTF style. The BORDERBOTTOMWIDTH=, BORDERBOTTOMCOLOR=, and BORDERBOTTOMSTYLE= style attributes specify the width, color, and line style of the bottom border of the table header.

```
      style TableHeaderContainer from TableHeaderContainer /
         borderbottomwidth=12
         borderbottomcolor=blue
         borderbottomstyle=dotted;
```

Modify the TableFooterContainer style element. The STYLE statement with the FROM option specified creates the style element TableFooterContainer, which inherits all of its style elements and style attributes from the instance of TableFooterContainer in the Styles.RTF style. The BORDERTOPWIDTH=, BORDERTOPCOLOR=, and BORDERTOPSTYLE= style attributes specify the width, color, and line style of the top border of the table footer.

```
      style TableFooterContainer from TableFooterContainer /
         bordertopwidth=6
         bordertopcolor=red
         bordertopstyle=double;
```

Modify the Table style element. The STYLE statement with the FROM option specified creates the style element Table, which inherits all of its style elements and style attributes from the instance of Table in the Styles.RTF style. The CELLSPACING=, RULES=, and FRAME= attributes modify the cellspacing, rules, and frame of the table.

```
      style table from table /
         cellspacing=0 rules=groups frame=void;
      end;
   run;
```

Edit the Base.Datasets.Members table template. The EDIT statement, along with the table template DEFINE statements and attributes, modifies the Base.Datasets.Members table template.

```
   proc template;
      edit Base.Datasets.Members;
         header hd1;
         footer ft1;
         define hd1;
            preformatted=on;
            just=l;
            text"     Table Header with Leading and Trailing Blanks    ";
         end;
         define ft1;
            preformatted=on;
            just=l;
            text"     Table Footer with Leading and Trailing Blanks    ";
```

```
      end;
      edit name;
       define header myheader;
         just=l;
         preformatted=on;
         text "     My new header";
       end;
       header=myheader;
       width=memname_width width_max=memname_width_max;
       preformatted=on;
      end;
   end;
run;
```

Create the RTF file, select the output object and run PROC DATASETS. The ODS RTF statement specifies the file that will contain the RTF output. The STYLE= option specifies the style to apply to the output. The ODS SELECT statement selects the output object Members to be sent to the open destinations.

```
ods rtf file="headerfooters.rtf" style=HeadersFootersBorders;
ods select members;
proc datasets lib=work;
run;
quit;
```

Close the open destinations and open the LISTING destination. The ODS _ALL_ CLOSE statement closes all open destinations and the files that are associated with them. If you do not close the destinations, then you will not be able to view the files. The ODS LISTING statement opens the LISTING destination.

```
ods rtf close;
```

RTF Output

Output 15.11 *RTF Output with Custom Headers and Footers*

TableHeaderContainer, TableFooterContainer, and Border Control Style Attributes Allows Control of Borders Between the Header, Body, and Footer of a Table

Table Header with Leading and Trailing Blanks

#	My new header	Member Type	File Size	Last Modified
1	STATS	DATA	5120	19Oct10:14:31:34
2	STATS2	DATA	5120	19Oct10:14:31:34

Table Footer with Leading and Trailing Blanks

16

Tabular Attributes

Understanding Table Templates, Table Elements, and Table Attributes 585
Column Attributes 586
Header and Footer Attributes 603
Table Attributes 614

Understanding Table Templates, Table Elements, and Table Attributes

A *table template* describes how to generate the output for a tabular output object. (Most ODS output is tabular.) A table template determines the order of column headings and the order of variables, as well the overall look of the output object that uses it. For information about customizing the table template, see the topic on the TEMPLATE procedure in Chapter 15, “TEMPLATE Procedure,” on page 515.

In addition to the parts of the table template that order the headers and columns, each table template contains or references *table elements*. A table element is a collection of table attributes that apply to a particular header, footer, or column. Typically, a *table attribute* specifies something about the data rather than about its presentation. For example, FORMAT specifies the SAS format, such as the number of decimal places. However, some table attributes describe presentation aspects of the data, such as how many blank characters to place between columns.

Note: The attributes of table templates that control the presentation of the data have no effect on output objects that go to the LISTING or OUTPUT destination. However, the attributes that control the structure of the table and the data values do affect LISTING output.

For information about table attributes, see “Table Attributes” on page 614.

Column Attributes

This section lists all the attributes that you can use in a column template. Column attributes are used to customize the attributes of a column and must be specified within a DEFINE COLUMN statement block. You can specify multiple column attributes together or separately. For example, you can specify the following column attributes together:

```
blank_dups=yes data_format_override=off just=l vjust=r;
```

or separately:

```
blank_dups=yes;
data_format_override=OFF;
just=l vjust=r;
```

For all of the attributes that support a value of ON, these forms are equivalent:

ATTRIBUTE-NAME
ATTRIBUTE-NAME=ON

For all of the attributes that support a value of *variable*, *variable* is any variable that you declare in the column template with the DYNAMIC, MVAR, or NMVAR statement. If the attribute is Boolean, then the value of *variable* should resolve to either true or false as shown in this table:

Table 16.1 *Boolean Values*

True	False
ON	OFF
_ON_	_OFF_
1	0
TRUE	FALSE
YES	NO
YES	_NO_

Table 16.2 *Column Attributes*

Task	Attribute	Destinations
Influence the appearance of the cells contents		

Task	Attribute	Destinations
Specify whether to suppress the value of a variable from one row to the next, if the value does not change based on the formatted value of the variable	BLANK_DUPS= on page 590	All except OUTPUT
Specify whether to suppress the value of a variable from one row to the next if the value does not change based on the raw value of the variable	BLANK_INTERNAL_DUPS= on page 590	All except OUTPUT
Select the best format for a column of a table	CHOOSE_FORMAT= on page 591	All
Specify whether to wrap the text in the current column	FLOW= on page 592	LISTING
Specify the format for the column	FORMAT= on page 592	All
Specify the number of decimals for the column if it is not specified with FORMAT= column attribute	FORMAT_NDEC= on page 593	All
Specify the format width for the column if it is not specified with FORMAT= column attribute	FORMAT_WIDTH= on page 593	All
Supply a numeric value against which values in the column are compared to eliminate trivial values from printing	FUZZ= on page 593	All except OUTPUT
Specify the horizontal justification of the format field within the column (and for the column header if the template for the header does not include JUST=)	JUST= on page 595	All except OUTPUT
Specify whether to justify the format field within the column, or to justify the value within the column, without regard to the format field	JUSTIFY= on page 596	All destinations except LISTING behave as if JUSTIFY=ON
When the text in the column uses more than one line, specify whether to try to divide the text equally among all lines	MAXIMIZE= on page 597	LISTING

Task	Attribute	Destinations
or to maximize the amount of text in each line		
Specify whether to draw a continuous line in the current column above the first table footer or below the last row of the column if there is no table footer	OVERLINE= on page 597	LISTING
Specify whether to treat the text as preformatted text	PREFORMATTED= on page 598	Markup family, printer family, and RTF
Specify whether to print the column	PRINT= on page 599	All except OUTPUT
Specify a separator character to append to each value in the column	SEPARATOR= on page 599	LISTING
Specify the style element and style attributes to use for the column	STYLE= on page 600	Markup family, printer family, and RTF
Specify that the text graphic columns be turned off when a procedure is going to output a graph	TEXT_GRAPHIC= on page 601	All except OUTPUT and DOCUMENT
Specify the split character for the data in the column	TEXT_SPLIT= on page 601	All except OUTPUT
Specify whether to draw a continuous line in the current column below the column header, or above the first row of the column if there is no column header	UNDERLINE= on page 601	LISTING
Specify the vertical justification for the column	VJUST= on page 602	Markup family, printer family, and RTF
Specify the width of the column in characters	WIDTH= on page 602	LISTING
Specify the maximum width for this column	WIDTH_MAX= on page 602	LISTING
Customize column headers		

Task	Attribute	Destinations
Specify the text for the column header or the name of the header template	HEADER= on page 594	All
Specify whether to print the column header	PRINT_HEADERS= on page 599	All except OUTPUT
Influence the relationship to other columns		
Specify whether the column template is generic (that is, whether more than one variable use the template)	GENERIC= on page 593	All except OUTPUT
Specify whether the column is an ID column	ID= on page 595	LISTING and printer family
Specify whether to merge the current column with the column immediately to its right	MERGE= on page 597	All except OUTPUT
Specify whether to merge the current column with the column immediately to its left	PRE_MERGE= on page 598	All except OUTPUT
Specify the number of blank characters to leave between the current column and the column immediately to its left	PRE_SPACE= on page 599	LISTING
Specify the number of blank characters to leave between the current column and the column immediately to its right	SPACE= on page 600	LISTING
Influence the presentation of data panels		
Influence the place at which ODS splits a table when it creates multiple data panels	GLUE= on page 594	LISTING, printer family, and RTF
Specify whether to delete the current column from the output object if doing so enables all the remaining columns to fit in the space that is provided without splitting the table into multiple data panels	OPTIONAL= on page 597	LISTING
Other column attributes		

Task	Attribute	Destinations
Specify which format to use if both a column template and a data component specify a format	DATA_FORMAT_OVERRIDE= on page 592	All
Specify the name of the column in the data component to associate with the current column	DATANAME= on page 592	All
Specify which special characters in headers for generic columns are to be used as split characters	DEF_SPLIT on page 592	All
Specify whether to include the column in an output data set	DROP= on page 592	OUTPUT
Specify a label for the column	LABEL= on page 596	OUTPUT
Specify the column template that the current template inherits from	PAREN= on page 598	All
Specify the name to use for the corresponding variable in an output data set	VARNAME= "VARNAME=*variable-name* | *variable*;" on page 601	OUTPUT

BLANK_DUPS<=ON | OFF | *variable*>;
specifies whether to suppress the value of a variable from one row to the next if the value does not change according to the formatted value of the variable.

Default OFF

Interaction If the CLASSLEVELS= table attribute is in effect, then ODS ignores BLANK_DUPS=ON when any value changes in a preceding column that is also marked with BLANK_DUPS=ON.

Tips The BLANK_DUPS attribute is valid in all destinations except the OUTPUT destination.

When the PRINTER destination suppresses the value of a variable, it also suppresses the horizontal rule above the blank cell.

See CLASSLEVELS= table attribute on page 619

Example "Example 4: Setting the Style Element for Cells Based on Their Values" on page 565

BLANK_INTERNAL_DUPS<=ON | OFF | *variable*>;
specifies whether to suppress the value of a variable from one row to the next if the value does not change according to the raw value of the variable.

Default OFF

Interaction If the CLASSLEVELS= table attribute is in effect, then ODS ignores BLANK_INTERNAL_DUPS=ON when any value changes in a preceding column that is also marked with BLANK_INTERNAL_DUPS=ON.

Tips The BLANK_INTERNAL_DUPS attribute is valid in all destinations except the OUTPUT destination.

When the PRINTER destination suppresses the value of a variable, it also suppresses the horizontal rule above the blank cell.

See CLASSLEVELS= table attribute on page 619

CHOOSE_FORMAT= COMPROMISE | MAX | MAX_ABS | MIN_MAX;
selects a format based on the actual values in the column of the table.

COMPROMISE
looks at all of the values in the column and selects a good compromise format that works well for most values, but extreme values might shift to BEST format.

Tips FORMAT_NDEC=d specifies the precision in digits.

The FORMAT_WIDTH= option suggests a maximum width. The actual format width might be smaller or it might be larger.

MAX
selects a format based on the maximum value in the column. Values are all expected to be positive, so no space is reserved for a minus sign.

Default By default, FORMAT_WIDTH=10 and FORMAT_NDEC= is ignored.

MAX_ABS
selects a format based on the maximum absolute value in the column. The format reserves space for a minus sign whether it is needed or not.

MIN_MAX
selects a format based on the minimum and maximum value in the column. The format reserves space for a minus sign only where it is actually needed.

Interaction If FORMAT_NDEC=*d* is specified, a maximum of *d* decimal places is used.

Default If you omit the CHOOSE_FORMAT column attribute, then the default format is determined by either the data component or other attributes.

Restriction CHOOSE_FORMAT is not supported for computed columns because those columns' values are computed outside of the data object.

Tips If you specify a small value for the FORMAT_WIDTH= option, then CHOOSE_FORMAT might create a dw.3 format.

The CHOOSE_FORMAT= attribute is valid in all destinations.

See For more information about column formats, see "Formatting Values in Table Columns" on page 548.

DATA_FORMAT_OVERRIDE<=ON | OFF | *variable*>;
specifies which format to use if both a column template and a data component specify a format.

ON
uses the format in the data component.

OFF
uses the format in the column template.

variable
uses the format of the specified variable.

Default OFF

Tip The DATA_FORMAT_OVERRIDE attribute is valid in all destinations.

DATANAME=*column-name*;
specifies the name of the column in the data component to associate with the current column.

Default By default, ODS associates the current column with a column of the same name in the data component.

Tip The DATANAME= attribute is valid in all destinations.

DEF_SPLIT;
specifies that special characters in headers for generic columns are to be used as split characters.

Tip The DEF_SPLIT destination is valid in all destinations.

DROP<=ON | OFF | *variable*>;
specifies whether to include the column in an output data set.

Default OFF

Tip The DROP attribute is valid only in the OUTPUT destination.

FLOW<=ON | OFF | *variable*>;
specifies whether to wrap the text in the current column if it is too long to fit in the space that is provided.

Default ON if the format width of the column is greater than the column width. OFF if the format width of the column is not greater than the column width.

Tips The FLOW attribute is valid only in the LISTING destination.

The HTML and PRINTER destinations always wrap the text if it is too long to fit in the space that is provided.

See MAXIMIZE= on page 597

FORMAT=*format-name <format-width <decimal-width>>* | *variable*;
specifies the format for the column.

Default If you omit the FORMAT= option, then the format that the data component provides is used. If the data component does not provide a format, ODS uses one of the following: BEST8. for integers, D12.3 for doubles, or the length of the variable for character variables.

Restriction If you specify a format width for a numeric column, then its value cannot exceed 32.

Tip The FORMAT= attribute is valid in all destinations.

FORMAT_NDEC= *number* | *variable*;
specifies the number of decimals for the column.

Default The decimal width that is specified with the FORMAT= column attribute

Range Number is a whole number from 0 to 32

Interaction If you specify a decimal width using both the FORMAT= and the FORMAT_NDEC= attributes, then ODS uses the width that you specify with the FORMAT= attribute.

Tip The FORMAT_NDEC= attribute is valid in all attributes.

FORMAT_WIDTH=*positive-integer* | *variable*;
specifies the format width for the column.

Default If you omit the column attribute FORMAT_WIDTH=, then ODS uses the format specified in the FORMAT= attribute.

Range 1 to 32 for numeric variables; operating system limit for character variables

Interaction If you specify a format width using both the FORMAT= and the FORMAT_WIDTH= attributes, then the width that you specify with the FORMAT= attribute is used.

Tip The FORMAT_WIDTH= attribute is valid in all destinations.

FUZZ=*number* | *variable*;
supplies a numeric value against which to compare values in the column to eliminate trivial values from printing. A number whose absolute value is less than or equal to the FUZZ= value is printed as 0. However, the real value of the number is used in any computations based on that number.

Default This is the smallest representable floating-point number on the computer that you are using.

Tip The FUZZ= attribute is valid in all destinations except the OUTPUT destination.

GENERIC<=ON | OFF | *variable*>;
specifies whether the column template can be used by more than one column. Generic columns are useful in tables with many similar columns. For example, the table templates for both PROC SQL and the DATA step define only two columns: one for character variables and one for numeric variables. When a

program runs, it determines which column template the data component should use for each column.

Default OFF

Tip The GENERIC attribute is valid in all destinations except the OUTPUT destination.

Examples "Example 3: Creating a New Table Template " on page 557

"Example 4: Setting the Style Element for Cells Based on Their Values" on page 565

GLUE=*integer* | *variable*;

Influences the places at which ODS splits a table when it creates multiple data panels. ODS creates multiple data panels from a table that is too wide to fit in the allotted space. The higher the value of GLUE= is, the less likely it is that ODS will split the table between the current column and the column to its right.

Default 1

Range -1 to 327

Tips A value of -1 forces the table to split between the current column and the column to its right.

The GLUE= attribute is valid only in the LISTING, printer family, and RTF destinations.

HEADER=*header-specification*;

specifies the text for the column header or the name of the header template. *header-specification* is one of the following:

"*text*"

specifies the actual text of the header.

Requirement *text* must be enclosed in quotation marks.

header-name

specifies the name of a header template to use. Create a header template with the DEFINE HEADER statement (see "DEFINE HEADER Statement" on page 532). If *header-name* is a single-level name, the header template must occur within the current column template.

variable

specifies the name of a variable declared with the DYNAMIC, MVAR, or NMVAR statement. The value of the variable becomes the column header.

LABEL

Uses the label that is specified in the data component for the column header.

Default _LABEL_

Interaction If you are using the OUTPUT destination, then the HEADER= attribute does not change the label of the variable in the data set. To change the label in the data set, use the LABEL= attribute.

Tips The HEADER= option provides a simple way to specify the text of a column header. To customize the header further, use the DEFINE

HEADER statement with the appropriate header attributes. (See “DEFINE HEADER Statement” on page 532.)

Use the split character in the text of the header to force the text to a new line.

The HEADER= attribute is valid in all destinations.

See “LABEL="*text*" | *variable*;” on page 596 and “TEXT_SPLIT="*character*" | *variable*;” on page 601

Examples “Example 3: Creating a New Table Template ” on page 557

“Example 1: Creating a Stand-Alone Style” on page 466

ID<=ON | OFF | *variable*>;
specifies whether the column is an ID column. An ID column is repeated on each data panel. (ODS creates multiple data panels when a table is too wide to fit in the allotted space.)

Default OFF

Tips ODS treats all columns up to and including a column that is marked with ID=ON as ID columns.

The ID attribute is valid only in LISTING and printer family destinations.

Example “Example 3: Creating a New Table Template ” on page 557

JUST=*justification* | *variable*;
specifies the horizontal justification of the format field within the column (and of the header if the template for the header does not include JUST=).

justification is one of the following:

CENTER
specifies center justification.

Alias C

Interaction To use center justification in printer family and RTF destinations, also specify JUSTIFY=ON.

DEC
specifies aligning the values by the decimal point.

Alias D

Restriction Decimal alignment is supported for printer family and RTF destinations only.

LEFT
specifies left justification.

Alias L

RIGHT
specifies right justification.

Alias R

Default LEFT for columns that contain character values; RIGHT for columns that contain numeric values.

Interactions The TEXTALIGN= style attribute on page 870 overrides the value of JUST=.

For the LISTING destination, ODS justifies the format field within the column width. At times, you can specify the JUSTIFY= attribute to get the results that you want. See the discussion of the JUSTIFY attribute on page 596.

Tip The JUST= attribute is valid in all destinations except the OUTPUT destination.

See "Values in Table Columns and How They Are Justified" on page 547

FORMAT= on page 592 and WIDTH= on page 593

Example "Example 1: Editing a Table Template That a SAS Procedure Uses" on page 550

JUSTIFY<=ON | OFF | *variable*>;
specifies whether to justify the format field within the column or to justify the value within the column without regard to the format field.

Default OFF

Interaction JUSTIFY=ON can interfere with decimal alignment in the LISTING destination.

Tips If you translate numeric data to character data, you might need to use JUSTIFY= to align the data.

All destinations except the LISTING destinations justify values as if JUSTIFY=ON.

See "Values in Table Columns and How They Are Justified" on page 547

Example "Example 4: Setting the Style Element for Cells Based on Their Values" on page 565

LABEL="*text*" | *variable*;
specifies a label for the column in the output data set.

Default If you omit a label, ODS uses the label that is specified in the data component. If no label is specified in the data component, ODS uses the header for the column as the label.

Tips The LABEL= attribute is valid only in the OUTPUT destination.

If the OUTPUT destination is open, then the LABEL= attribute provides a label for the corresponding variable in the output data set. This label overrides any label that is specified in the data component.

MAXIMIZE<=ON | OFF | *variable*>;
specifies whether to try to divide the text equally among all lines or to maximize the amount of text in each line when the text in the column uses more than one line. For example, if the text spans three lines, MAXIMIZE=ON can result in 45% of the text on the first line, 45% of the text on the second line, and 10% of the text on the third line. MAXIMIZE=OFF can result in 33% of the text on each line. MAXIMIZE=ON can write lines of text that vary greatly in length. MAXIMIZE=OFF can result in using less than the full column width.

Default OFF

Interaction This attribute is effective only if the column is defined with FLOW=ON. (See the discussion of the FLOW= column attribute on page 592).

Tip The MAXIMIZE= attribute is valid only in the LISTING destination.

MERGE<=ON | OFF | *variable*>;
specifies whether to merge the current column with the column immediately to its right. When you set MERGE=ON for the current column, the data in each row of the column is merged with the data in the same row of the next column. ODS applies the format, justification, spacing, and prespacing attributes to each column independently. Then, it concatenates the columns. Finally, it applies to the concatenated data all the remaining attributes that are specified on the column that does not have MERGE= set.

Default OFF

Restriction You cannot use both MERGE=ON and PRE_MERGE=ON in the same column template. You cannot merge or premerge a column with another column that has either MERGE=ON or PRE_MERGE=ON. Note that you can merge three columns by setting MERGE=ON for the first column, no merge or premerge attributes for the second column, and PRE_MERGE=ON for the third column.

Tip The MERGE= attribute is valid in all destinations except the OUTPUT destination.

See PRE_MERGE= column attribute on page 598

OPTIONAL<=ON | OFF | *variable*>;
specifies whether to delete the current column from the output object if doing so enables all the remaining columns to fit in the space that is provided without splitting the table into multiple data panels.

Default OFF

Interaction If multiple column templates contain OPTIONAL=ON, either all or none of these columns are included in the output object.

Tip The OPTIONAL attribute is valid only in the LISTING destination.

OVERLINE<=ON | OFF | *variable*>;
specifies whether to draw a continuous line in the current column above the first table footer (or, if there is no table footer, below the last row of the column). The second formatting character is used to draw the line.

Default OFF

Tip The OVERLINE= attribute is valid only in the LISTING destination.

See For information about formatting characters, see the FORMCHAR= table attribute on page 621.

PARENT=*variable*;
specifies the column template from which the current template inherits attributes and statements. A *column-path* consists of one or more names that are separated by periods. Each name represents a directory in a template store, which is a type of SAS file. The current template inherits from the specified column in the first readable template store in the current path.

When you specify a parent, all of the attributes and statements that are specified in the parent's template are used in the current template unless the current template specifically overrides them.

Tip The PARENT= attribute is valid in all destinations.

PREFORMATTED<=ON | OFF | *variable*>;
specifies whether to treat the text as preformatted text. When text is preformatted, ODS honors line breaks as well as leading, trailing, and internal spaces. It also displays the text in a monospace font.

Default OFF

Interaction When PREFORMATTED=ON, ODS uses the DataFixed style element unless you specify another style element with the STYLE= column attribute.

Tip The PREFORMATTED attribute is valid in the markup family, printer family, and RTF destinations.

PRE_MERGE<=ON | OFF | *variable*>;
specifies whether to merge the current column with the column immediately to its left. When you set PRE_MERGE=ON for the current column, the data in each row of the column is merged with the data in the same row of the previous column. ODS applies the format, justification, spacing, and prespacing attributes to each column independently. Then, it concatenates the columns. Finally, it applies to the concatenated data all the remaining attributes that are specified on the column that does not have PRE_MERGE= set.

Default OFF

Restriction You cannot use both MERGE=ON and PRE_MERGE=ON in the same column template. You cannot merge or premerge a column with another column that has either MERGE=ON or PRE_MERGE=ON. Note that you can merge three columns by setting MERGE=ON for the first column, no merge or premerge attributes for the second column, and PRE_MERGE=ON for the third column.

Tip The PRE_IMAGE attribute is valid in all destinations except the OUTPUT destination.

See MERGE= column attribute on page 597

PRE_SPACE=*non-negative-integer*;
specifies the number of blank characters to leave between the current column and the column immediately to its left.

Default A value in the range that is bounded by the COL_SPACE_MIN and COL_SPACE_MAX table attributes.

Interaction If PRE_SPACE= and SPACE= are specified for the same intercolumn space, ODS honors PRE_SPACE=.

Tip The PRE_SPACE= attribute is valid only in the LISTING destination.

See SPACE= column attribute on page 600, COL_SPACE_MIN table attribute on page 619, and COL_SPACE_MIN table attribute on page 619.

PRINT<=ON | OFF | *variable*>;
specifies whether to print the column.

Default ON

Interaction If you specify the column attribute PRINT=OFF, then you turn off the value of a column if it is part of a stacked column. If all columns in a stacked column have PRINT=OFF set, then the entire column is removed from the table.

Tips If all columns in a stacked column have PRINT=OFF specified, then the entire column is removed from the table.

The PRINT attribute is valid in all destination except the OUTPUT destination.

See OPTIONAL column attribute on page 597 and DROP= column attribute on page 592

PRINT_HEADERS<=ON | OFF | *variable*>;
specifies whether to print the column header and any underlining and overlining.

Default ON

Tip The PRINT_HEADERS attribute is valid in all destination except the OUTPUT destination.

See UNDERLINE= column attribute on page 601 and OVERLINE= column attribute on page 597.

SEPARATOR="*character*" | *variable*;
specifies a separator character to append to each value in the column.

Default None

Restriction The SEPARATOR= column attribute is valid only for character variables.

Tips To specify a hexadecimal character as the separator character, put an x after the closing quotation mark. For example, this option assigns the hexadecimal character 2-D as the separator character:

```
separator="2D"x
```

The SEPARATOR= attribute is valid only in the LISTING destination.

SPACE=*positive-integer* | *variable*;
specifies the number of blank characters to leave between the current column and the column immediately to its right.

Default A value in the range that is bounded by the COL_SPACE_MIN and COL_SPACE_MAX table attributes.

Interaction If PRE_SPACE= and SPACE= are specified for the same intercolumn space, ODS honors PRE_SPACE=.

Tip The SPACE= attribute is valid only in the LISTING destination.

See The PRE_SPACE= column attribute on page 599, the COL_SPACE_MAX= attribute on page 619, and the "COL_SPACE_MIN= *positive-integer* | *variable*;" on page 619.

STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>;
specifies the style element and any changes to its attributes to use for the current column. Neither *style-attribute-specification* nor *style-element-name* is required. However, you must use at least one of them.

Note: You can use braces ({ and }) instead of square brackets ([and]).

style-element-name
is the name of the style element to use to display the data in the column. The style element must be part of a style that is registered with the Output Delivery System. SAS provides some styles. You can create customized styles with PROC TEMPLATE (see Chapter 14, "TEMPLATE Procedure," on page 433).

By default, ODS displays different parts of ODS output with different style elements. For example, by default, the data cells in a column are displayed with the style element Data. You would be most likely to use the following style elements with the STYLE= column attribute:

- Data
- DataFixed
- DataEmpty
- DataEmphasis
- DataEphasisFixed
- DataStrong
- DataStrongFixed

The style element provides the basis for displaying the column. Additional style attributes that you provide can modify the display.

For information about viewing a style so that you can see the style elements that are available, see "Working with Styles " on page 438. For information about the default style that ODS uses, see "Modifying Style Elements in the Default Style for HTML and Markup Languages" on page 440.

style-element-name is either the name of a style element or a variable whose value is a style element. For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

Default Data

style-attribute-specification
describes the style attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

For information about the style attributes that you can specify, see “About Style Attributes” on page 465.

Tips The STYLE= attribute is valid only in the markup family, printer family, and RTF destinations.

If you use the STYLE= attribute inside a quoted string, then add a space before or after the carriage return to prevent errors. SAS does not interpret a carriage return as a space. You must explicitly specify spaces in quoted strings.

Example “Example 3: Creating a New Table Template ” on page 557

TEXT_GRAPHIC= ON | OFF;
specifies that the text graphic columns be turned off or on when a procedure is going to output a graph.

Default OFF

TEXT_SPLIT="*character*" | *variable*;
specifies the split character for the data in the column. The value in the column is broken when it reaches that character and continues the value on the next line. The split character itself is not part of the data and does not appear in the column.

Default None

Tip The TEXT_SPLIT= attribute is valid in all destinations except the OUTPUT destination.

UNDERLINE<=ON | OFF | *variable*>;
specifies whether to draw a continuous line in the current column below the column header (or, if there is no column header, above the first row of the column). The second formatting character is used to draw the line.

Default OFF

Tip The UNDERLINE= attribute is valid only in the LISTING destination.

See For information about formatting characters, see the FORMCHAR= table attribute on page 621.

VARNAME=*variable-name* | *variable*;
specifies the name to use for the corresponding variable in an output data set.

Default If you omit VARNAME=, then the value of the DATANAME= attribute is used. If you omit DATANAME=, then the name of the column is used.

Tips If you use VARNAME= to specify the same name for different columns, a number is appended to the name each time that the name is used.

The VARNAME= attribute is valid only in the OUTPUT destination.

VJUST=*justification* | *variable*;
Specifies the vertical justification for the column. *justification* is one of the following:

TOP
places the first line of text as high as possible.

Alias T

CENTER
centers the text vertically.

Alias C

BOTTOM
places the last line of text as low as possible.

Alias B

Default TOP

Tip The VJUST= attribute is valid only in the markup family, printer family, and RTF destinations.

Example “Example 3: Creating a New Table Template ” on page 557

WIDTH=*positive-integer* | *variable*;
specifies the width of the column in characters.

Default If you omit a width, the format width is used. If the column has no format associated with it, ODS uses one of the following widths: 8 for integers, 12 for doubles, or data length for character variables.

Interaction The length of the column header can influence the width of the column.

Tip The WIDTH= attribute is valid only in the LISTING destination.

WIDTH_MAX=*positive-integer* | *variable*;
specifies the maximum width allowed for this column. By default, PROC TEMPLATE extends the width of the column if the header is wider than the data. The width of the column can be anywhere between the values of WIDTH= and WIDTH_MAX=.

Default The width of the format for the column

Tip The WIDTH_MAX= attribute is valid only in the LISTING destination.

Header and Footer Attributes

This section lists all the attributes that you can use in a header or footer template. Header and footer attributes are used to customize the attributes of a header or footer. They can be specified only within a DEFINE HEADER statement block or a DEFINE FOOTER statement block. You can specify multiple attributes together or separately. For example, you can specify the following attributes together:

```
overline=on underline=on just=c;
```

or separately:

```
overline=on;
underline=on;
just=c;
```

A column header spans a single column. A spanning header spans multiple columns. These two types of headers are defined in the same way except that a spanning header uses the START= or the END= attribute, or both.

For all attributes that support a value of ON, these forms are equivalent:

ATTRIBUTE-NAME
ATTRIBUTE-NAME=ON

For all of the attributes that support a value of *variable*, *variable* is any variable that you declare in the table template with the DYNAMIC, MVAR, or NMVAR statement. If the attribute is a Boolean, then the value of *variable* should resolve to either true or false as shown in this table:

Table 16.3 *Boolean Values*

True	False
ON	OFF
_ON_	_OFF_
TRUE	FALSE
YES	NO
YES	_NO_

Table 16.4 *Header Attributes*

Task	Attribute	Destinations
Influence the appearance of the contents of the header		

Task	Attribute	Destinations
Specify that special characters in headers for generic columns are to be used as split characters	DEF_SPLIT on page 592	All
Specify a character to use to expand the header to fill the space over the column or columns that the header spans	EXPAND= on page 607	LISTING
Specify whether to try to expand the column width to accommodate the longest word in the column header	FORCE= on page 607	LISTING
Specify the horizontal justification for the column header	JUST= on page 608	All except OUTPUT
Specify whether to try to divide the text equally among all lines or to maximize the amount of text in each line when the text in the header uses more than one line	MAXIMIZE= on page 609	LISTING
Specify whether to draw a continuous line above the header	OVERLINE= on page 609	LISTING
Specify whether to treat the text as preformatted text	PREFORMATTED= on page 609	Markup family, printer family, and RTF
Specify whether to print the header	PRINT= on page 599	All
Specify whether to repeat the text of the header until the space that is allotted for the header is filled	REPEAT= on page 610	LISTING
Specify the number of blank lines to place between the current header and the next header or between the current footer and the previous footer	SPACE= on page 610	LISTING
Specify the split character for the header	SPLIT= on page 611	All except OUTPUT

Task	Attribute	Destinations
Specify the style element and any changes to its attributes to use for the header	STYLE= on page 600	Markup family, printer family, and RTF
Specify whether to start a new header line in the middle of a word	TRUNCATE= on page 613	LISTING
Specify whether to draw a continuous line underneath the header	UNDERLINE= on page 601	LISTING
Specify vertical justification for the header	VJUST= on page 613	Markup family, PRINTER, family, and RTF
Specify the width of the header in characters	WIDTH= on page 602	LISTING
Influence the content of the header		
Influence the placement of the header		
Specify the last column that a spanning header covers	END= on page 607	All except OUTPUT
Specify whether to expand the header to reach the sides of the page	EXPAND_PAGE= on page 607	LISTING
Specify whether a spanning header appears only on the first data panel if the table is too wide to fit in the space that is provided	FIRST_PANEL= on page 607	LISTING, printer family, and RTF
Specify whether a table footer appears only on the last data panel if the table is too wide to fit in the space that is provided	LAST_PANEL= on page 608	LISTING, printer family, and RTF
Specify whether to extend the text of the header into the header space of adjacent columns	SPILL_ADJ= on page 610	LISTING
Specify the first column that a spanning header covers	START= on page 611	All except OUTPUT

Task	Attribute	Destinations
Specify whether to extend the text of the header into the adjacent margin	SPILL_MARGIN= on page 610	LISTING
Other header attributes		
Specify an abbreviation for the header	ABBR= on page 606	MARKUP
Specify an acronym for the header	ACRONYM= on page 606	MARKUP
Specify an alternate description for the header	ALT= on page 606	MARKUP
Specify whether multiple columns can use the header	GENERIC= on page 608	All except OUTPUT
Specify a long description for the header	LONGDESC= on page 608	MARKUP
Specify the header template that the current template inherits from	PARENT= on page 609	All

ABBR= "*text*" | *variable*;
specifies an abbreviation for the header.

Requirement The text must be enclosed in quotation marks.

Tip The ABBR attribute is valid only in the MARKUP destination.

ACRONYM= "*text*" | *variable*;
specifies an acronym for the header.

Requirement The text must be enclosed in quotation marks.

Tip The ACRONYM= attribute is valid only in the MARKUP destination.

ALT= "*text*" | *variable*;
specifies an alternate description of the header.

Requirement The text must be enclosed in quotation marks.

Tip The ALT= attribute is valid only in the MARKUP destination.

DEF_SPLIT;
specifies that special characters in headers for generic columns are to be used as split characters.

Tip The DEF_SPLIT attribute is valid in all destinations.

END=*column-name* | *variable*;
specifies the last column that a spanning header covers.

Default The last column

Tip The END= attribute is valid in all destinations except the OUTPUT destination.

See START= header attribute on page 611

EXPAND="*string*" | *variable*;
specifies a character to use to expand the header to fill the space over the column or columns that the header spans.

Default None

Interaction If you specify both the REPEAT=ON and EXPAND=ON attributes, then the EXPAND= attribute is used.

Tips If the string or the variable that you specify contains more than one character, then only the first character is used.

The EXPAND= attribute is valid only in the LISTING destination.

See REPEAT= header attribute on page 610

EXPAND_PAGE=

EXPAND_PAGE<= ON | OFF | *variable*>;
specifies whether to expand the header to reach the sides of the page.

Default OFF

Tip The EXPAND_PAGE attribute is valid only in the LISTING destination.

See EXPAND=

FIRST_PANEL<= ON | OFF | *variable*>;
specifies whether a spanning header appears only on the first data panel if the table is too wide to fit in the space that is provided.

Default OFF

Restriction Applies only to headers, not to footers

Tip The FIRST_PANEL attribute is valid in the LISTING, printer family, and RTF destinations.

See LAST_PANEL header attribute on page 608

FORCE<= ON | OFF | *variable*>;
specifies whether to try to expand the column width to accommodate the longest word in the column header. The column width can be anything between the values for the WIDTH= and WIDTH_MAX= column attributes.

Default ON

Tip The FORCE= attribute is valid only in the LISTING destination.

See WIDTH= column attribute on page 602 and WIDTH_MAX= column attribute on page 602.

GENERIC<= ON | OFF | *variable*> ;
specifies whether multiple columns can use the header.

Default OFF

Restriction This attribute is primarily for writers of SAS procedures and for DATA step programmers.

Tip The GENERIC= attribute is valid in all destinations except the OUTPUT destination.

JUST=*justification* | *variable*;
specifies the horizontal justification for the column header, where *justification* is one of the following:

LEFT
specifies left justification.

Alias L

RIGHT
specifies right justification.

Alias R

CENTER
specifies center justification.

Alias C

Default The justification for the column

Tip The JUST= attribute is valid in all destinations except the OUTPUT destination.

Example "Example 1: Editing a Table Template That a SAS Procedure Uses" on page 550

LAST_PANEL<= ON | OFF | *variable*> ;
specifies whether a table footer appears only on the last data panel if the table is too wide to fit in the space that is provided.

Default OFF

Restriction Applies only to footers, not to headers

Tip The LAST_PANEL= attribute is valid only in the LISTING, printer family, and RTF destinations.

See FIRST_PANEL= header attribute on page 607

LONGDESC= "*string*" | *variable*;
specifies the long description of the header.

Requirement The text must be enclosed within quotation marks.

Tip The LONGDESC= attribute is valid only in markup family destinations.

MAXIMIZE<=ON | OFF | *variable*> ;
specifies whether to try to divide the text equally among all lines or to maximize the amount of text in each line when the text in the header uses more than one line. For example, if the text spans three lines, MAXIMIZE=ON can result in 45% of the text on the first line, 45% of the text on the second line, and 10% of the text on the third line. MAXIMIZE=OFF can result in 33% of the text on each line. MAXIMIZE=ON can write lines of text that vary greatly in length. MAXIMIZE=OFF can result in using less than the full column width.

Default OFF

Tip The MAXIMIZE= attribute is valid only in the LISTING destination.

OVERLINE<=ON | OFF | *variable*>;
specifies whether to draw a continuous line above the header. The second formatting character is used to draw the line. See the discussion of “FORMCHAR= "*string*" | *variable*;” on page 621.

Default OFF

Tip The OVERLINE= attribute is valid only in the LISTING destination.

PARENT=*header-path*;
specifies the header template that the current template inherits from. A *header-path* consists of one or more names, separated by periods. Each name represents a directory in a template store. (A template store is a type of SAS file.) The current template inherits from the specified header template in the first readable template store in the current path.

When you specify a parent, all of the attributes and statements that are specified in the parent's template are used in the current template unless the current template specifically overrides them.

Tip The PARENT= attribute is valid in all destinations.

PREFORMATTED<=ON | OFF | *variable*>;
specifies whether to treat the text as preformatted text. When text is preformatted, ODS honors line breaks as well as leading, trailing, and internal spaces. It also displays the text in a monospace font.

Default OFF

Interactions When PREFORMATTED=ON, and you are defining a table header or a footer, ODS uses the HeaderFixed or the FooterFixed style element unless you specify another style element with the STYLE= column attribute.

When PREFORMATTED=ON, and you are defining a column header, ODS uses the RowHeaderFixed style element unless you specify another style element with the STYLE= column attribute.

Tip The PREFORMATTED attribute is valid in the markup family, printer family, and RTF destinations.

PRINT<=ON | OFF | *variable*>;
specifies whether to print the header.

Default ON

Tips When PRINT=ON, the column header becomes the label of the corresponding variable in any output data sets that the OUTPUT destination creates if neither the column template nor the data component provides a label.

The PRINT= attribute is valid in all destinations.

REPEAT<=ON | OFF | *variable*>;
specifies whether to repeat the text of the header until the space that is allotted for the header is filled.

Default OFF

Interaction If you specify both the REPEAT=ON and EXPAND=ON attributes, then the EXPAND= attribute is used.

Tip The REPEAT attribute is valid only in the LISTING destination.

See EXPAND= header attribute on page 607

SPACE=*positive-integer* | *variable*;
specifies the number of blank lines to place between the current header and the next header or between the current footer and the previous footer.

Default 0

Tips A row of underlining or overlining is considered a header or a footer.

The SPACE= attribute is valid only in the LISTING destination.

Example “Example 3: Creating a New Table Template ” on page 557

SPILL_ADJ<=ON | OFF | *variable*>;
specifies whether to extend the text of the header into the header space of adjacent columns.

Default OFF

Interaction FORCE=, SPILL_MARGIN=, SPILL_ADJ=, and TRUNCATE= are mutually exclusive. If you specify more than one of these attributes, then only one of these attributes are used. FORCE= takes precedence over the other three attributes, followed by SPILL_MARGIN=, SPILL_ADJ=, and TRUNCATE=.

Tip The SPILL_ADJ attribute is valid only in the LISTING destination.

See FORCE= header attribute on page 607, SPILL_MARGIN= on page 610, and TRUNCATE= header attribute on page 613.

SPILL_MARGIN<=ON | OFF | *variable*>;
specifies whether to extend the text of the header into the adjacent margin.

Default ON

Restriction SPILL_MARGIN= applies only to a spanning header that spans all the columns in a data panel.

Interaction The FORCE=, SPILL_MARGIN=, SPILL_ADJ=, and TRUNCATE= attributes are mutually exclusive. If you specify more than one of these attributes, then only one of these attributes are used. The FORCE= attribute takes precedence over the other three attributes, followed by SPILL_MARGIN=, SPILL_ADJ=, and TRUNCATE=.

Tip The SPILL_MARGIN attribute is valid only in the LISTING destination.

See FORCE= header attribute on page 607, SPILL_ADJ= header attribute on page 610, and the TRUNCATE= header attribute on page 613.

SPLIT= "*character*" | *variable*;
specifies the split character for the header. PROC TEMPLATE starts a new line when it reaches that character and continues the header on the next line. The split character itself is not part of the header although each occurrence of the split character counts toward the maximum length for a label.

The first character in a header defines the split character if it is not one of the following:

- an alphanumeric character
- a blank
- an underscore (_)
- a hyphen (-)
- a period (.)
- a percent (%)

Note: The split will not occur until the split character appears after text.

Tip The SPLIT= attribute is valid in all destinations except the OUTPUT destination.

START=*column-name* | *variable*;
specifies the first column that a spanning header covers.

Default The first column

Tip The START= attribute is valid in all destinations except the OUTPUT destination.

STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>;
specifies the style element and any changes to its attributes to use for the header.

style-element-name
is the name of the style element to use to produce the header. The style element must be part of a style that is registered with the Output Delivery System. SAS provides some styles. You can create customized styles by using PROC TEMPLATE (see "DEFINE STYLE Statement" on page 454).

By default, ODS produces different parts of ODS output with different elements. For example, by default, a table header is displayed with the style element Header. The style elements that you would be most likely to use with the STYLE= attribute for a table header are as follows:

- Header
- HeaderFixed
- HeaderEmpty
- HeaderEmphasis
- HeaderEmphasisFixed
- HeaderStrong
- HeaderStrongFixed

The style elements that you would be most likely to use with the STYLE= attribute for a table footer are as follows:

- Footer
- FooterFixed
- FooterEmpty
- FooterEmphasis
- FooterEmphasisFixed
- FooterStrong
- FooterStrongFixed

The style elements that you would be most likely to use with the STYLE= attribute for a column header are as follows:

- Rowheader
- RowheaderFixed
- RowheaderEmpty
- RowheaderEmphasis
- RowheaderEmphasisFixed
- RowheaderStrong
- RowheaderStrongFixed

The style element provides the basis for displaying the header. Additional style attributes that you provide can modify the display.

style-element-name is either the name of a style element or a variable whose value is a style element.

Default Header

See “Working with Styles ” on page 438

“Finding and Viewing the Default Style for ODS Destinations” on page 438

For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

style-attribute-specification
describes the style attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

Requirement The STYLE= option requires either a *style-attribute-specification* or a *style-element-name*.

Tips You can use braces ({ and }) instead of square brackets ([and]).

If you use the STYLE= attribute inside a quoted string, then add a space before or after the carriage return to prevent errors. SAS does not interpret a carriage return as a space. You must explicitly specify spaces in quoted strings.

The STYLE= attribute is valid only in the markup family, printer family, and RTF destinations.

See Chapter 21, “Style Attributes,” on page 807

Examples “Example 1: Editing a Table Template That a SAS Procedure Uses” on page 550

“Example 3: Creating a New Table Template ” on page 557

TRUNCATE<=ON | OFF | *variable*>;
specifies whether to start a new header line in the middle of a word.

ON
starts a new line of the header when the text fills the specified column width.

OFF
extends the width of the column to accommodate the longest word in the column header, if possible. TRUNCATE=OFF is the same as FORCE=ON.

Default OFF

Interaction If you specify FORCE=, SPILL_MARGIN=, or SPILL_ADJ=, then the TRUNCATE= attribute is ignored.

Tip The TRUNCATE= attribute is valid only in the LISTING destination.

See FORCE= header attribute on page 607, SPILL_MARGIN= header attribute on page 610, and SPILL_ADJ header attribute on page 610.

UNDERLINE<=ON | OFF | *variable*> ;
specifies whether to draw a continuous line below the header. The second formatting character is used to draw the line.

Default OFF

Tip The UNDERLINE attribute is valid only in the LISTING destination.

See For information about formatting characters, see the discussion of FORMCHAR= table attribute on page 621.

VJUST=*justification* | *variable*;
Specifies vertical justification for the header. *justification* is one of the following:

TOP
places the header as high as possible.

Alias T

CENTER
centers the header vertically.

Alias C

BOTTOM
places the header as low as possible.

Alias B

Default BOTTOM

Tip The VJUST= attribute is valid only in the MARKUP and PRINTER families of destinations.

WIDTH=*positive-integer* | *variable*;
specifies the width of the header in characters.

Default If you omit a width, the column width is used.

Tips To create a vertical header, specify a width of 1.

The WIDTH= attribute is valid only in the LISTING destination.

Table Attributes

This section lists all the attributes that you can use in a table template. Table attributes are used to customize the attributes of a table and must be specified within a DEFINE TABLE statement block. You can specify multiple table attributes together or separately. For example, you can specify the following table attributes together:

```
order_data=yes use_format_defaults=yes print_headers=off;
```

or separately:

```
order_data=yes;
use_format_defaults=yes;
print_headers=off;
```

For all attributes that support a value of ON, these forms are equivalent:

```
ATTRIBUTE-NAME;
ATTRIBUTE-NAME=ON;
```

For all of the attributes that support a value of *variable*, *variable* is any variable that you declare in the table template with the DYNAMIC, MVAR, or NMVAR statement. If the attribute is a Boolean, then the value of *variable* should resolve to either true or false as shown in this table:

Table 16.5 Boolean Values

True	False
ON	OFF
_ON_	_OFF_
1	0
TRUE	FALSE
YES	NO
YES	_NO_

Table 16.6 Table Attributes

Task	Attribute	Destinations
Influence the layout of the table		
Specify whether to try to place the same number of columns in each data panel if the entire table does not fit in one data panel	BALANCE on page 618	LISTING, printer family, and RTF
Specify whether to center each data panel independently if the entire table does not fit in one data panel	CENTER on page 619	LISTING, printer family, RTF
Specify whether to force a new page before printing the table	NEWPAGE on page 622	All except OUTPUT
Specify the number of sets of columns to place on a page	PANELS= on page 623	LISTING and printer family
Specify the number of blank characters to place between sets of columns when PANELS= is in effect	PANEL_SPACE= on page 623	LISTING
Specify the number of lines that must be available on the page in order to print the body of the table	REQUIRED_SPACE= on page 624	LISTING and printer family
Specify the number of lines to place between the previous output object and the current one	TOP_SPACE= on page 625	LISTING and printer family
Specify whether to split a table that is too wide to fit in the space that is	WRAP on page 626	LISTING and printer family

Task	Attribute	Destinations
provided or to wrap each row of the table		
Specify whether to add a double space after the last line of a single row when the row is wrapped	WRAP_SPACE on page 626	LISTING and printer family
Influence the layout of rows and columns		
Specify the maximum number of blank characters to place between columns	COL_SPACE_MAX= on page 619	LISTING
Specify the minimum number of blank characters to place between columns	COL_SPACE_MIN= on page 619	LISTING
Specify the name of the column whose value provides formatting information about the space before each row of the template	CONTROL= on page 620	All except OUTPUT
Specify whether to double space between the rows of the table	DOUBLE_SPACE on page 620	LISTING
Specify whether extra space is evenly divided among all columns of the table	EVEN on page 621	LISTING
Specify whether to split a long stacked column across page boundaries	SPLIT_STACK on page 624	LISTING
Influence the display of the values in header cells and data cells		
Specify whether to suppress blanking the value in a column that is marked with the BLANK_DUPS column attribute if the value changes in a previous column that is also marked with the BLANK_DUPS attribute	CLASSLEVELS= on page 619	LISTING and printer family
Specify which format to use if both a column template and a data component specify a format	DATA_FORMAT_OVERRIDE on page 620	All
Specify whether to justify the format fields within the columns or to justify	JUSTIFY on page 621	LISTING

Task	Attribute	Destinations
the values within the columns without regard to the format fields		
Specify whether to order the columns by their order in the data component	ORDER_DATA on page 622	All except OUTPUT
Specify the source of the values for the format width and the decimal width if they are not specified	USE_FORMAT_DEFAULTS on page 626	All
Use the column name as the column header if neither the column template nor the data component specifies a header	USE_NAME on page 626	All
Influence the layout of headers and footers		
Specify the number of blank lines to place between the last row of data and the first row of output	FOOTER_SPACE= on page 621	LISTING
Specify the number of blank lines to place between the last row of headers and the first row of data	HEADER_SPACE= on page 621	LISTING
Specify whether to draw a continuous line above the first table footer or, if there is no table footer, below the last row of data on a page	OVERLINE on page 622	LISTING
Specify whether to print table footers and any overlining of the table footers	PRINT_FOOTERS on page 623	All except OUTPUT
Specify whether to print table headers and any underlining of the table headers	PRINT_HEADERS on page 623	All except OUTPUT
Specify whether to draw a continuous line under the last table header or, if there is no table header, then above the last row of data on a page	UNDERLINE on page 625	LISTING
Influence the non-LISTING output		

Task	Attribute	Destinations
Specify whether to place the output object in a table of contents, if you create a table of contents	CONTENTS on page 619	HTML
Specify the label to use for the output object in the contents file, the Results window, and the trace record	CONTENTS_LABEL= on page 619	HTML, PDF, PRINTER, PS PDFMARK
Other table attributes		
Control whether BY lines are printed above each BY group	BYLINE= on page 618	All except OUTPUT
Define the characters to use as the line-drawing characters in the table	FORMCHAR= on page 621	LISTING
Specify a label for the table	LABEL= on page 622	All
Specify the table that the current template inherits from	PARENT= on page 623	All
Specify the style element to use for the table and any changes to the attributes	"STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>;" on page 624STYLE=	Markup family, printer family, and RTF
Specify the special data set type of a SAS data set	"TYPE=*string* | *variable*;" on page 625TYPE=	OUTPUT

BALANCE <=ON | OFF | *variable*>;
specifies whether to try to place the same number of columns in each data panel if the entire table does not fit in one data panel.

Default OFF

Tip The BALANCE attribute is valid only in the LISTING, printer family, and RTF

BYLINE <=ON | OFF | *variable*>;
controls whether BY lines are printed above each BY group in a configuration file, SAS invocation, OPTIONS statement, or Systems Options window.

Category PROC OPTIONS GROUP= LISTCONTROL

Default OFF

Restriction This attribute applies only if the table is not the first one on the page. If BY-group processing is in effect, a byline automatically precedes the first table on the page.

Tip The BYLINE attribute is valid in all destinations except the OUTPUT destination.

CENTER <=ON | OFF | *variable*>;
specifies whether to center each data panel independently if the entire table does not fit in the space that is provided.

Default ON

Tip The CENTER attribute is valid only in the LISTING, printer family, and RTF destinations.

CLASSLEVELS <=ON | OFF | *variable*>;
specifies whether to suppress blanking the value in a column that is marked with the BLANK_DUPS column attribute if the value changes in a previous column that is also marked with the BLANK_DUPS attribute.

Default OFF

Tip The CLASSLEVELS attribute is valid for all destinations except the OUTPUT destination.

Example “Example 1: Creating a Stand-Alone Style” on page 466

COL_SPACE_MAX=*positive-integer* | *variable*;
specifies the maximum number of blank characters to place between the columns.

Default 4

Tip The COL_SPACE_MAX= table attribute is valid only in the LISTING destination.

COL_SPACE_MIN=*positive-integer* | *variable*;
specifies the minimum number of blank characters to place between the columns.

Default 2

Tip The COL_SPACE_MIN= attribute is valid only in the LISTING destination.

CONTENTS <=ON | OFF | *variable*>;
specifies whether to place the output object in a table of contents, if you create a table of contents.

Default ON

Tip The CONTENTS attribute is valid in markup family and printer family destinations.

CONTENTS_LABEL= "*string*" | *variable*;
specifies the label to use for the output object in the contents file, the Results window, and the trace record.

Default If the SAS system option LABEL is in effect, the default label is the object's label. If LABEL is not in effect, the default label is the object's name.

Tip The CONTENTS_LABEL= attribute is valid only in markup family and printer family destinations.

CONTROL=*column-name* | *variable*;
specifies the name of the column whose values provide formatting information about the space before each row of the template. The value of CONTROL= should be the name of a column of type character with a length equal to 1.

Table 16.7 *Values in the Control Column*

Column Control Value	Result
A digit from 1-9	The specified number of blank lines precedes the current row.
A hyphen (-)	A row of underlining precedes the current row.
"b" or "B"	ODS tries to insert a panel break if the entire table does not fit in the space that is provided.

Default None

Restriction The "b" and "B" column control values are not supported for the PRINTER destination.

Tip The CONTROL= attribute is valid in all destinations except the OUTPUT destination.

DATA_FORMAT_OVERRIDE<=ON | OFF | *variable*>;
specifies which format to use if both a column template and a data component specify a format.

ON
uses the format that the data component specifies.

OFF
use the format that the column template specifies.

Default OFF

Tip The DATA_FORMAT_OVERRIDE attribute is valid in all destinations.

DOUBLE_SPACE<=ON | OFF | *variable*>;
specifies whether to double space between the rows of the table.

Default OFF

Tip The DOUBLE_SPACE attribute is valid only in the LISTING destination.

Examples “Example 1: Editing a Table Template That a SAS Procedure Uses” on page 550

“Example 3: Creating a New Table Template ” on page 557

EVEN<=ON | OFF | *variable*>;
specifies whether extra space is evenly divided among all columns of the table.

Default OFF

Tip The EVEN attribute is valid only in the LISTING destination.LISTING

FOOTER_SPACE=0 | 1 | 2 | *variable*;
specifies the number of blank lines to place between the last row of data and the first row of the table footer.

Default 1

Tip The FOOTER_SPACE= attribute is valid only in the LISTING destination.

FORMCHAR= "*string*" | *variable*;
defines the characters to use as the line-drawing characters in the table. Currently, ODS uses only the second of the 20 possible formatting characters. This formatting character is used for underlining and overlining. To change the second formatting character, specify both the first and second formatting characters. For example, this option assigns the asterisk (*) to the first formatting character, the plus sign (+) to the second character, and does not alter the remaining characters: `formchar="*+"`

Default The SAS system option FORMCHAR= specifies the default formatting characters.

Tips Use any character in formatting-characters, including hexadecimal characters. If you use hexadecimal characters, then put an x after the closing quotation mark. For example, this option assigns the hexadecimal character 2-D to the first formatting character, the hexadecimal character 7C to the second character, and does not alter the remaining characters: `formchar="2D7C"x`

The FORMCHAR= attribute is valid only in the LISTING destination.

HEADER_SPACE=0 | 1 | 2 | *variable*;
specifies the number of blank lines to place between the last row of headers and the first row of data. A row of underscores is a header.

Default 1

Tip The HEADER_SPACE= attribute is valid only in the LISTING destination.

JUSTIFY<=ON | OFF | *variable*>;
specifies whether to justify the format fields within the columns or to justify the values within the columns without regard to the format fields.

Default OFF

Interactions JUSTIFY=ON can interfere with decimal alignment.

If the column is numeric, then values are aligned to the right if you specify JUSTIFY=OFF and JUST=C.

All of the destinations except for the LISTING destination justify the values in columns as if JUSTIFY=ON for JUST=R and JUST=L.

Tips If you translate numeric data to character data, you might need to use JUSTIFY= to align the data.

The JUSTIFY attribute is valid only in the LISTING destination.

See "Values in Table Columns and How They Are Justified" on page 547

LABEL= "*text*" | *variable*;
specifies a label for the table.

Default ODS uses the first of the following that it finds: a label that the table template provides, a label that the data component provides, or the first spanning header in the table.

Tip The LABEL= attribute is valid in all destinations.

NEWPAGE<=ON | OFF | *variable*>;
specifies whether to force a new page before printing the table.

Default OFF

Restriction If the table is the first item on the page, ODS ignores this attribute.

Tip The NEWPAGE attribute is valid in all destinations except the OUTPUT destination.

ORDER_DATA<=ON | OFF | *variable*>;
specifies whether to order the columns by their order in the data component.

Defaults OFF

When ORDER_DATA=OFF, the default order for columns is the order that they are specified in the COLUMN statement. If you omit a COLUMN statement, the default order for columns is the order in which you define them in the template.

Interaction ORDER_DATA is most useful for ordering generic columns.

Tip The ORDER_DATA attribute is valid in all destinations except the OUTPUT destination. The OUTPUT destination always uses the order of the columns in the data component when it creates an output data set.

OVERLINE<=ON | OFF | *variable*>;
specifies whether to draw a continuous line above the first table footer or, if there is no table footer, below the last row of data on a page. The second formatting character is used to draw the line.

Default OFF

Tip The OVERLINE attribute is valid only in the LISTING destination.

See For information about formatting characters, see the discussion of "FORMCHAR= "*string*" | *variable*;" on page 621.

UNDERLINE= table attribute on page 625, UNDERLINE= column attribute on page 601, and the OVERLINE= column attribute on page 597.

Example "Example 1: Editing a Table Template That a SAS Procedure Uses" on page 550

PANELS=*positive-integer* | *variable*;
specifies the number of sets of columns to place on a page. If the width of all the columns is less than half of the line size, display the data in multiple sets of columns so that rows that would otherwise appear on multiple pages appear on the same page.

Tips If the number of panels that is specified is larger than the number of panels that can fit on the page, the template creates as many panels as it can. Let the table template put data in the maximum number of panels that can fit on the page by specifying a large number of panels (for example, 99).

The PANELS= attribute is valid only in LISTING and printer family destinations.

PANEL_SPACE=*positive-integer* | *variable*;
specifies the number of blank characters to place between sets of columns when PANELS= is in effect.

Default 2

Tip The PANEL_SPACE= attribute is valid only in the LISTING destination.

PARENT=*table-path*;
specifies the table that the current template inherits from. A *table-path* consists of one or more names, separated by periods. Each name represents a directory in a template store. (A template store is a type of SAS file.) The current template inherits from the specified table in the first template store in the current path that you can read from.

When you specify a parent, all of the attributes and statements that are specified in the parent's template are used in the current template unless the current template overrides them.

Tip The PARENT= attribute is valid in all destinations.

PRINT_FOOTERS<=ON | OFF | *variable*>;
specifies whether to print table footers and any overlining of the table footers.

Default ON

Tip The PRINT_FOOTERS attribute is valid in all destinations except the OUTPUT destination.

See OVERLINE= table attribute on page 622

PRINT_HEADERS<=ON | OFF | *variable*>;
specifies whether to print the table headers and any underlining of the table headers.

Default ON

Interaction When used in a table template, PRINT_HEADERS affects only headers for the table, not the headers for individual columns. For individual columns, use the following column attribute: “PRINT_HEADERS<=ON | OFF | *variable*>;” on page 599.

Tip The PRINT_HEADERS attribute is valid in all destinations except the OUTPUT destination.

See “UNDERLINE<=ON | OFF | *variable*>;” on page 625

REQUIRED_SPACE=*positive-integer* | *variable*;
specifies the number of lines that must be available on the page in order to print the body of the table. The body of the table is the part of the table that contains the data. It does not include headers and footers.

Default 3

Tip The REQUIRED_SPACE= attribute is valid in LISTING and printer family destinations.

SPLIT_STACK<=ON | OFF | *variable*>;
specifies whether to split a long stacked column across page boundaries.

Default OFF

Tip The SPLIT_STACK attribute is valid only in the LISTING destinations.

STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>;
specifies the style element and any changes to its attributes to use for the table.

style-element-name
is the name of the style element to use to display the table. The style element must be part of a style that is registered with the Output Delivery System. SAS provides some styles. You can create customized styles with PROC TEMPLATE (see “DEFINE STYLE Statement” on page 454). By default, ODS produces different parts of ODS output with different elements. For example, by default, a table is produced with the style element Table. The styles that SAS provides do not provide another style element that you would be likely to want to use instead of Table. However, you might have a user-defined style element at your site that would be appropriate to specify.

The style element provides the basis for displaying the table. Additional style attributes that you provide can modify the display.

style-element-name is either the name of a style element or a variable whose value is a style element.

See “Working with Styles ” on page 438

“Finding and Viewing the Default Style for ODS Destinations” on page 438

For a table of style element names, see Chapter 20, “Style Elements,” on page 777.

style-attribute-specification
describes the style attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

See "About Style Attributes" on page 465

Default Table

Requirement Specify either a *style-attribute-specification* or a *style-element-name* with the STYLE= option.

Tips You can use braces ({ and }) instead of square brackets ([and]).

If you use the STYLE= attribute inside a quoted string, then add a space before or after the carriage return to prevent errors. SAS does not interpret a carriage return as a space. You must explicitly specify spaces in quoted strings.

The STYLE= attribute is valid only in the markup family, printer family, and RTF destinations.

TOP_SPACE=*positive-integer* | *variable* ;
specifies the number of lines to place between the previous output object and the current one.

Default 1

Tip The TOP_SPACE= attribute is valid only in LISTING and printer family destinations.

TYPE=*string* | *variable*;
specifies special type of SAS data set.

Restriction PROC TEMPLATE does *not* verify that a SAS data set type that you specify is a valid data set type or the structure of the data set that you create is appropriate for the type that you have assigned.

Tips Most SAS data sets have no special type. However, certain SAS procedures, like the CORR procedure, can create a number of special SAS data sets. In addition, SAS/STAT software and SAS/EIS software support special data set types.

The TYPE= attribute is valid only in the OUTPUT destination.

UNDERLINE<=ON | OFF | *variable*>;
specifies whether to draw a continuous line under the last table header (or, if there is no table header, then above the first row of data on a page). The second formatting character is used to draw the line.

Default OFF

Tip The UNDERLINE attribute is valid only in the LISTING destination.

See For information about formatting characters, see "FORMCHAR= "*string*" | *variable*;" on page 621.

UNDERLINE= column attribute on page 601 and OVERLINE= column attribute on page 597.

Also see OVERLINE table attribute on page 622.

Examples "Example 1: Editing a Table Template That a SAS Procedure Uses" on page 550

"Example 3: Creating a New Table Template " on page 557

USE_FORMAT_DEFAULTS<=ON | OFF | *variable*>;
specifies the source of the values for the format width and the decimal width if they are not specified.

ON
uses the default values, if any, that are associated with the format name.

OFF
uses the PROC TEMPLATE defaults.

Default OFF

Tip The USE_FORMAT_DEFAULTS attribute is valid in all destinations except the OUTPUT destination.

USE_NAME<=ON | OFF | *variable*>;
uses the column name as the column header if neither the column template nor the data component specifies a header.

Default OFF

Tips Use this attribute when column names are derived from a data set and the columns are generic.

The USE_NAME attribute is valid in all destinations except the OUTPUT destination.

WRAP<=ON | OFF | *variable*>;
specifies whether to split a wide table into multiple data panels, or to wrap each row of the table so that an entire row is printed before the next row starts.

Default OFF

Interaction When ODS wraps the rows of a table, it does not place multiple values in any column that contains an ID column.

Tip The WRAP attribute is valid only in LISTING and printer family destinations.

See WRAP_SPACE table attribute on page 626 and ID= column attribute on page 595

WRAP_SPACE<=ON | OFF | *variable*>
specifies whether to double space after the last line of a single row of the table when the row is wrapped onto more than one line.

Default OFF

Tip The WRAP_SPACE attribute is valid only in the LISTING, printer family, and RTF destinations.

See WRAP= table attribute on page 626

Chapter 17

TEMPLATE Procedure: Creating Markup Language Tagsets

Overview: TEMPLATE Procedure: Creating Markup Language Tagsets ***627***
Overview: ODS Tagsets and the TEMPLATE PROCEDURE 628

Concepts: TEMPLATE Procedure: Creating Markup Language Tagsets ***629***
Getting Familiar with Tagsets 629
Creating Custom Tagsets 633

Syntax: TEMPLATE Procedure: Creating Markup Language Tagsets ***637***
DEFINE TAGSET Statement 640
DEFINE EVENT Statement 648
BLOCK Statement 650
BREAK Statement 651
CLOSE Statement 651
CONTINUE Statement 652
DELSTREAM Statement 652
DO Statement 653
DONE Statement 653
ELSE Statement 654
END Statement 654
EVAL Statement 654
FLUSH Statement 657
ITERATE Statement 657
NDENT Statement 658
NEXT Statement 659
NOTES Statement 660
OPEN Statement 660
PUT Statement 661
PUTL Statement 663
PUTLOG Statement 664
PUTQ Statement 666
PUTSTREAM Statement 667
PUTVARS Statement 668
SET Statement 670
STOP Statement 675
TRIGGER Statement 675
UNBLOCK Statement 676

UNSET Statement 677
XDENT Statement 679

Usage: TEMPLATE Procedure: Creating Markup Language Tagsets ***680***
Event Variables 680
Event Statement Conditions 687

Examples: TEMPLATE Procedure: Creating Markup Language Tagsets ***689***
Example 1: Creating a Tagset through Inheritance 689
Example 2: Creating a Tagset By Copying a Tagset's Source 692
Example 3: Creating a New Tagset 698
Example 4: Executing Events Using the TRIGGER= Statement 703
Example 5: Indenting Output 704
Example 6: Using Different Styles for Events 706
Example 7: Modifying an Event to Include Other Style Sheets 708
Example 8: Using the STACKED_COLUMNS Attribute in a Tagset 708

Overview: TEMPLATE Procedure: Creating Markup Language Tagsets

Overview: ODS Tagsets and the TEMPLATE PROCEDURE

The TEMPLATE procedure enables you to create a tagset A tagset is a type of template that defines how to generate a markup language output type from SAS output. You can specify a tagset to create markup language output from ODS. SAS provides tagsets for a variety of markup language output. For example, SAS provides several tagsets for XML output, HTML output, XSL, and more. The TEMPLATE procedure enables you to modify any of the SAS tagsets or create custom markup language tagsets.

The Output Delivery System uses the specified tagsets to mark the SAS output, which you can view with an online browser or viewer.

For information about terms used in the TEMPLATE procedure, see “Using the TEMPLATE Procedure” on page 329.

Concepts: TEMPLATE Procedure: Creating Markup Language Tagsets

Getting Familiar with Tagsets

Listing Tagset Names

SAS provides a set of tagsets. To get a list of the tagsets that SAS supplies and any tagsets that you have created and stored in the Sashelp.Tmplmst template store, submit the following SAS statements:

```
proc template;
   list tagsets;
run;
```

By default, PROC TEMPLATE lists the tagsets in Sashelp.Tmplmst and Work.Templat. Typically, you have Read-Only permissions to the Sashelp.Tmplmst item store where the SAS tagset directory is located. The Work.Templat is the item store where the tagsets that you create or customize are stored by default.

Specifying Tagset Names

To specify a SAS tagset stored in Sashelp.Tmplmst or a tagset that you have created and stored in Work.Templat or any other item store, use a two-level name: Tagsets.*tagset-name*. For example, Tagsets.Chtml or Tagsets.Mytagset are valid two-level tagset names. By default, SAS knows that the specified tagset is stored in either Sashelp.Tmplmst or Work.Templat.

To specify a tagset that you have created and stored in an item store other than Work.Templat, assign the item store to the ODS search path with the ODS PATH statement. For information about the ODS PATH statement, see “ODS PATH Statement” in *SAS Output Delivery System: User’s Guide*.

Viewing the Contents of a Tagset

To see the source for a tagset, use PROC TEMPLATE and specify the two-level name of the tagset. For example, to see the source of a SAS tagset that generates CHTML output, submit these SAS statements:

```
proc template;
      source tagsets.chtml;
```

The source for Tagsets.Chtml consists of the following:

- a DEFINE TAGSET statement that names the tagset
- events that define what is written to the output file
- tagset attributes, such as output type and the character to use for line breaks

Understanding Events

A tagset controls output generation through a series of events and variables. An event defines what is written to the output file. Here are some key points about events:

- Events have unique names. SAS procedures that generate ODS output use a standard set of events, which you can customize by redefining them in the customized tagset. In addition, you can define custom events.
- The DEFINE EVENT statement assigns a name to an event.
- An event can include start sections, finish sections, or both. These sections specify different actions. If the event does not include either a start or finish section, then the event is stateless: no matter how the event is called, all of the actions in the event are executed. If an event has a finish section, then a start section is assumed if there are statements above the finish section.
- An event can execute another event using the TRIGGER statement. From the start section of an event, any event triggered also runs its start section. From the finish section, the triggered event runs its finish section. If a triggered event does not have start or finish sections, then the event runs the statements that it does have. A trigger can also explicitly ask for an event's specific section. See “Example 4: Executing Events Using the TRIGGER= Statement” on page 703.
- Events can perform actions based on conditions.
- An event consists of PUT statements, text, and event variables.

For example, here is a simple event for an HTML table output:

```
define event table; 1
start: 2
      put '<table>' nl;
finish:
      put '</table>' nl;
end;
```

In the event:

1. The DEFINE EVENT statement begins the event and assigns it the name TABLE.
2. The START section defines the beginning portion of the event, and the FINISH section defines the ending portion of the event. An event for a table needs START and FINISH sections because ODS needs to know how to define the beginning and the ending. ODS also expects other events to define how to format the table's rows and columns. The PUT statements specify to write the tags `<table>` and `</table>` to the output file, and to add a new line after each tag.

The following event does not include a start and finish section. The PUT statements specify to write the tags `<TD>` and `</TD>` to the output file. In addition, the event variable VALUE is used so that the data value from the SAS procedure or data set is written to the output file. The data value is enclosed with the `<TD>` and `</TD>` tags.

```
define event data;
      put '<TD>';
      put VALUE;
      put '</TD>';
end;
```

Understanding Variables

A variable is a programming structure that is used to hold data. A variable holds the data that is assigned to it until you assign a new value or end the program. Each variable has a unique name and holds information that is either internal information to handle the requested output (metadata that is used by ODS or the XML LIBNAME engine) or is information that is directly related to the output itself. For example, the variable COLCOUNT holds the value for the number of columns in the output, and the variable DATE holds the date.

Variables that are used by tagsets are divided into two groups: internally generated and user-created.

There are three logical divisions of internally generated variables:

event variables
: are variables that include text, formatting, and data values. These variables can originate in many places, such as the table template, the procedure, the title, or byline processing.

style variables
: are variables that specify a value for one aspect of the presentation. Style variables are specified by the ODS style attributes that are currently in use. The style variables are only differentiated from other event variables in that you know exactly where they originate.For more information about style attributes, see Chapter 14, “TEMPLATE Procedure,” on page 433.

dynamic variables
: are variables that are dynamically created within SAS. Because these variables are dynamically created, their names, or how they are used, are unknown. These variables are dynamic because they are not defined by ODS, but by applications such as SAS/GRAPH and the XML LIBNAME engine. Dynamic variables are designated by a preceding @ symbol. Dynamic variables are listed

with the DYNAMIC statement. For more information about SAS/GRAPH, see *SAS/GRAPH: Reference*.

There are five types of user-created variables:

dictionary variables
: are arrays that contain a list of numbers or text strings that are identified by a key. A dictionary variable has, as part of its name, a preceding '$' symbol and a subscript that contains a text string or a variable that has a character value. The text string or variable within the subscript is called a key. Keys are case preserving and case sensitive. After dictionary variables are created, they are globally available in all events and persist until you unset them with the UNSET statement.

  For example, the following dictionary variable is identifying the entry in the $MyDictionary variable that contains the text 'dog': $MyDictionary['dog']. In this example, the key is 'dog'. Dictionary variables are accessed sequentially by using the ITERATE and NEXT statements.

list variables
: are arrays that contain a list of numbers or text strings that are indexed. A list variable has, as part of its name, a preceding '$' symbol and a subscript that is empty or contains a number or numeric variable. The number within the subscript is called an index. After they are created, list variables are globally available in all events and persist until you unset them with the UNSET statement.

  List entries are accessed by positive or negative indexes. Positive indexes start at the beginning of a list. Negative indexes start at the end of a list. For example, the list variable $Mylist[2] identifies the second entry in the list variable $Mylist. In this case, the index is 2. The list variable $Mylist[-2] identifies the second entry from the end of the list variable $Mylist. In this case, the index is [-2]. List variables are accessed sequentially by using the ITERATE and NEXT statements.

macro variables
: are variables that are part of the SAS macro programming language. Macro variables must be specified with the MVAR or NMVAR statements. After they are declared, macro variables can be used anywhere within an event. For more information. see the “MVAR Statement” on page 539 and “NMVAR Statement” on page 540.

memory variables
: are areas of memory that contain numeric data, character data, or lists of numeric or character data. A memory variable is classified as a dictionary variable if it is created with a subscript that contains a key. A memory variable is classified as a list variable if it is created with a subscript that is empty or contains an index. If you omit a key or an index, then the memory variable is a numeric or character scalar variable, depending on the variable's value.

scalar variables
: are areas of memory that contain numeric or character data. Scalar variables must be preceded by the '$' symbol. After scalar variables are created, they are globally available in all events and persist until you unset them with the UNSET statement.

stream variables
: are temporary item stores that contain output. All output from PUT statements is directed to the open stream variable until it is closed. Stream variables must be preceded by the '$$' symbol except when used with the OPEN or PUTSTREAM statements. Stream variables are created with the SET, EVAL, or OPEN

statements, within the DEFINE EVENT statement. Stream variables are different from other variables in that they can hold very large amounts of data. They can hold very large amounts of data because as they increase in size, they are written to disk as needed.

Displaying Event Variables and Their Values

Because variables represent data, their values might or might not be present, depending on the SAS procedure and the job. For example, some variables have values only if they are specified with procedure options or style options. Other variables have values because the internal information, such as how many columns are in the output, is needed. For example, Tagsets.Chtml contains the event COLSPECS, which uses the event variable COLCOUNT so that ODS knows how many columns are in the output:

```
define event colspecs;
      put '<p>' nl '<table';
      putq ' columns=' COLCOUNT;
      put ' cellpadding=2 border=1>' nl;
end;
```

To determine which variables have values and what the values are, use the EVENT_MAP statement to submit the SAS program. For more information, see “ Defining a Tagset Using the Event_Map Tagset” on page 634. For a list of event variables and their descriptions, see “Event Variables” on page 680.

Creating Custom Tagsets

Methods for Creating Custom Tagsets

To create a tagset, use the TEMPLATE procedure to define the tagset. In general, three methods are available to create a custom tagset:

- Define a tagset through inheritance.
- Copy an existing tagset, and then modify it.
- Define a custom tagset.

Inheriting Events in a Tagset

Tagsets can inherit events from each other. For example, the SAS tagset Tagsets.Csvall t inherits most of its events from Tagsets.Csv, and

Tagsets.MSOffice2k gets most of its events from Tagsets.Html4. Inheriting events from an existing tagset is the easiest way to define a new tagset.

To inherit events, a tagset uses the PARENT= attribute in the DEFINE TAGSET statement to specify the name of a tagset from which to inherit. When a parent is specified for a tagset, all of the tagset options, attributes, and statements that are specified in the parent's template are used in the new template, unless the new template overrides them. That is, in the new tagset, an event can override the operation of the same-named event that is defined in the parent tagset. For example, if the parent tagset defines an event named TABLE, then you can change the operation in the new tagset by redefining the event named TABLE.

For an example of inheriting events in a tagset, see "Example 1: Creating a Tagset through Inheritance" on page 689.

Defining a Tagset Using the Event_Map Tagset

SAS procedures that generate ODS output use a standard set of events and variables. To generate customized output, create a customized tagset with customized events. However, in order to customize the events, you need to know the names of the events that ODS uses.

A good way to start defining the customized tagset is to use the Event_Map tagset that SAS supplies. This enables you to determine which events are triggered and which variables are used by an event to send output from a SAS process to an output file. When you run a SAS process with Tagsets.Event_Map, ODS writes XML markup to an output file that shows all event names and variable names as tags. In the output, tag names are the event names. Tag attributes are the variables that have values for those events.

For example, the following statements run ODS MARKUP with TYPE=Event_Map to see which events and variables ODS uses for various parts of the PROC PRINT output:

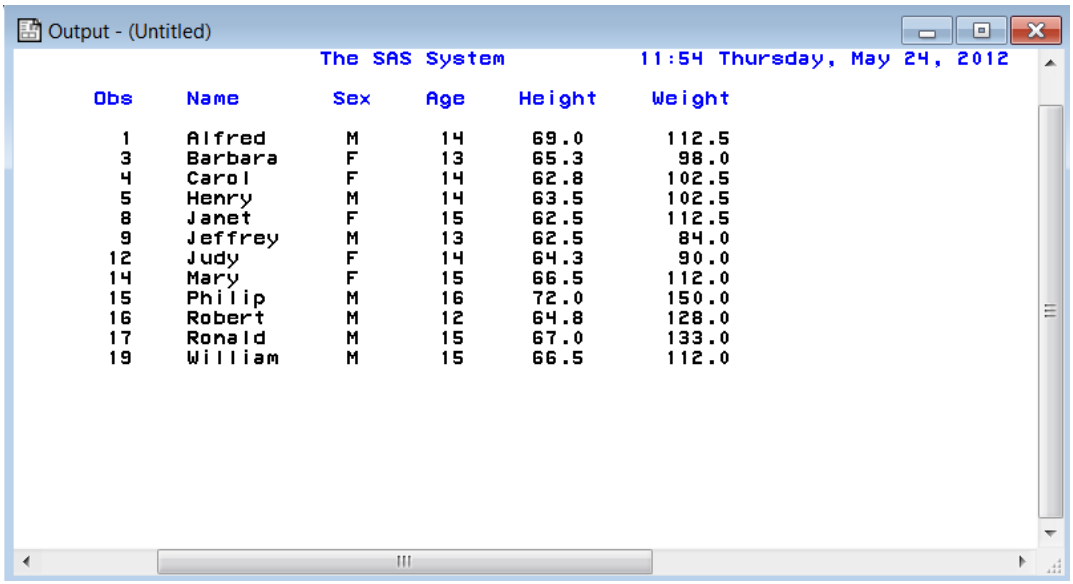
```
ods listing;
ods markup type=event_map file='custom-tagset-filename.xml';

proc print data=sashelp.class;
   where Height gt 60;
run;

ods markup close;
ods listing close;
```

Here is the listing output and resulting XML file:

Output 17.1 *LISTING Output*



Output 17.2 *XML Output*

Results Viewer

```
<?xml version="1.0" encoding="windows-1252" ?>
- <doc operator="juspen" sasversion="9.3" saslongversion="9.03.01M0P06122011" date="2012-05-24"
    time="11:54:21" encoding="windows-1252" event_name="doc" trigger_name="attr_out" class="body"
    just="l">
  - <doc_head event_name="doc_head" trigger_name="attr_out" class="body" just="l">
      <doc_meta event_name="doc_meta" trigger_name="attr_out" class="body" just="l" />
      <auth_oper event_name="auth_oper" trigger_name="attr_out" class="body" just="l" />
      <doc_title event_name="doc_title" trigger_name="attr_out" class="body" just="l" />
      <stylesheet_link event_name="stylesheet_link" trigger_name="attr_out" just="c" />
    - <javascript event_name="javascript" trigger_name="attr_out" class="body" just="l">
        <startup_function event_name="startup_function" trigger_name="attr_out"
          class="startupfunction" just="l" />
        <shutdown_function event_name="shutdown_function" trigger_name="attr_out"
          class="shutdownfunction" just="l" />
      </javascript>
    </doc_head>
  - <doc_body event_name="doc_body" trigger_name="attr_out" class="body" just="c">
    - <proc event_name="proc" trigger_name="attr_out" name="Print" just="c">
        <anchor event_name="anchor" trigger_name="attr_out" class="body" name="IDX" index="IDX"
          just="c" />
      - <page_setup event_name="page_setup" trigger_name="attr_out" class="body" index="IDX"
          just="c">
        - <system_title_setup_group event_name="system_title_setup_group"
            trigger_name="attr_out" class="systitleandfootercontainer" index="IDX" just="c">
          - <title_setup_container event_name="title_setup_container" trigger_name="attr_out"
              class="systitleandfootercontainer" type="table" index="IDX" just="c">
            - <title_setup_container_specs event_name="title_setup_container_specs"
                trigger_name="attr_out" colcount="1" index="IDX" just="c">
                <title_setup_container_spec event_name="title_setup_container_spec"
                  trigger_name="attr_out" colcount="1" col="1" type="string" index="IDX"
                  just="c" colwidth="100" />
              </title_setup_container_specs>
            - <title_setup_container_row event_name="title_setup_container_row"
                trigger_name="attr_out" section="body" colcount="1" index="IDX" just="c">
                <system_title_setup event_name="system_title_setup"
                  trigger_name="attr_out" section="body" class="systemtitle" value="The
                  SAS System" row="1" data_row="1" colcount="1" col="1" type="string"
                  index="IDX" just="c" />
              </title_setup_container_row>
            </title_setup_container>
          </system_title_setup_group>
        </page_setup>
      - <system_title_group event_name="system_title_group" trigger_name="attr_out"
          class="systitleandfootercontainer" colcount="1" index="IDX" just="c">
        - <title_container event_name="title_container" trigger_name="attr_out"
            class="systitleandfootercontainer" colcount="1" type="table" index="IDX" just="c">
          - <title_container_specs event_name="title_container_specs" trigger_name="attr_out"
              colcount="1" index="IDX" just="c">
              <title_container_spec event_name="title_container_spec"
                trigger_name="attr_out" colcount="1" col="1" type="string" index="IDX" just="c"
                colwidth="100" />
            </title_container_specs>
          - <title_container_row event_name="title_container_row" trigger_name="attr_out"
```

In the XML output that is generated by Event_Map, PROC PRINT uses events named DOC_HEAD, PROC, TABLE, and so on. The TABLE event uses data from event variables such as STATE, CLASS, and TYPE. After you know the events and variables that generate the output, define the tagset and customize your events. For example, you could redefine the TABLE event to produce customized output.

To define a tagset with which to customize your output, start by specifying Tagsets.Event_Map as the parent tagset. As you redefine events to customize output, these events replace the default events that are defined in the Event_Map tagset. In addition, you can remove the operation of a default event by redefining it as an empty event in the tagset. When you are satisfied with the customized output, remove the Event_Map inheritance and the empty events. Then the output reflects only the events that you defined.

Note: When you first run a SAS process and specify TYPE=Event_Map, you can also generate a style sheet along with the body file. The style sheet shows which style attributes you are using.

Alternatives to Event_Map

To create other types of output, you can use one of the following tagsets as alternatives:

- The Text_Map tagset generates output that is similar to a LISTING output.
- The Style_Popup tagset generates HTML like HTMLCSS. However, in Internet Explorer, Style_Popup displays a window that shows the resolved ODS style template for any item that you click.
- The Style_Display tagset is similar to the Style_Popup tagset, but it generates a simple page of output for you to click.
- The NamedHtml tagset generates HTML output similar to the Style_Popup, tagset but all of the objects are labeled the same as with ODS TRACE.

Defining a Tagset Using SAS DATA Step Functions

A SAS DATA step function performs a computation or system manipulation on arguments and returns a value. In Base SAS software, you can use SAS functions in DATA step programming statements, WHERE expressions, macro language statements, the REPORT procedure, Structured Query Language (SQL), and in statements that are used when creating custom tagsets. Functions can be used on any statement within the tagset language. For information about DATA step functions and statements, see *SAS Functions and CALL Routines: Reference*, *SAS DATA Step Statements: Reference*, and *SAS Programmer's Guide: Essentials*.

Syntax: TEMPLATE Procedure: Creating Markup Language Tagsets

PROC TEMPLATE;
DEFINE TAGSET *tagset-path* </ STORE=*libref.template-store* >;
    <*tagset-attribute-1*; <... *tagset-attribute-n*;>>
    DEFINE EVENT *event-name*;
        <*event-attribute-1*;<*event-attribute-n*;> >
        BLOCK *event-name* < / *event-statement-condition(s)*>;

BREAK </ *event-statement-condition(s)*>;
CLOSE </ *event-statement-condition(s)*>;
CONTINUE </ *event-statement-condition(s)*>;
DELSTREAM $$*stream-variable-name* </ *event-statement-condition(s)*>;
DO </ *event-statement-condition(s)*>;
DONE;
ELSE </ *event-statement-condition(s)*>;
EVAL $<$>*user-defined-variable where-expression*< / *event-statement-condition(s)*>;
FLUSH </*event-statement-condition(s)*>;
ITERATE $*dictionary-variable* | $*list-variable*</ *event-statement-condition(s)*>;
NDENT </ *event-statement-condition(s)*>;
NEXT $*dictionary-variable* | $*list-variable* </ *event-statement-condition(s)*>;
OPEN $$*stream-variable-name* </ *event-statement-condition(s)*>;
PUT <*function*> <NL> <*variable*> <'*text*' > < / *event-statement-condition(s)*>;
PUTL (<*variable*> | <'*text*' >| <*function*> | <NL>) < / *event-statement-condition(s)*>;
PUTLOG (<*variable*> <'*text*' > <*function*>)</ *event-statement-condition(s)*>;
PUTQ (<*variable*> | <'*text*' >| <*function*> | <NL>)</ *event-statement-condition(s)*>;
PUTSTREAM $$*stream-variable-name* </ *event-statement-condition(s)*>;
PUTVARS *variable-group variable-group-value*< / *event-statement-condition(s)*>;
SET $<$>*user-defined-variable-name user-defined-variable-value*
</ *event-statement-condition(s)*>;
STOP </ *event-statement-condition(s)*>;
TRIGGER *event-name* <START | FINISH> </ *event-statement-condition(s)*>;
UNBLOCK *event-name* </ *event-statement-condition(s)*>;
UNSET ALL | $*memory-variable* | $$*stream-variable*</ *event-statement-condition(s)*>;
XDENT </ *event-statement-condition(s)*>;
END;
NOTES;
END;

Statement	Task	Example
DEFINE TAGSET	Create a tagset	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 8
DEFINE EVENT	Define what is written to the output file	Ex. 1, Ex. 2, Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 7
BLOCK	Disable an event	
BREAK	Stop the current event from executing	

Statement	Task	Example
CLOSE	Close the current stream variable and direct all future output to the output file	
CONTINUE	Return the execution of the DO loop to the corresponding DO statement for re-evaluation of the IF event statement condition	
DELSTREAM	Delete the specified stream variable	
DO	Begin a statement block that executes if the required condition is true	
DONE	End a DO or ELSE statement block	
ELSE	Begin a statement block that executes if the corresponding DO statement is false	
Event	Specify one or more conditions that must be true for a DEFINE EVENT statement to execute	
“Event Variables”	Set one or more event attributes	
END	End the event	
EVAL	Create or update a user-defined variable	
FLUSH	Write buffered output to the current output file or the current stream variable	
ITERATE	Iterate through a dictionary variable or list variable and assign its value to the _NAME_ and _VALUE_ event variables for each iteration	
NDENT	Indent output one more level than specified by the INDENT= attribute	Ex. 3, Ex. 5
NEXT	Increase a dictionary or list variable incrementally to the next value and repopulate event variables	
NOTES	Provide information about the tagset	Ex. 3
OPEN	Open or create a stream variable	
PUT	Write text, new lines, variable values, or DATA step function return values to an output file	Ex. 1, Ex. 3, Ex. 4, Ex. 5, Ex. 6
PUTL	Write text, new lines, variable values, or DATA step function return values to an output file and add a new line to the end of the output	
PUTLOG	Write text, new lines, variable values, or DATA step function return values to the log	

Statement	Task	Example
PUTQ	Write text, new lines, variable values, or DATA step function return values to an output file and enclose in quotation marks	Ex. 7
PUTSTREAM	Write the contents of the specified stream variable to an output file	
PUTVARS	Write text, new lines, variable values, or DATA step function return values to an output file for each value in a variable group, list, or dictionary	
SET	Create or update a user-defined variable and its value	
STOP	Move the execution to the end of the current statement block	
TRIGGER	Execute an event	Ex. 3, Ex. 4, Ex. 5, Ex. 6
UNBLOCK	Enable a disabled event	
UNSET	Delete a user-defined variable and its value	
XDENT	Indent output one less indention level	Ex. 3, Ex. 5

DEFINE TAGSET Statement

Creates a tagset.

Requirement: An END statement must be the last statement in the template.

Syntax

DEFINE TAGSET *tagset-path* | Base.Template.Tagset
</ STORE=*libref.template-store* <(READ | WRITE | UPDATE)>>;
<*tagset-attribute-1*; < *tagset-attribute-n*;>>
DEFINE EVENT *event-name*;
statements and attributes
NOTES *'text'*;
END;

Summary of Optional Arguments

STORE=*libref.template-store access-options*

Required Arguments

tagset-path
specifies where to store the tagset.

Default PROC TEMPLATE writes the template to the first template store in the current path where you have Write access.

Requirement A *tagset-path* consists of one or more names that are separated by periods. Each name represents a directory, or level, in a template store.

Tips Use the ODS PATH statement to control the item store where the tagset is stored.

Names are not case sensitive. However, PROC TEMPLATE puts the first letter in uppercase for easier reading.

Base.Template.Tagset
creates a tagset that is the parent of all tagsets that do not explicitly specify a parent. After this template is created, you do not need to explicitly specify it in your SAS programs. It is automatically applied to all output until you specifically remove it from the item store.

CAUTION
The Base.Template.Tagset supplied by SAS contains information used by many tagsets. If this information is not retained, unexpected behavior might occur. To safely create your own Base.Template.Tagset, you can start with the existing Base.Template.Tagset template by writing it to an external file and editing the existing template contents.

Interaction The Base.Template.Tagset top-level template attributes are overridden by other tagsets.

Tip To view an existing tagset to base your own Base.Template.Tagset on, see “Viewing the Contents of a Tagset” on page 630.

Optional Argument

STORE=*libref.template-store access-options*
specifies the template store where the template is stored.

libref.template-store
specifies the current template store.

Default If you omit an *access-option*, then the *template-store* is accessed with UPDATE permissions unless you have Read-Only access.

Tip If the specified template store does not exist, then it is created.

access-options
specifies the access mode for the specified template store.

READ
provides Read-Only access.

WRITE
provides Write access as well as Read access. If the tagset does not exist, then WRITE access creates a new tagset. If the tagset does exist, then WRITE access does not replace an existing tagset.

UPDATE
provides Update access as well as Read access. If the tagset does not exist, then UPDATE does not create a new tagset. If the tagset does exist, then UPDATE replaces it.

Restriction The STORE= option syntax does not become part of the compiled template.

Interaction The STORE= option overrides the search list specified in the PATH statement.

Tagset Attributes

Table 17.1 *Tagset Attributes by Task*

Task	Attribute
Specify the maximum number of characters that will be considered for forced line breaks by ODS	BREAKTEXT_LENGTH= (p. 643)
Specify the maximum ratio of the width of space available for text entry to the length of the text that is supposed to fit in that space	BREAKTEXT_RATIO= (p. 643)
Specify the maximum width of space available for text entry that ODS will consider for placement of automatic line breaks	BREAKTEXT_WIDTH= (p. 644)
Specify the text to use as a copyright	COPYRIGHT= (p. 644)
Specify the name of the event to use by default	DEFAULT_EVENT= (p. 644)
Specify whether the tagset supports embedded style sheets	EMBEDDED_STYLESHEET= (p. 644)
Specify a comma-delimited list of image types or file extensions that are valid for an output destination	IMAGE_FORMATS= (p. 645)
Specify the number of spaces the NDENT and XDENT event statements indent the output	INDENT= (p. 645)
Specify a string, which is printed to the SAS log when the tagset is used	LOG_NOTE= (p. 645)

Task	Attribute
Specify special characters and their translations	MAP= (p. 645)
Specify strings to substitute for special characters	MAPSUB= (p. 645)
Set a category for the output	OUTPUT_TYPE= (p. 646)
Specify whether a byte-order mark is written to the output files when using a UTF character set	NO_BYTE_ORDER_MARK= (p. 646)
Define a nonbreaking space for the markup output	NOBREAKSPACE= (p. 646)
Specify the tagset from which the current template inherits	PARENT= (p. 646)
Specify whether all style attributes are available at all times	PURE_STYLE= (p. 647)
Specify the text to use as a registered trademark	REGISTERED_TM= (p. 647)
Define a string to use for line breaks in the markup output	SPLIT= (p. 647)
Specify whether the tagset lets procedures place columns on top of each other, or side by side	STACKED_COLUMNS= (p. 647)
Specify the text to use as a trademark	TRADEMARK= (p. 647)

BREAKTEXT_LENGTH=*number*
specifies the maximum number of characters that will be considered for forced line breaks by ODS. When the number of characters in the text exceeds the number specified by the BREAKTEXT_LENGTH= option, then line breaks are inserted by the application that displays the output. If the number of characters in the text is less than or equal to the number specified by the BREAKTEXT_LENGTH= option, then any necessary line breaks are inserted by ODS. The placement of the line breaks is based on the total available text width.

Example To instruct ODS to not insert line breaks in text that is longer than 80 characters, specify the following:

```
BreakText_Length=80;
```

BREAKTEXT_RATIO=*number*
specifies the maximum ratio of the width of space available for text entry to the length of the text that is supposed to fit in that space. If the ratio of width space to text length is greater than the ratio specified by the BREAKTEXT_RATIO= option, then any necessary line breaks are inserted by the application that

displays the output. If the ratio of width space to text length is equal to or less than the ratio specified by the BREAKTEXT_RATIO= option, then any necessary line breaks are inserted by ODS.

Example To not insert line breaks into text that is more than 1.5 times longer than the width of space that it is to fit in, specify the following:

```
BreakText_Ratio=1.5;
```

BREAKTEXT_WIDTH=*number*

specifies the maximum width of space available for text entry that ODS will consider for placement of automatic line breaks. If the width of space is greater than the number specified by the BREAKTEXT_WIDTH= option, then any necessary line breaks are inserted by the application that displays the output. If the width of space is less than or equal to the number specified by the BREAKTEXT_WIDTH= option, then ODS inserts necessary line breaks.

Example To instruct ODS to not insert line breaks in text that is going into a space greater than or equal to 40 characters wide, specify the following:

```
BreakText_Width=40;
```

COPYRIGHT= '(*text*)'

specifies the text to use as the copyright.

Requirement When specifying *text*, enclose the text in parentheses and then quotation marks.

DEFAULT_EVENT= '*event-name*'

specifies the name of an event to execute by default when the requested event cannot be found in the tagset.

Requirement When specifying an *event-name*, enclose the name of the event in quotation marks.

Example “Example 3: Creating a New Tagset” on page 698

EMBEDDED_STYLESHEET= YES | NO

specifies whether the tagset supports embedded style sheets.

YES

supports embedded style sheets.

Alias ON

NO

does not support embedded style sheets.

Alias OFF

Default YES

Tip If embedded style sheets are supported and you do not specify a style sheet in the ODS statement, then the style sheet is written to the top of the output file.

IMAGE_FORMATS= '*image-type(s)*'
specifies a comma-delimited list of image types or file extensions that are valid for an output destination. The image types can be any that are supported by SAS/GRAPH. List them in order of preference.

Example The following IMAGE_FORMATS= statement lists valid image types for the HTML destination:

```
image_formats='gif,jpeg,png';
```

INDENT=*n*
specifies how many spaces the NDENT and XDENT event statements indent the output.

n
specifies a numeric value for the number of spaces that you want the output to indent.

Default 0

Tip The INDENT= attribute is valid only in markup family destinations.

Examples “Example 3: Creating a New Tagset” on page 698

“Example 5: Indenting Output” on page 704

LOG_NOTE= '*string*'
defines a string that is printed to the SAS log when the tagset is used.

string
specifies the text that is printed to the SAS log.

Requirement Specify only one string at a time.

MAP= '*characters*'
specifies the special characters that require translation.

characters
specifies one or more special characters.

Requirements When listing special characters in the MAP= statement, omit blank spaces between them.

When you specify special characters, enclose the list of special characters in quotation marks.

Use the MAP= statement with the MAPSUB statement.

Example “Example 3: Creating a New Tagset” on page 698

MAPSUB= '*strings*'
specifies the text to substitute for the characters that are specified in the MAP= statement.

strings
specifies the text strings to substitute for the characters that are specified in the MAP= statement.

Requirements When specifying multiple strings, use a forward slash (/) to separate the text strings.

When specifying strings, enclose the entire string list in quotation marks.

Use the MAPSUB= statement with the MAP= statement.

Example "Example 3: Creating a New Tagset" on page 698

NOBREAKSPACE= '*string*'
defines a nonbreaking space for the markup output.

string
specifies the character that defines a nonbreaking space.

Restriction Specify only one string at a time.

Requirement When specifying a string, enclose the string in quotation marks.

Example "Example 3: Creating a New Tagset" on page 698

NO_BYTE_ORDER_MARK=YES | ON | NO | OFF
specifies whether a byte-order mark is written to the output files when using a UTF character set.

OUTPUT_TYPE= CSV | HTML | LATEX | WML | XML
sets a category for the output.

CSV produces output with comma-separated values.

HTML produces Hypertext Markup Language output.

LATEX produces output in LaTeX, which is a document preparation system for high-quality typesetting.

WML uses the Wireless Application Protocol (WAP) to produce a wireless markup language.

XML produces output in Extensible Markup Language.

Example "Example 3: Creating a New Tagset" on page 698

PARENT= *tagset-path*
specifies the tagset from which the current template inherits.

tagset-path
specifies the name of a directory in a template store.

Default The current template inherits from the specified template in the first template store where you have Read access. The PATH statement specifies which locations to search for templates that were created by PROC TEMPLATE, as well as the order in which to search for them.

Requirement When you specify a parent, all of the template options, attributes, and statements that are specified in the parent's template are used in the current template, unless the current template overrides them.

Tips Specify a tagset that SAS supplies or a customized tagset.

Control the item store from which the tagset is read by using the ODS PATH statement.

Examples "Example 1: Creating a Tagset through Inheritance" on page 689

"Example 8: Using the STACKED_COLUMNS Attribute in a Tagset" on page 708

PURE_STYLE=YES | NO
specifies whether all of the style attributes are available at all times.

REGISTERED_TM= '(*text*)'
specifies the text to use as the registered trademark.

Requirement When specifying *text*, enclose the text in parentheses and then quotation marks.

SPLIT= '*string*'
defines a text string to use for line breaks in the markup output.

Restriction Specify one string at a time.

Requirement When specifying a string, enclose the string in quotation marks.

Example "Example 3: Creating a New Tagset" on page 698

STACKED_COLUMNS= YES | NO
specifies whether the tagset lets procedures stack columns on top of each other, or place them side by side.

YES
stacks columns on top of each other.

Alias ON

NO
stacks columns side by side.

Alias OFF

Default YES.

Tip To place columns side by side, specify the NO or OFF value.

Examples "Example 3: Creating a New Tagset" on page 698

"Example 8: Using the STACKED_COLUMNS Attribute in a Tagset" on page 708

TRADEMARK= '(*text*)'
specifies the text to use as the trademark.

Requirement When specifying *text*, enclose the text in parentheses and then quotation marks.

DEFINE EVENT Statement

Defines what is written to the output file.

Interaction: You can add event statement conditions to any DEFINE EVENT statement. For more information about event statement conditions, see "Event Statement Conditions" on page 687.

Examples: "Example 6: Using Different Styles for Events" on page 706

"Example 7: Modifying an Event to Include Other Style Sheets" on page 708

Syntax

DEFINE EVENT *event-name*;
<*event-attribute-1*;<*event-attribute-n*;> >
BLOCK *event-name* < / *event-statement-condition(s)*>;
BREAK </ *event-statement-condition(s)*>;
CLOSE </ *event-statement-condition(s)*>;
CONTINUE </ *event-statement-condition(s)*>;
DELSTREAM $$*stream-variable-name* </ *event-statement-condition(s)*>;
DO </ *event-statement-condition(s)*>;
DONE;
ELSE </ *event-statement-condition(s)*>;
EVAL $<$>*user-defined-variable where-expression*< / *event-statement-condition(s)*>;
FLUSH </*event-statement-condition(s)*>;
ITERATE $*dictionary-variable* | $*list-variable*</ *event-statement-condition(s)*>;
NDENT </ *event-statement-condition(s)*>;
NEXT $*dictionary-variable* | $*list-variable* </ *event-statement-condition(s)*>;
OPEN $$*stream-variable-name* </ *event-statement-condition(s)*>;
PUT <*function*> <NL> <*variable*> <'*text*' > < / *event-statement-condition(s)*>;
PUTL (<*variable*> | <'*text*' >| <*function*> | <NL>) < / *event-statement-condition(s)*>;
PUTLOG (<*variable*> <'*text*' > <*function*>)</ *event-statement-condition(s)*>;
PUTQ (<*variable*> | <'*text*' >| <*function*> | <NL>)</ *event-statement-condition(s)*>;
PUTSTREAM $$*stream-variable-name* </ *event-statement-condition(s)*>;
PUTVARS *variable-group variable-group-value*< / *event-statement-condition(s)*>;
SET $<$>*user-defined-variable-name user-defined-variable-value*</ *event-statement-condition(s)*>;
STOP </ *event-statement-condition(s)*>;
TRIGGER *event-name* <START | FINISH> </ *event-statement-condition(s)*>;
UNBLOCK *event-name* </ *event-statement-condition(s)*>;
UNSET ALL | $*memory-variable* | $$*stream-variable*</ *event-statement-condition(s)*>;

XDENT </ *event-statement-condition(s)*>;
END;

Summary of Optional Arguments

FILE= BODY | CODE | CONTENTS | DATA | FRAME | PAGES | STYLESHEET;
: Redirect event output to any of the known types of output that are open

PURE_STYLE= YES | NO;
: Enable the event to use any style element that has been defined

STYLE= *style-element*;
: Specify a style element

Required Argument

event-name
specifies the name of the event.

Event Attributes

FILE= BODY | CODE | CONTENTS | DATA | FRAME | PAGES | STYLESHEET;
redirects event output to any of the known types of output files that are open.

Restriction The FILE= attribute is valid only in markup family destinations.

Interaction The names of the output files correspond to the output filenames in the ODS MARKUP statement that are specified with the BODY=, CODE=, CONTENTS=, FRAME=, PAGES=, and STYLESHEET= options. For more information about these options, see the "ODS MARKUP Statement" in *SAS Output Delivery System: User's Guide*.

See The BODY= option in "ODS MARKUP Statement" in *SAS Output Delivery System: User's Guide* for a complete description of the FILE= attribute.

PURE_STYLE= YES | NO;
specifies whether to enable the event to use any of the style elements that have been defined.

YES
enables the event to use any of the style elements that have been defined.

Alias ON

NO
does not enable the event to use any of the style elements that have been defined.

Alias OFF

Default NO

Restriction The PURE_STYLE= attribute is valid only in markup family destinations.

See "DEFINE STYLE Statement" on page 454

STYLE= *style-element*;
specifies a style attribute that applies to a particular part of the output.

Restriction The STYLE= attribute is valid only in markup family destinations.

Tip When a carriage return separates style attributes, add a space before or after the carriage return to prevent syntax errors. SAS does not interpret a carriage return as a space.

See "DEFINE STYLE Statement" on page 454

Example "Example 6: Using Different Styles for Events" on page 706

BLOCK Statement

Disables the specified event.

Tips: To enable the blocked event, use the UNBLOCK statement.

You can block the same event multiple times, but to enable the event, use the same number of UNBLOCK statements.

Syntax

BLOCK *event* </ *event-statement-condition(s)*>;

Required Argument

event
specifies the event.

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

BREAK Statement

Stops the current event from executing. Statements below the BREAK statement are not executed.

Tip: The BREAK statement is most useful when combined with event statement conditions.

Syntax

BREAK < / *event-statement-condition(s)*>;

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

CLOSE Statement

Closes the current stream variable and directs all future output to the output file.

Syntax

CLOSE < / *event-statement-condition(s)*>;

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

CONTINUE Statement

Specifies that the execution of the DO loop returns to the corresponding DO statement for re-evaluation of the IF event statement condition.

See: "DO Statement" on page 653

Syntax

CONTINUE </ *event-statement-condition(s)*>;

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

DELSTREAM Statement

Deletes the specified stream variable.

Syntax

DELSTREAM *stream-variable* < / *event-statement-condition(s)*>;

Required Argument

stream-variable
specifies the stream variable to be deleted.

See "OPEN Statement" on page 660

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

DO Statement

Begins a statement block that executes if the required condition is true.

Syntax

DO / *event-statement-condition(s)*;

Required Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

DONE Statement

Ends a DO or ELSE statement block.

See: "DO Statement" on page 653
"ELSE Statement" on page 654

Syntax

DONE;

ELSE Statement

Begins a statement block that executes if the corresponding DO statement is false.

Tip: If you specify the ELSE statement with the DO statement and the WHILE condition, then the ELSE statement executes only if the WHILE condition is false on the first evaluation.

See: "DO Statement" on page 653

Syntax

ELSE </ *event-statement-condition(s)*>;

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

END Statement

Ends the tagset or event.

Syntax

END;

EVAL Statement

Creates or updates a user-defined variable by setting the value of the variable to the return value of a WHERE expression.

Syntax

EVAL $<$>*user-defined-variable where-expression*< / *event-statement-condition(s)*>;

Required Arguments

user-defined-variable
specifies the user-defined variable that you want to create or update.

A *user-defined-variable* has one of the following forms:

- $*dictionary-variable*['*key*']
- $*list-variable*[<*index*>]
- $*scalar-variable*
- $$*stream-variable*

dictionary-variable
specifies a dictionary variable to assign a *where-expression* return value. A dictionary variable is an array that contains a list of numbers or text strings that are identified by a key.

['*key*']
specifies a subscript that contains the text that identifies where in the dictionary variable that you want to add the return value of the WHERE expression.

Requirement Enclose *key* in quotation marks and brackets.

Tip *key* is case preserving and case sensitive.

Requirement *dictionary-variable* must be preceded by the "$" symbol.

Tip After you create dictionary variables, they are globally available in all events until you delete them with the UNSET Statement on page 677.

See For more information, see “Understanding Variables” on page 631.

list-variable
specifies a list variable to which to assign a *where-expression* return value. A list variable is an array that contains a list of numbers or text strings that are indexed.

[<*index*>]
specifies a subscript that contains a number or numeric variable.

The *index* identifies the location in the list to add the return value of the WHERE expression. If you omit the *index* and specify only empty brackets, or if the value of *index* is greater than the highest *index* number, then the EVAL statement appends the return value to the end of the list.

Requirements Specify brackets [ ], even if you omit an index.

Enclose *index* in brackets.

See For more information, see “Understanding Variables” on page 631.

Requirement *list-variable* must be preceded by a '$' symbol.

Tips List variables are accessed sequentially by using the “ITERATE Statement” on page 657 and the “NEXT Statement” on page 659.

After you create list variables, they are globally available in all events until you use the “UNSET Statement” on page 677 to delete them.

scalar variable
specifies a scalar variable to which to assign a *where-expression* return value.

Requirements Scalar variables must be preceded by the '$' symbol.

After you create scalar variables, they are globally available in all events and persist until you use the UNSET statement to unset them.

stream-variable
specifies a stream variable to which you want to assign a *where-expression* return value. A stream variable is a temporary item store that contains output.

While the stream variable is open, all output from PUT statements is directed to the stream variable until it is closed.

Requirement *stream-variable* must be preceded by the "$$" symbol.

See For information about stream variables, see “Understanding Variables” on page 631.

where-expression
any expression that can be used in the WHERE= data set option.

See For information about expressions that you can use in the WHERE data set option, see the “WHERE= Data Set Option” in *SAS Data Set Options: Reference* . Also see “WHERE Expressions” in *SAS Programmer’s Guide: Essentials*.

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see “Event Statement Conditions” on page 687.

FLUSH Statement

Writes buffered output to the current output file or the current stream variable.

Syntax

FLUSH </ *event-statement-condition(s)*>;

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

ITERATE Statement

Specifies a dictionary variable or list variable to loop through, and assigns the variable's value to the _NAME_ and _VALUE_ event variables for each iteration.

Requirement: You must use the ITERATE statement with the _VALUE_ or _NAME_ event variables. The first value of the dictionary variable or list variable is placed in the _VALUE_ event variable. For dictionary variables, the *key* is placed in the _NAME_ event variable.

See: _VALUE_ and _NAME_ in Table 17.57 on page 680.

Syntax

ITERATE *dictionary-variable | list-variable* </ *event-statement-condition(s)*>;

Required Arguments

dictionary-variable
specifies a dictionary variable.

Requirement *dictionary-variable* must be preceded by the "$" symbol.

Tip User-defined variables are not case sensitive.

See The "EVAL Statement" on page 654 or the "SET Statement" on page 670 for information about dictionary variables

list-variable
specifies a list variable.

Requirement *list-variable* must be preceded by the "$" symbol.

Tip User-defined variables are not case sensitive.

See The "EVAL Statement" on page 654 or the "SET Statement" on page 670 for information about list variables

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

NDENT Statement

Indents output one more level than the number of spaces specified by the INDENT= attribute.

Interaction: The start position of the indention level is set by the INDENT= attribute.

Examples: "Example 3: Creating a New Tagset" on page 698
"Example 5: Indenting Output" on page 704

Syntax

NDENT < / *event-statement-condition(s)*>;

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

NEXT Statement

Specifies to increase a dictionary or list variable incrementally to the next value and to repopulate the event variables _VALUE_ and _NAME_ as appropriate.

Requirement: Use the NEXT statement with the ITERATE statement.

See: _VALUE_ and _NAME_ in Table 17.57 on page 680.

Syntax

NEXT $*dictionary-variable* | $*list-variable* </ *event-statement-condition(s)*>;

Required Arguments

dictionary-variable
specifies a dictionary variable that is designated as an iterator by the ITERATE statement.

Requirement *dictionary-variable* must be preceded by the "$" symbol.

Tip User-defined variables are not case sensitive.

See "ITERATE Statement" on page 657

The "EVAL Statement" on page 654 or the "SET Statement" on page 670 for information about dictionary variables

list-variable
specifies a list variable that is designated as an iterator by the ITERATE statement.

Requirement *list-variable* must be preceded by the "$" symbol.

Tip User-defined variables are not case sensitive.

See "ITERATE Statement" on page 657

The "EVAL Statement" on page 654 or the "SET Statement" on page 670 for information about list variables

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

NOTES Statement

Provides information about the tagset.

Tip: The NOTES statement becomes part of the compiled tagset, which you can view with the SOURCE statement.

See: "Example 3: Creating a New Tagset" on page 698
"Example 8: Using the STACKED_COLUMNS Attribute in a Tagset" on page 708

Syntax

NOTES '*text*';

Required Argument

text
provides information about the tagset.

Requirement When specifying *text*, enclose the text in quotation marks.

OPEN Statement

Opens or creates a stream variable. When the PUT statements occur after the OPEN statement, all text or variable data that is specified by PUT statements is appended to the stream variable instead of the output file.

Interaction: An open stream variable is closed when a new stream variable is opened.

Syntax

OPEN *stream-variable* </ *event-statement-condition(s)*>;

Required Argument

stream–variable
specifies a stream variable, which is a temporary item store that contains output.

Tips User-defined variables are not case sensitive.

If you assign the name of a memory variable to *stream-variable*, then the stream variable resolves as the value of the memory variable. For example, the following program uses the memory variable $MyStream as a stream variable:

```
set $mystream 'test';
open $mystream;
put 'The memory variable $mystream is used as a stream variable';
close;
```

Therefore, the following statements are equivalent:

```
put $$test;
putstream $mystream;
putstream test;
```

The following statements are also equivalent:

```
unset $$test;
delstream $mystream;
delstream test;
```

See "memory variables" on page 632.

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions "Event Statement Conditions" on page 687.

PUT Statement

Writes text, new lines, variable values, or DATA step function return values to an output file.

Examples:
"Example 1: Creating a Tagset through Inheritance" on page 689
"Example 3: Creating a New Tagset" on page 698
"Example 4: Executing Events Using the TRIGGER= Statement" on page 703
"Example 5: Indenting Output" on page 704
"Example 6: Using Different Styles for Events" on page 706

Syntax

PUT <'*text*'> <NL(s)> <*value(s)*> </ *event-statement-condition(s)*>;

Optional Arguments

NL
specifies a new line.

Aliases CR

LF

text
specifies a string of text.

Requirement *text* must be enclosed in quotation marks.

Interactions The PUT statement pairs text strings with variables. A string of text that precedes a variable creates a string-value pair if the variable has a value. For example, for the following PUT statement, if the event variable ForeGround has a value of blue, then the output is color=blue:

```
put 'color=' foreground;
```

If the variable does not have a value, then the text is not written, and there is no output for the text or the variable. For example, for the following PUT statement, if the variables BackGround, ForeGround, and CellPadding do not have values, then the output is <table> followed by a new line:

```
put '<table' 'background=' background 'foreground=' foreground
     'cellpadding=' cellpadding '>' nl;
```

value
specifies any event variable, style variable, dynamic variable, user-defined variable, or DATA step function whose value you want to output.

Restriction DATA step functions cannot be nested.

Requirement User-defined variables must be preceded by a '$' or '$$' character.

Interactions The PUT statement pairs text strings with variables. A string of text that precedes a variable creates a string-value pair, if the variable has a value. For example, for the following PUT statement, if the event variable ForeGround has a value of blue, then the output is color=blue:

```
put 'color=' foreground;
```

If the variable does not have a value, then the text is not written, and there is no output for the text or the variable. For example, for the following PUT statement, if the variables BackGround, ForeGround, and CellPadding do not have values, then the output is <table> followed by a new line:

```
put '<table' 'background=' background 'foreground=' foreground
     'cellpadding=' cellpadding '>' nl;
```

Tip User-defined variables are not case sensitive.

See For information about DATA step functions, see *SAS Functions and CALL Routines: Reference*.

For information about variables, see “Understanding Variables” on page 631.

“Event Variables” on page 680 for a list of event variables

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see “Event Statement Conditions” on page 687.

PUTL Statement

Writes text, new lines, variable values, or DATA step function return values to an output file and automatically adds a new line to the end of the output.

Tip: When the output is large, it is useful to use the PUTL statement because it adds a new line to the end of the output.

Syntax

PUTL <'*text*'> <NL(s)> <*value(s)*> </ *event-statement-condition(s)*>;

Optional Arguments

NL
specifies a new line.

Aliases CR

LF

text
specifies a string of text.

Requirement *text* must be enclosed in quotation marks.

Interactions The PUTL statement pairs text strings with variables. A string of text that precedes a variable creates a string-value pair if the variable has a value. For example, for the following PUTL statement, if the event variable ForeGround has a value of blue, then the output is color=blue followed by a new line:

```
putl 'color=' foreground;
```

If the variable does not have a value, then the text is not written, and there is no output for the text or the variable. For example, for the following PUTL statement, if the variables BackGround,

ForeGround, and CellPadding do not have values, then the output is <table> followed by two new lines:

```
putl '<table' 'background=' background 'foreground=' foreground
     'cellpadding=' cellpadding '>' nl;
```

value
specifies any event variable, style variable, dynamic variable, user-defined variable, or DATA step function whose value you want to output.

Restriction DATA step functions cannot be nested.

Requirement User-defined variables must be preceded by a '$' or '$$' character.

Interactions The PUTL statement pairs strings with variables. A string of text that precedes a variable creates a string-value pair if the variable has a value. For example, for the following PUTL statement, if the event variable ForeGround has a value of blue, then the output is color=blue followed by a new line:

```
putl 'color=' foreground;
```

If the variable does not have a value, then the text is not written, and there is no output for the text or the variable. For example, for the following PUTL statement, if the variables BackGround, ForeGround, and CellPadding do not have values, then the output is <table> followed by two new lines: one that is specified and the other that is generated automatically:

```
putl '<table' 'background=' background 'foreground=' foreground
     'cellpadding=' cellpadding '>' nl;
```

Tip User-defined variables are not case sensitive.

See For information about DATA step functions, see *SAS Functions and CALL Routines: Reference*.

For information about variables, see “Understanding Variables” on page 631.

“Event Variables” on page 680 for a list of event variables

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see “Event Statement Conditions” on page 687.

PUTLOG Statement

Writes text, new lines, variable values, or DATA step function return values to the log.

Restriction: Unlike the other PUT statements, the PUTLOG statement does not specify new lines.

Syntax

PUTLOG <'*text*'> <*value(s)*> </ *event-statement-condition(s)*>;

Optional Arguments

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

text
specifies a string of text.

Requirement *text* must be enclosed in quotation marks.

Interactions The PUTLOG statement pairs text strings with variables. A string of text that precedes a variable creates a string-value pair if the variable has a value. For example, for the following PUTLOG statement, if the event variable ForeGround has a value of blue, then the output that is written to the log is color=blue:

```
putlog 'color=' foreground;
```

If the variable does not have a value, then the text is not written, and there is no output for the text or the variable. For example, for the following PUT statement, if the variables BackGround, ForeGround, and CellPadding do not have values, then the output that is written to the log is <table>:

```
putlog '<table' 'background=' background 'foreground=' foreground
     'cellpadding=' cellpadding '>';
```

value
specifies any event variable, style variable, dynamic variable, user-defined variable, or DATA step function whose value you want to output.

Restriction DATA step functions cannot be nested.

Requirement User-defined variables must be preceded by a '$' or '$$' character.

Interactions The PUTLOG statement pairs text strings with variables. A string of text that precedes a variable creates a string-value pair, if the variable has a value. For example, for the following PUTLOG statement, if the event variable ForeGround has a value of blue, then the output that is written to the log is color=blue:

```
putlog 'color=' foreground;
```

If the variable does not have a value, then the text is not written, and there is no output for the text or the variable. For example, for the following PUTLOG statement, if the variables BackGround, ForeGround, and CellPadding do not have values, then the output that is written to the log is <table>:

```
putlog '<table' 'background=' background 'foreground=' foreground
     'cellpadding=' cellpadding '>';
```

Tip User-defined variables are not case sensitive.

See For information about DATA step functions, see *SAS Functions and CALL Routines: Reference*.

For information about variables, see "Understanding Variables" on page 631.

"Event Variables" on page 680 for a list of event variables

PUTQ Statement

Writes text, new lines, variable values, or DATA step function return values to an output file and places quotation marks around the value of the variable.

Example: "Example 7: Modifying an Event to Include Other Style Sheets" on page 708

Syntax

PUTQ <'*text*'> <NL(s)> <*value(s)*> </ *event-statement-condition(s)*>;

Optional Arguments

NL
specifies a new line.

Aliases CR

LF

text
specifies a string of text.

Requirement *text* must be enclosed in quotation marks.

Interactions The PUTQ statement pairs strings with variables. A string of text that precedes a variable creates a string-value pair if the variable has a value. For example, for the following PUTQ statement, if the event variable ForeGround has a value of blue, then the output is color='blue':

```
putq 'color=' foreground;
```

If the variable does not have a value, then the text is not written, and there is no output for the text or the variable. For example, for the following PUTQ statement, if the variables BackGround, ForeGround, and CellPadding do not have values, then the output is <table> followed by a new line:

```
putq '<table' 'background=' background 'foreground=' foreground
     'cellpadding=' cellpadding '>' nl;
```

value
specifies any event variable, style variable, dynamic variable, user-defined variable, or DATA step function whose value you want to output.

Restriction DATA step functions cannot be nested.

Requirement User-defined variables must be preceded by a '$' or '$$' character.

Interactions The PUTQ statement pairs text strings with variables. A string of text that precedes a variable creates a string-value pair, if the variable has a value. For example, for the following PUTQ statement, if the event variable ForeGround has a value of blue, then the output is color=blue:

```
putq 'color=' foreground;
```

If the variable does not have a value, then the text is not written, and there is no output for the text or the variable. For example, for the following PUTQ statement, if the variables BackGround, ForeGround, and CellPadding do not have values, then the output is <table> followed by a new line:

```
putq '<table' 'background=' background 'foreground=' foreground
     'cellpadding=' cellpadding '>' nl;
```

Tip User-defined variables are not case sensitive.

See For information about DATA step functions, see *SAS Functions and CALL Routines: Reference*.

For information about variables, see "Understanding Variables" on page 631.

"Event Variables" on page 680 for a list of event variables

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

PUTSTREAM Statement

Writes the contents of the specified stream variable to an output file.

Syntax

PUTSTREAM *stream-variable* < / *event-statement-condition(s)*>;

Required Argument

stream-variable
specifies a stream variable, which is a temporary item store that contains output.

Tip If you assign the name of a memory variable to *stream-variable-name*, then the stream variable resolves as the value of the memory variable. For example, the following partial program uses the memory variable $MyStream as a stream variable:

```
set $mystream 'test';
open $mystream;
put 'The memory variable $mystream is used as a stream variable';
close;
```

Therefore, the following statements are equivalent:

```
put $$test;
putstream $mystream;
putstream test;
```

The following statements are also equivalent:

```
unset $$test;
delstream $mystream;
delstream test;
```

See “memory variables” on page 632.

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see “Event Statement Conditions” on page 687.

PUTVARS Statement

Iterates over each value in a variable group, list, or dictionary and writes text, new lines, variable values, or DATA step function return values to an output file. Each iteration populates the special variables _VALUE_ and __NAME_ . PUTVARS prints once for each variable or value that it finds.

Tip: The variable _NAME_ contains the name of the variable. The variable _VALUE_ contains the value of the variable.

See: _VALUE_ and _NAME_ in Table 17.57 on page 680.

Syntax

PUTVARS (*variable-group* | *dictionary-variable* | *list-variable*)<NL(s)> <*'text'*>
<*value(s)*>
< / *event-statement-condition(s)*>;

Required Arguments

variable-group
specifies the type of variable to use in each iteration when you specify the name or value in the variable. For example, if you specify the EVENT option, then the PUTVARS statement loops through all of the event variables in the program.

variable-group is one of the following:

EVENT
specifies to loop through all event variables.

See "Event Variables" on page 680

STYLE
specifies to loop through all style variables.

DYNAMIC
specifies to loop through all dynamic variables.

MEMORY
specifies to loop through all memory variables. A memory variable is classified as a dictionary variable if it is created with a subscript that contains a key. A memory variable is classified as a list variable if it is created with a subscript that is empty or contains an index. If you omit a key or an index, then the memory variable is a numeric or character scalar variable, depending on the variable's value.

Restriction The PUTVARS statement ignores list or dictionary memory variables.

STREAM
specifies to loop through all stream variables.

Interaction The PUTVARS statement pairs text strings with variables. If a string is followed by a variable, then they become a pair. If the variable has a value, then the pair becomes output. If the variable does not have a value, then neither becomes output.

dictionary-variable
specifies a dictionary variable.

Requirement *dictionary-variable* must be preceded by the '$' symbol.

Tip User-defined variables are not case sensitive.

See For information about list variables, see the following sections: "EVAL Statement" on page 654, "SET Statement" on page 670, and "Understanding Variables" on page 631.

list-variable
specifies a list variable.

Requirement *list-variable* must be preceded by the "$" symbol.

Tip User-defined variables are not case sensitive.

See For information about list variables, see the following sections: "EVAL Statement" on page 654, "SET Statement" on page 670, and "Understanding Variables" on page 631.

Optional Arguments

NL
specifies a new line.

Aliases CR

LF

text
specifies a string of text.

Requirement *text* must be enclosed in quotation marks.

value
specifies any event variable, style variable, dynamic variable, user-defined variable, or DATA step function whose value you want to output.

Restriction DATA step functions cannot be nested.

Requirement User-defined variables must be preceded by a '$' or '$$' character.

Tip User-defined variables are not case sensitive.

See For information about DATA step functions, see *SAS Functions and CALL Routines: Reference*.

For information about variables, see "Understanding Variables" on page 631.

For a list of event variables, see "Event Variables" on page 680.

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

SET Statement

Creates or updates a user-defined variable and its value.

Syntax

SET $*dictionary-variable entry* </ *event-statement-condition(s)*>;
SET $*list-variable entry* </ *event-statement-condition(s)*>;
SET $*scalar-variable* | $$*stream-variable entry* </ *event-statement-condition(s)*>;

Required Arguments

dictionary-variable
specifies an array that contains a list of numbers or text strings that is identified by a key.

dictionary-variable has the following form:

$*dictionary-variable*['*key*']

['*key*']
specifies a subscript that contains text or a variable that has a character value.

Requirement Enclose *key* in quotation marks and brackets.

Tip *key* is case preserving and case sensitive.

Example The following example puts two key value pairs into the dictionary variable MyDictionary:

```
set $mydictionary['URL1'] 'links internally';
 set $mydictionary['URL2'] 'links externally';
```

Requirement *dictionary-variable* must be preceded by the '$' symbol.

Tips Dictionary variables are accessed sequentially by using the ITERATE and NEXT statements. See “ITERATE Statement” on page 657. Also see “NEXT Statement” on page 659.

After they are created, dictionary variables are globally available in all events until you delete them by using the “UNSET Statement” on page 677.

entry
specifies the value of a dictionary variable, list variable, scalar variable, or stream-variable.

An *entry* is one of the following:

function
specifies a DATA step function.

Restriction Functions cannot be nested.

See *SAS Functions and CALL Routines: Reference* for information about SAS functions

text
specifies a string of text.

Requirement *text* must be enclosed in quotation marks.

variable
specifies any event variable, style variable, dynamic variable, user-defined variable, or DATA step function whose value you want to output.

Restriction *variable* cannot be a stream variable.

Requirement User-defined variables must be preceded by a '$' character.

Tips If you assign a *variable* entry that is the name of a memory variable to *stream variable*, then the stream variable resolves as the value of the memory variable. For example, the following program uses the memory variable $MyStream as a stream variable:

```
set $mystream 'test';
open $mystream;
put 'The memory variable $mystream is used as a stream variable';
close;
```

Therefore, the following statements are equivalent:

```
put $$test;
putstream $mystream;
putstream test;
```

The following statements are also equivalent:

```
unset $$test;
delstream $mystream;
delstream test;
```

User-defined variables are not case sensitive.

See "memory variables" on page 632

For information about variables, see "Understanding Variables" on page 631.

"Event Variables" on page 680 for a list of event variables

list-variable
specifies an array that contains a list of numbers or strings of text that are indexed.

list-variable has the following form:

$*list-variable*[<*index*>]

[<*index*>]
specifies a subscript that contains a number or numeric variable. The index identifies the location in the list to add an entry. If you omit the index and only specify empty brackets, or if the value of the index is greater than the highest index number, then the SET statement appends the entry to the end of the list.

Requirements Specify brackets [ ], even if you omit an index.

Enclose *index* in brackets.

Tip List entries are accessed by positive or negative indexes. Positive indexes start at the beginning of a list. Negative indexes start at the end of a list. For example, the following list variable, $Mylist[2], identifies the second entry in the list variable $Mylist. In this case, the index is 2. The list variable

$Mylist[-2] identifies the second entry from the end of the list variable $Mylist. In this case, the index is [-2].

Example The following example adds three values onto the end of the list variable MyList and modifies the value of the second entry.

```
set $mylist[] 'one';
 set $mylist[] 'two';
 set $mylist[] 'hello';
 set $mylist[2] 'This is Really two';
```

Requirement *list-variable* must be preceded by a '$' symbol.

Tips List variables are accessed sequentially by using the ITERATE and NEXT statements. See “ITERATE Statement” on page 657 and “NEXT Statement” on page 659.

After they are created, list variables are globally available in all events until you delete them using the “UNSET Statement” on page 677.

scalar-variable

an area of memory that contains numeric or character data.

Requirement Scalar variables must be preceded by the '$' symbol.

Tip After they are created, list variables are globally available in all events until you delete them using the “UNSET Statement” on page 677.

stream-variable

specifies a stream variable, which is a temporary item store that contains output.

While the stream variable is open, all output from PUT statements is directed to the stream variable until it is closed.

Requirement *user-defined-variable-name* must be preceded by the '$$' symbol.

Tip If you assign a variable entry that is the name of a memory variable to *stream-variable*, then the stream variable resolves as the value of the memory variable. For example, the following program uses the memory variable $MyStream as a stream variable:

```
set $mystream 'test';
open $mystream;
put 'The memory variable $mystream is used as a stream variable';
close;
```

Therefore, these statements are equivalent:

```
put $$test;
putstream $mystream;
putstream test;
```

These statements are also equivalent:

```
unset $$test;
delstream $mystream;
delstream test;
```

See “*variable*” on page 672

"memory variables" on page 632

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

Adding Entries to Dictionary Variables

Use this form of the SET statement to add an entry to a dictionary variable.

SET $*dictionary-variable entry* </ *event-statement-condition(s)*>;

A dictionary variable is an array that contains a list of numbers or text strings that is identified by a key. A dictionary variable has, as part of its name, a preceding '$' symbol and a subscript that contains a text string or a variable that has a character value. The text or variable within the subscript is called a key. Keys are case preserving and case sensitive. After they are created, dictionary variables are globally available in all events and persist until you unset them with the UNSET statement.

An entry is a variable, string of text, or function. If a string of text follows the dictionary variable, then the entry becomes a key-value pair. For example, the following program adds two key-value pairs to a dictionary:

```
set $mydictionary['URL1'] 'links internally';
set $mydictionary['URL2'] 'links externally';
```

Adding Entries to List Variables

Use this form of the SET statement to add an entry to a list variable.

SET $*list-variable entry* </ *event-statement-condition(s)*>;

A list variable is an array that contains a list of numbers or text strings that are indexed. As part of their name, list variables have a preceding '$' symbol and a subscript that is empty or contains a number or numeric variable. The number within the subscript is called an index. After they are created, list variables are globally available in all events and persist until you unset them with the UNSET statement. List entries are accessed by positive or negative indexes. Positive indexes start at the beginning of a list. Negative indexes start at the end of a list.

For example, the following list variable, $Mylist[2], identifies the second entry in the list variable $Mylist. In this case, the index is 2. The list variable $Mylist[-2] identifies the second entry from the end of the list variable $Mylist. In this case, the index is [-2].

STOP Statement

Specifies that execution moves to the end of the current statement block.

Syntax

STOP </ *event-statement-condition(s)*>;

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition(s)* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

TRIGGER Statement

Executes an event.

Tip: The TRIGGER statement explicitly requests a specific action of an event.

Examples:
"Example 3: Creating a New Tagset" on page 698
"Example 4: Executing Events Using the TRIGGER= Statement" on page 703
"Example 5: Indenting Output" on page 704
"Example 6: Using Different Styles for Events" on page 706

Syntax

TRIGGER *event-name* <START | FINISH> </ *event-statement-condition(s)*>;

Without Arguments

If a triggered event does not have start or finish sections, then it runs the current event statements.

Required Argument

event-name
specifies the name of the event.

Optional Arguments

START
specifies the start section of an event.

Interaction If the program is in the start section of an event, then any event that is triggered runs its start section.

FINISH
specifies the finish section of an event.

Interaction If the program is in the finish section of an event, then any event that is triggered runs its finish section.

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement an *event-statement-condition* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

UNBLOCK Statement

Enables a disabled event.

Requirement: Because you can block the same event multiple times, to enable the event use the same number of UNBLOCK statements as BLOCK statements.

Interaction: To disable an event, use the BLOCK statement.

Syntax

UNBLOCK *event-name* </ *event-statement-condition(s)*>;

Required Argument

event-name
specifies the name of the event.

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement an *event-statement-condition* must be preceded by a slash (/).

See For information about these conditions, see "Event Statement Conditions" on page 687.

UNSET Statement

Deletes a user-defined variable and its value.

Syntax

UNSET ALL | *dictionary-variable* | *list-variable* | *scalar-variable* | *stream-variable*
</ *event-statement-condition(s)*>;

Required Arguments

ALL
deletes all dictionary variables, list variables, and scalar variables.

Tip You must delete stream variables individually.

dictionary-variable
specifies an array that contains a list of numbers or text strings that are identified by a key.

A *dictionary-variable* has the following form:

$*dictionary-variable*['*key*']

['*key*']
specifies the location in the dictionary variable of the value that you want to delete.

Requirements Enclose *key* in quotation marks and brackets.

key must be a string of text or a character variable.

Tip *key* is case preserving and case sensitive.

Requirement A *dictionary-variable* must be preceded by the '$' symbol.

list-variable
specifies an array that contains a list of numbers or strings of text that are indexed.

A *list-variable* has the following form:

$*list-variable*[<*index*>]

[<*index*>]
specifies the location in the list variable of the value to be deleted. If you omit the *index* and specify empty brackets, then the entire list variable is deleted.

Requirements Specify brackets [ ], even if you omit an index.

index must be number or numeric variable.

Enclose *index* in brackets.

Tip List entries are accessed by positive or negative indexes. Positive indexes start at the beginning of a list. Negative indexes start at the end of a list. For example, in the following code, the first UNSET statement deletes the first entry from the top of the list variable MyList. The second UNSET statement deletes the first entry from the bottom of the MyList list variable:

```
unset $mylist[-1];
 unset $mylist[1];
```

Requirement A *list-variable* must be preceded by a '$' symbol.

scalar-variable
specifies a scalar variable to delete.

Requirement Scalar variables must be preceded by the '$' symbol.

See "SET Statement" on page 670 or "Understanding Variables" on page 631 for information about scalar variables

stream-variable
specifies a stream variable to delete.

Requirement A *user-defined-variable-name* must be preceded by the '$$' symbol.

Tip If you assign a variable entry that is the name of a memory variable to *stream-variable*, then the stream variable resolves as the value of the memory variable. For example, the following program uses the memory variable $MyStream as a stream variable:

```
set $mystream 'test';
open $mystream;
put 'The memory variable $mystream is used as a stream variable';
 close;
```

Therefore, the following statements are equivalent:

```
put $$test;
putstream $mystream;
putstream test;
```

The following statements are also equivalent:

```
unset $$test;
delstream $mystream;
delstream test;
```

See “SET Statement” on page 670 or “Understanding Variables” on page 631 for information about memory variables.

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement An *event-statement-condition* must be preceded by a slash (/).

See For information about these conditions, see “Event Statement Conditions” on page 687.

XDENT Statement

Indents output one less indention level, using the number of spaces specified by the INDENT= attribute.

Interaction: The starting level of indention is set by the NDENT= statement.

Examples: “Example 3: Creating a New Tagset” on page 698
“Example 5: Indenting Output” on page 704

Syntax

XDENT </ *event-statement-condition(s)*>;

Optional Argument

event-statement-condition(s)
specifies one or more conditions that must be true for the event statement to execute.

Requirement *event-statement-condition* must be preceded by a slash (/).

See For information about these conditions, see “Event Statement Conditions” on page 687.

Usage: TEMPLATE Procedure: Creating Markup Language Tagsets

Event Variables

Event variables include text, formatting, and data values that are associated with events. These variables originate in many places, such as table templates, the procedures, titles, bylines, and processing. Event variables also include any style attributes that are used in the program. The following table lists the internally generated event variables that are used in the DEFINE EVENT statement of PROC TEMPLATE.

SAS includes these accessibility and compatibility features to improve the usability of SAS for users with a wide range of abilities.

Table 17.2 *Accessibility Variables*

Event Variable	Description
ABBR	Specifies an abbreviation for the event variable.
ACRONYM	Specifies an acronym for an event variable.
ALT	Specifies an alternate description of an event variable.
CAPTION	Specifies the caption for a table.
LONGDESC	Specifies the long description of an event variable.
SUMMARY	Specifies a summary of a table.

Table 17.3 *Data Variables*

Event Variable	Description
NAME	Contains the name of the current variable.
VALUE	Contains the value of the current variable.

Event Variable	Description
DNAME	Specifies the name of the column in the data component to associate with the current column. DNAME is specified with the DATANAME= attribute in a column template. For information, see the “DATANAME=*column-name*;” on page 592.
LABEL	Specifies a label for the variable. The LABEL event variable is set with the LABEL= attribute in the column template. For information, see the “LABEL="*text*" | *variable*;” on page 596.
NAME	Specifies the name of the variable. NAME is set with the VARNAME= attribute in the column template. For information, see the “VARNAME=*variable-name* | *variable*;” on page 601.
VALUE	Specifies the current value.
VALUECOUNT	Specifies the count of the variable.

Table 17.4 *Event Meta Variables*

Event Meta Variables	Description
EMPTY	Sets a flag to determine whether an event is called as an empty tag.
EVENT_NAME	Specifies the requested event name.
STATE	Specifies the current state of the event, which is either START or FINISH.
TRIGGER_NAME	Specifies the name of the event that is triggered.

Table 17.5 *Formatting Data*

Event Variable	Description
CLOSURE	Specifies whether the endpoints of a format range are included or excluded, for example (<-, -, -<, <-<, and so on).
COL_ID	Specifies the column ID to identify columns. Used for the OIMDBM format type by the XML LIBNAME engine.

Event Variable	Description
DATAENCODING	Specifies the encoding type for Raw value. It is always Base64.
MISSING	Specifies the value that indicates that no data value is stored. By default, SAS uses a single period (.) for a missing numeric value and a blank space for a missing character value. In addition, for a numeric missing value, a special missing value indicator represents different categories of missing data by assigning one of the letters A through Z, or an underscore.
NO_WRAP	Specifies that the current cell should not wrap text or insert hyphens.
PRECISION	Specifies the number of places to the right of the decimal. The PRECISION variable is used by the XML LIBNAME engine.
RANGEEND	Specifies the end value of a range in a format.
RANGESTART	Specifies the start value of a range in a format
RAWVALUE	Specifies the base64 encoding of the stored machine representation of the original value.
SASFORMAT	Specifies the SAS format used to format a value.
SCALE	Specifies the total number of places in the floating point number. The SCALE event variable is used by the XML LIBNAME engine.
TYPE	Specifies the STRING, DOUBLE, CHAR, BOOL, or INT data type.
UNFORMATTEDTYPE	Specifies the data type before formatting.
UNFORMATTEDVALUE	Specifies the value before formatting.
UNFORMATTEDWIDTH	Specifies the width before formatting.

Table 17.6 *General Use Variables*

Variable	Description
ANCHOR	Specifies the current anchor, which is the last value of the anchor tag (for example, IDX).

Variable	Description
DATA_VIEWER	Specifies the name of the Data Viewer, such as Table, Batch, Tree, Graph, Report, or Print.
DATE	Specifies the date.
DEST_FILE	Specifies the current destination file, which is one of the following: body, contents, pages, frame, code, or style sheet.
FIRSTPAGE	Specifies the first page of the output file.
LANGUAGE	Specifies the language of the current output. The LANGUAGE event variable is set only when it is an Asian language.
OUTPUT_LABEL	Specifies the label of the current output object.
OUTPUT_NAME	Specifies the name of the current output object.
OUTPUT_TYPE	Specifies the output type as specified in the tagset.
PAGE_COUNT	Specifies the page count since the files were opened.
PROC_COUNT	Specifies how many procedures have run since the files were opened.
PROC_NAME	Specifies the name of the current procedure.
SASLONGVERSION	Specifies the long format of the SAS version.
SASVERSION	Specifies the short format of the SAS version.
SPACE	Specifies the string that the tagset uses for a nonbreaking space.
SPLIT	Specifies the string that the tagset uses for line breaks.
STYLE	Specifies the current style that is in use.
STYLE_ELEMENT	Specifies the name of the current style element.
SUPPRESS_CHARSET	Specifies the Suppress Charset Registry setting.
TIME	Specifies the time.
TOCLEVEL	Specifies the table of contents level.

Variable	Description
TOTAL_PAGE_COUNT	Specifies the total page count since ODS was opened.
TOTAL_PROC_COUNT	Specifies the number of procedures that have run since ODS was opened.

Table 17.7 ODS Statement Variables: Variables That Originate with the ODS Statement That Invoked the Tagset

Event Variable	Description
AUTHOR	Specifies the author of the output. The value of the AUTHOR event variable is set from an ODS statement, or, by default, is the user that is running SAS.
BASENAME	Specifies the name of the BASE= option as set in an ODS statement.
BODY_NAME	Specifies the name of the body file.
BODY_TITLE	Specifies the title of the body file.
BODY_URL	Specifies the URL of the body file.
CODE_NAME	Specifies the name of the code file.
CODE_TITLE	Specifies the title of the code file.
CODE_URL	Specifies the URL of the code file.
CONTENTS_NAME	Specifies the name of the contents file.
CONTENTS_TITLE	Specifies the title of the contents file.
CONTENTS_URL	Specifies the URL of the contents file.
DATA_NAME	Specifies the name of the data file.
DATA_TITLE	Specifies the title of the data file.
DATA_URL	Specifies the URL of the data file.
ENCODING	Specifies the encoding of the output for converting text data into a numbering system that computers recognize.
FRAME_NAME	Specifies the name of the frame file.

Event Variable	Description
FRAME_TITLE	Specifies the title of the frame file.
FRAME_URL	Specifies the URL of the frame file.
GRAPH_PATH_NAME	Specifies the path of the graph as specified by the ODS PATH statement.
GRAPH_PATH_URL	Specifies the URL of the graph.
NO_BOTTOM	is nonzero if you specified the NO_BOTTOM_MATTER option in the ODS MARKUP statement.
NO_TOP	is nonzero if you specified the NO_TOP_MATTER option in the ODS MARKUP statement.
OPERATOR	Specifies the operator. The value of the OPERATOR event variable is set from an ODS statement or, by default, is the user that is running SAS.
PAGES_NAME	Specifies the name of the pages file.
PAGES_TITLE	Specifies the title of the pages file.
PAGES_URL	Specifies the URL of the pages file.
PATH	Specifies the path as set by an ODS statement.
PATH_NAME	Specifies the pathname.
PATH_URL	Specifies the path location.
STYLESHEET_NAME	Specifies the name of the style sheet file.
STYLESHEET_TITLE	Specifies the title of the style sheet file.
STYLESHEET_URL	Specifies the URL of the style sheet file.
TAGSET	Specifies the name of the current tagset.
TAGSET_ALIAS	Specifies the alias of the current tagset as given in the ODS MARKUP statement.
TITLE	Specifies the title from the ODS statement.
TRANTAB	Specifies the translation table name for character conversions.

Table 17.8 Table Variables

Event Variable	Description
CLABEL	Specifies the label for the output object in the contents file, the Results window, and the trace record. Set with the CONTENTS_LABEL= attribute in the table template. For information, see the “CONTENTS_LABEL= "*string*" | *variable*;” on page 619.
COLCOUNT	Specifies the number of columns in the current table.
COLEND_EA	Specifies the ending column number.
COLSPAN	Specifies the number of columns that the cell spans.
COLSTART	Specifies the column number where the cell starts.
DATA_ROW	Specifies that the current row is a data row.
IS_STACKED	Specifies that the columns are stacked.
ROW	Specifies the current table row, which includes headers.
ROWSPAN	Specifies the number of rows that the current cell spans.
SECTION	Specifies the header, body, or footer of the table.
WIDTH	Specifies the width. WIDTH is most commonly used for COLSPECS.

Table 17.9 URL Variables

Event Variable	Description
NOBASE	Sets a flag to determine whether to use the value for BASE= option as part of the URL. 0 uses the BASE= option, and 1 does not use BASE= option.
TARGET	Specifies the target that is associated with the URL.
URL	Specifies a fully formed URL.

Event Statement Conditions

Event statement conditions specify one or more conditions that must be true for a DEFINE EVENT statement to execute. An event statement condition must be preceded by a slash (/).

Event statement conditions have the following form:

define-event-statement </ *event-statement-condition(s)*>;

define-event-statement
specifies a DEFINE EVENT statement.

event-statement-condition
specifies a condition to evaluate.

event-statement-condition is one of the following:

ANY (*variable-1*,<…, *variable-n*>)
checks a list of comma-delimited variables for values. If any of the variables has a value, then the condition is true.

Example
```
put 'One of our variables has a value!'
    nl/if any(background, foreground, cellpadding, cellspacing);
```

BREAKIF *event*
stops an event that is executing. The current statement is executed and the event ends.

Tip Using the BREAKIF condition is more efficient than using a PUT event statement and a BREAK event statement with an IF condition together. For example, the following statements are equivalent:

```
put 'Foreground has a value!' /breakif exists(foreground);
```

```
put 'Foreground has a value!' /if exists(foreground);
 break /if exists(foreground);
```

**CMP ("*string*", *variable*
variable-list)**
compares, for equality, a string to a variable or list of variables.

Example
```
put 'The foreground is blue!' nl/if cmp('blue',foreground);
```

CONTAINS (*argument-1*, *argument-2*)
searches the first argument for the second argument.

Example
```
set $myvariable 'some random text';
 put 'myvariable contains 'ran' nl/if contains($myvariable, 'ran');
```

EXIST
**EXISTS (*variable*
variable-list)**
determines whether a variable or a list of variables has values. If all of the variables have values, then the condition is true. If a variable has an empty string of length 0, then the variable has no value and the condition is false.

Tip Use the MISSING event variable with the EXIST condition to determine whether a value is missing.

Example
```
put 'All of our variables have a value!'
    nl/if exists(background, foreground, cellpadding, cellspacing);
```

IF
WHEN
WHERE (<*value*><'*string*'><*variable*>)

tests for existence or equality. IF, WHEN, and WHERE are optional and interchangeable. An IF, a WHEN, or a WHERE condition compares values and strings, or checks variables for values.

Restriction When you specify an IF condition with a single, user-defined variable, then the variable is evaluated to determine whether it has a value, according to the variable's type. A string variable type uses the length to determine existence, a numeric variable type uses value, and a dictionary array variable type uses the key (if there is a key specified, then the test is true).

Example All of the following are equivalent:
```
put 'Foreground has a value!' nl/if (foreground);
 put 'Foreground has a value!' nl/if exists(foreground);
 put 'Foreground has a value!' nl/when exists(foreground);
 put 'Foreground has a value!' nl/exists(foreground);
 put 'Foreground has a value!' nl/where existsforeground);
```

NOT
!
^ <'*string*'><*variable*>

negates a condition. You can use the keyword NOT or the characters '!' or '^'.

Restriction The character '!' works only as the first character in a condition. The standard WHERE processing syntax is required for subsequent characters.

Example
```
put 'The foreground is not red!' nl/if not cmp('red', foreground);
 put 'The foreground is not red or blue' /if !cmp('red', foreground)
      and ^cmp('blue', foreground);
 put 'The foreground is not red or blue' /if ^cmp('red', foreground)
      and ^cmp('blue', foreground);
```

WHILE *condition-expression*

indicates that the corresponding statement block should loop until the WHILE value becomes false.

Restriction The WHILE condition can be used only with the DO statement.

Example
```
eval $count 0;

  do /while $count < 10;
     eval $i $count+1;
     continue /if $count eq 5;
     stop /if $count eq 8;
     put 'Count is ' $i nl;
 else;
     put 'Count was never less than 10' nl;
```

```
done;
```

Examples: TEMPLATE Procedure: Creating Markup Language Tagsets

Example 1: Creating a Tagset through Inheritance

Features:

DEFINE TAGSET statement
- DEFINE EVENT statement
- PUT statement
- PARENT= attribute

Other ODS features
- ODS PATH statement
- ODS MARKUP statement

Details

This example defines a new tagset called Tagsets.MyTags that creates customized HTML output. The new tagset is created through inheritance. Most of the required formatting is available in the tagset Tagsets.Chtml, which SAS supplies.

Program

```
ods path Work.templat (update)
   sashelp.tmplmst (read);

proc template;
   define tagset tagsets.mytags /store=Work.templat;
      parent=tagsets.chtml;



define event colspecs;
   put 'These are my new colspecs' nl;
end;

define event table;
   put '<p>' nl '<table>';
```

```
finish:
   put '</table>';
end;

define event system_title;
end;


end;
run;

ods tagsets.mytags body='custom-tagset-filename.html';



proc print data=sashelp.class;
run;

ods tagsets.mytags close;
```

Program Description

Define a new tagset. The DEFINE TAGSET statement creates a new tagset called Tagsets.Mytags. The PARENT= attribute is used so that the new tagset Tagsets.Mytags inherits events from Tagsets.Chtml. Note that the ODS PATH statement is specified at the beginning to establish the search path.

```
ods path Work.templat (update)
   sashelp.tmplmst (read);

proc template;
   define tagset tagsets.mytags /store=Work.templat;
      parent=tagsets.chtml;
```

Define three events. The DEFINE EVENT statements create three events called COLSPECS, TABLE, and SYSTEM_TITLE. The COLSPECS event specifies text. The TABLE event specifies tags to include in the template. The SYSTEM_TITLE event deletes titles.

```
define event colspecs;
   put 'These are my new colspecs' nl;
end;

define event table;
   put '<p>' nl '<table>';
finish:
   put '</table>';
end;

define event system_title;
end;
```

End the tagset. This END statement ends the tagset. The RUN statement executes the PROC TEMPLATE step.

```
end;
run;
```

Specify the user-defined tagset. The following code specifies the user-defined tagset Tagsets.Mytags as the tagset for the output.

```
ods tagsets.mytags body='custom-tagset-filename.html';
```

Print the data set. PROC PRINT creates the report. ODS writes the report to the body file.

```
proc print data=sashelp.class;
run;
```

Stop the creation of the tagset. The ODS TAGSET. MYTAGS CLOSE statement closes the MARKUP destination and all the files that are associated with it. Close the destination so that you can view the output with a browser.

```
ods tagsets.mytags close;
```

To see the customized CHTML tags, view the source from the web browser: From the browser's toolbar, select **View** ⇨ **Source**.

Output 17.3 *Generated Output: Mytags.Chtml*

These are my new colspecs

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

Use the tagset Tagsets.Chtml, which SAS provides. Compare the output from Tagsets.Mytags to that from Tagsets.Chtml, which SAS supplies. Use the following ODS code to specify the SAS tagset. You can specify any tagset by using TYPE= in an ODS MARKUP statement.

```
ods markup tagset=tagsets.chtml body='default-tagset-filename.html';

proc print data=sashelp.class;
run;

ods markup close;
```

To view the customized CSV tagset Tagsets.CHTML submit the following code:

```
proc template;
   source tagsets.chtml;
run;
```

Output 17.4 *A Display That Uses the Default CHTML Tagset*

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

Example 2: Creating a Tagset By Copying a Tagset's Source

Features: SOURCE statement

DEFINE TAGSET statement
    DEFINE EVENT statement

Details

This example copies the source for a tagset that SAS supplies, modifies the template, and then builds a new tagset for custom output. To create a new tagset, use the SOURCE statement in PROC TEMPLATE to copy a tagset's source. Then you can customize the template as necessary.

Program

Copy the SAS tagset to an external file. The following statements copy the tagset source from the SAS tagset Tagsets.CSV to the SAS log.

```
proc template;
      source tagsets.csv;
run;
```

Partial Log: Default CSV Tagset That SAS Supplies

```
1    proc template;
NOTE: Writing HTML Body file: sashtml.htm
2          source tagsets.csv;
define tagset Tagsets.Csv;
   notes "This is the CSV definition";
   mvar _CURRENCY_SYMBOL _DECIMAL_SEPARATOR _THOUSANDS_SEPARATOR _CURRENCY_AS_NUMBER
      _PERCENTAGE_AS_NUMBER _DELIMITER;

   define event initialize;

      trigger set_options;

      trigger documentation;

      trigger compile_regexp;
   end;

   define event options_set;

      trigger set_options;

      trigger documentation;

      trigger compile_regexp;
   end;

   define event documentation;

      trigger help /if cmp( $options["DOC"], "help");

      trigger quick_reference /if cmp( $options["DOC"], "quick");
   end;

   define event help;
      putlog
"================================================================================";
      putlog "The CSV Tagset Help Text.";
      putlog " ";
      putlog "This Tagset/Destination creates output in comma separated value
format.";
      putlog " ";
      putlog
            "Numbers, Currency and percentages are correctly detected and show as
numeric


 ...more lines of log...



define event warning;
   break /if ^$notes;
   put VALUE NL;
end;
mapsub = "/""""/";
map = """";
registered_tm = "(r)";
trademark = "(tm)";
copyright = "(c)";
output_type = "csv";
stacked_columns = OFF;
end;
NOTE: Path 'Tagsets.Csv' is in: Sashelp.Tmplmst.
3    run;
NOTE: PROCEDURE TEMPLATE used (Total process time):
      real time           13.14 seconds
```

Create the customized tagset. Submit the following PROC TEMPLATE code to create the customized tagset Tagsets.Mycsv. The DEFINE EVENT TABLE statement uses the PUT NL statements to add two blank lines to the output file. One blank line is placed before the table and the other is placed after the table.

```
proc template;
define tagset Tagsets.mycsv;
   notes 'This is the My CSV template';
   define event table;
      start:
         put nl;
      finish:
         put nl;
   end;
   define event put_value;
      put VALUE;
   end;
   define event put_value_cr;
      put VALUE NL;
   end;
   define event row;
      finish:
         put NL;
   end;
   define event header;
      start:
         put ',' /if ^cmp( COLSTART, '1');
         put '''';
         put VALUE;
      finish:
         put '''';
   end;
   define event data;
      start:
         put ',' /if ^cmp( COLSTART, '1');
         put '''';
         put VALUE;
      finish:
         put '''';
   end;
   define event colspanfill;
      put ',';
   end;
   define event rowspanfill;
      put ',' /if ^exists( VALUE);
   end;
   define event breakline;
      put NL;
   end;
   define event splitline;
      put NL;
   end;
   registered_tm = '(r)';
   trademark = '(tm)';
   copyright = '(c)';
   output_type = 'csv';
   stacked_columns = OFF;
```

```
end;
```

To view the customized CSV tagset Tagsets.Mycsv, submit the following code:

```
proc template;
   source tagsets.mycsv;
run;
```

Log Output: Customized CSV Tagset Tagsets.Mycsv

```
proc template;
   define tagset Tagsets.Mycsv / store = Work.Templat;
      notes 'This is the My CSV template';
      define event table;
         start:
            put NL;
         finish:
            put NL;
      end;
      define event put_value;
         put VALUE;
      end;
      define event put_value_cr;
         put VALUE NL;
      end;
      define event row;
         finish:
            put NL;
      end;
      define event header;
         start:
            put ',' /if ^cmp( COLSTART, '1');
            put '''';
            put VALUE;
         finish:
            put '''';
      end;
      define event data;
         start:
            put ',' /if ^cmp( COLSTART, '1');
            put '''';
            put VALUE;
         finish:
            put '''';
      end;
      define event colspanfill;
         put ',';
      end;
      define event rowspanfill;
         put ',' /if ^exists( VALUE);
      end;
      define event breakline;
         put NL;
      end;
      define event splitline;
         put NL;
      end;
      output_type = 'csv';
      copyright = '(c)';
      trademark = '(tm)';
      registered_tm = '(r)';
      stacked_columns = OFF;
   end;
run;
```

Example 3: Creating a New Tagset

Features:

PROC TEMPLATE features
- DEFINE TAGSET statement
- NOTES statement
- DEFINE EVENT statement
- NDENT statement
- PUT statement
- TRIGGER statement
- XDENT statement

Tagset Attributes
- DEFAULT_EVENT attribute
- INDENT= attribute
- OUTPUT_TYPE attribute
- MAP= attribute
- MAPSUB= attribute
- NOBREAKSPACE= attribute
- SPLIT= attribute
- STACKED_COLUMNS= attribute

Details

This example shows a new tagset that does not inherit events from another tagset. This is a customized tagset for specific PROC FREQ output.

Program

```
proc template;
   define tagset Tagsets.newloc / store = Work.Templat;
      notes 'This is the Location Report Template';


   define event basic;
   end;

   define event doc;
   start:
      put '' nl nl;
      put '' nl;
      put '' nl;
      put '' nl;
      ndent;
   finish:
      xdent;
      put nl;
      put '';
```

```
   end;

   define event system_title;
      put '';
      put VALUE;
      put '';
      put nl nl;
   end;
define event header;
   start:
   trigger country /if cmp(LABEL, 'EmpCountry');
   end;

   define event data;
   start:
   trigger frequency /if cmp(name, 'Frequency');
   end;

   define event country;
      put '' nl ;
      ndent ;
      put '' ;
      put VALUE ;
      put '' nl  ;
   end;

   define event frequency;
      put '' ;
      put VALUE ;
      put '' nl  ;
      xdent ;
      put '' nl ;
   end;

   output_type = 'xml';
   default_event = 'basic';
   indent = 2;
   split = '';
   nobreakspace = '';
   mapsub = '/</>/&/';
   map = '<>&';
   stacked_columns=off;
   end;
run;
```

Program Description

Create the new tagset Tagsets.Newloc. The DEFINE TAGSET statement creates a new tagset Tagsets.Newloc and specifies where you want to store the tagset.

```
proc template;
   define tagset Tagsets.newloc / store = Work.Templat;
      notes 'This is the Location Report Template';
```

Define seven events. The seven DEFINE statements create the events named BASIC, DOC, SYSTEM_TITLE, HEADER, DATA, COUNTRY, and FREQUENCY.

```
   define event basic;
   end;

   define event doc;
   start:
      put '' nl nl;
      put '' nl;
      put '' nl;
      put '' nl;
      ndent;
   finish:
      xdent;
      put nl;
      put '';
   end;

   define event system_title;
      put '';
      put VALUE;
      put '';
      put nl nl;
   end;
define event header;
   start:
   trigger country /if cmp(LABEL, 'EmpCountry');
   end;

   define event data;
   start:
   trigger frequency /if cmp(name, 'Frequency');
   end;

   define event country;
      put '' nl ;
      ndent ;
      put '' ;
      put VALUE ;
      put '' nl  ;
   end;

   define event frequency;
      put '' ;
      put VALUE ;
      put '' nl  ;
      xdent ;
      put '' nl ;
   end;

   output_type = 'xml';
   default_event = 'basic';
   indent = 2;
   split = '';
   nobreakspace = '';
```

```
   mapsub = '/</>/&/';
   map = '<>&';
   stacked_columns=off;
   end;
run;
```

New Tagsets.NewlocTemplate Source

```
proc template;
   define tagset Tagsets.newloc / store = Work.Templat;
      notes 'This is the Location Report Template';
      define event basic;
      end;
      define event doc;
         start:
            put '' NL NL;
            put '' NL;
            put '' NL;
            put '' NL;
            ndent;
         finish:
            xdent;
            put NL;
            put '';
      end;
      define event system_title;
         put '';
         put VALUE;
         put '';
         put NL NL;
      end;
      define event header;
         start:
            trigger country /if cmp( LABEL, 'EmpCountry');
      end;
      define event data;
         start:
            trigger frequency /if cmp( name, 'Frequency');
      end;
      define event country;
         put '' NL;
         ndent;
         put '';
         put VALUE;
         put '' NL;
      end;
      define event frequency;
         put '';
         put VALUE;
         put '' NL;
         xdent;
         put '' NL;
      end;
      map = %nrstr('<>&');
      mapsub = %nrstr('//&/');
      nobreakspace = ' ';
      split = '';
      indent = 2;
      default_event = 'basic';
      output_type = 'xml';
      stacked_columns = OFF;
   end;
run;
```

Example 4: Executing Events Using the TRIGGER= Statement

Features:

DEFINE TAGSET statement
- DEFINE EVENT statement
- PUT statement
- TRIGGER statement

Other ODS features
- ODS *directory.tagset-name* statement

Details

This example illustrates how to execute events.

Program

Execute different events. The TRIGGER statement executes another event. For example, the start section of DOC triggers the start section of MYTEST and OTHEREVENT. MYTEST has a start section, so output is generated. OTHEREVENT is stateless (no start or finish sections), but output is generated.

```
proc template;
   define tagset tagsets.mytagset;
      define event doc;
      start:
         put 'start of doc' nl;
         trigger mytest;
         trigger otherevent;
      finish:
         trigger mytest;
         put 'finish of doc' nl;
         trigger mytest start;
         trigger otherevent;
         trigger mytest finish;
      end;

      define event mytest;
      start:
         put 'start of mytest' nl;
      finish:
         put 'finish of mytest' nl;
      end;

      define event otherevent;
         put 'This is my other event' nl;
```

```
      end;
   end;
run;

ods tagsets.mytagset file='custom-tagset-filename.txt';
ods tagsets.mytagset close;
```

Output

To view the output Tagsets.Mytagset, open the file in a text editor.

Output 17.5 *Output Created from Events and Tagsets.Mytagset Template*

```
start of doc
start of mytest
This is my other event
finish of mytest
finish of doc
start of mytest
This is my other event
finish of mytest
```

Example 5: Indenting Output

Features:

PROC TEMPLATE features
- DEFINE TAGSET statement
- DEFINE EVENT statement
- PUT statement
- NDENT statement
- TRIGGER statement
- XDENT statement

TAGSET attributes
- INDENT= attribute

Other ODS features
- ODS directory.tagset-name statement

Details

This example illustrates how to indent the output using a tagset. When you view a file with an extension of .xml in an XML-compliant browser, the browser ignores any indention in the file in favor of its own indention algorithm.

Program

Set the beginning indention level and then proceed to increment the indention levels. The INDENT= tagset attribute determines how much the NDENT and XDENT event statements indent output.

```
proc template;
   define tagset tagsets.mytagset2;
   indent = 4;

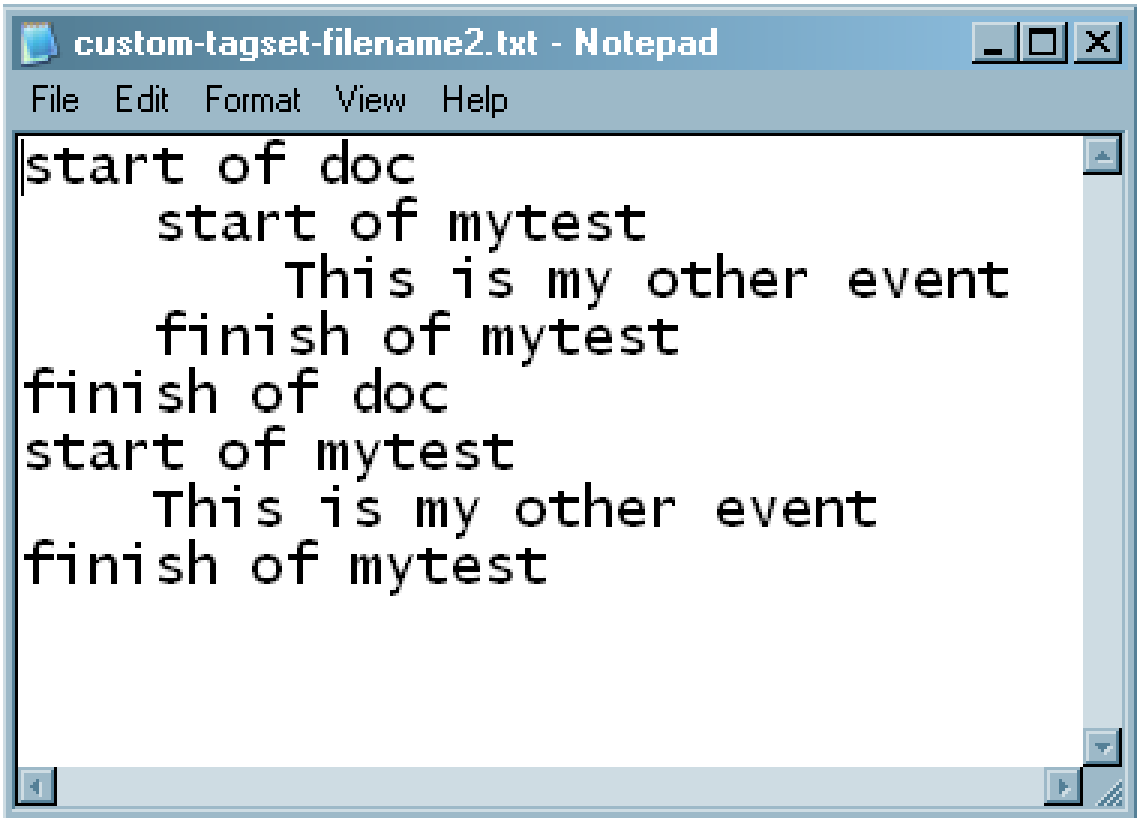
      define event doc;
      start:
         put 'start of doc' nl;
         ndent;
         trigger mytest;
         trigger otherevent;
      finish:
         trigger mytest;
         xdent;
         put 'finish of doc' nl;
         trigger mytest start;
         trigger otherevent;
         trigger mytest finish;
      end;

      define event mytest;
      start:
         put 'start of mytest' nl;
         ndent;
      finish:
         xdent;
         put 'finish of mytest' nl;
      end;

      define event otherevent;
         put 'This is my other event' nl;
      end;
   end;
run;
ods tagsets.mytagset2 file='custom-tagset-filename2.txt';
ods tagsets.mytagset2 close;
```

Output

Output 17.6 *Output Created from Events and Using Tagsets.Mytagset2 Template Source*



Example 6: Using Different Styles for Events

Features: DEFINE EVENT statement
PUT statement
TRIGGER statement
STYLE= event attribute

Details

This example uses different styles for events.

Program

Specify the events. The following events are from the SAS tagset Tagsets.Htmlcss, and they show how ODS creates notes. By defining the GNOTE event and setting

the proper style in the right place, ODS creates a two-cell table that has a banner using the appropriate banner style and a content cell that has the appropriate content style.

```
define event Gnote;
           start:
               put '<div>';
               trigger align;
               put '>';
               put '<table>';
               put '<tr>' nl;
           finish:
               put '</tr>' nl;
               put '</table>' nl;
               put '</div>';
       end;

       define event GBanner;
           put '' nl;
           trigger pre_post;
           put '' nl;
       end;

       define event GNContent;
           put '';
           trigger pre_post start;
           put VALUE;
           trigger pre_post finish;
           put '';
       end;

       define event noteBanner;
           style=NoteBanner;
           trigger GBanner;
       end;

       define event NoteContent;
           style=NoteContent;
           trigger GNContent;
       end;

       define event note;
           trigger Gnote start;
           trigger noteBanner;
           trigger noteContent;
           trigger Gnote finish;
       end;

       define event WarnBanner;
           style=WarnBanner;
           trigger GBanner;
       end;

       define event WarnContent;
           style=WarnContent;
           trigger GNContent;
       end;
```

```
define event Warning;
    trigger Gnote start;
    trigger WarnBanner;
    trigger WarnContent;
    trigger Gnote finish;
end;
```

Example 7: Modifying an Event to Include Other Style Sheets

Features: PROC TEMPLATE features
DEFINE EVENT statement
PUTQ statement

Details

The following program provides some example code that you can use to link a previously created style sheet to an event that you define.

Program

Define an event that links to a style sheet. This code defines an event that creates a link to a previously created style sheet instead of the style sheet that SAS generated.

```
define event stylesheet_link;
putq '<link rel= 'STYLESHEET' type='text/css'
href=' URL '>' nl / if exists(url);
putq '<link rel= 'STYLESHEET' type='text/css'
href='http://your/stylesheet/url/goes/here'>' nl;
putq '<link rel= 'STYLESHEET' type='text/css'
href='http://your/stylesheet/url/goes/here'>' nl;
end;
```

Example 8: Using the STACKED_COLUMNS Attribute in a Tagset

Features: DEFINE TABLE statement
NOTES statement
COLUMN statement

DEFINE statement (for columns)

DEFINE TAGSET statement
- Tagset attribute
- PARENT= attribute
- STACKED_COLUMNS= attribute

Other ODS features
- ODS *directory.tagset-name* statement
- ODS PHTML statement
- ODS _ALL_ CLOSE statement

Details

This example shows the difference between stacking data one column on top of another and placing data side by side. (For more information about stacked columns, see the “DEFINE TABLE Statement” on page 533.)

Program

```
proc template;
   define table Base.Standard;
      notes 'Table template for PROC Standard.';
      column name (mean std) n label;
      define name;  header='Name' varname='Name' style=RowHeader;
      end;
      define mean;  header='Mean/Std Dev' varname='Mean' format=D12.;
      end;
      define std;   header='/Standard/Deviation'
         varname='stdDev' format=D12.;
      end;
      define n;     header='N' format=best.;
      end;
      define label; header='Label' varname='Label';
      end;
      byline wrap required_space=3;
   end;
run;
proc template;
   define tagset tagsets.myhtml;
        parent=tagsets.phtml;
        stacked_columns=no;
   end;
run;



proc template;
   define tagset tagsets.myhtml;
        parent=tagsets.phtml;
        stacked_columns=no;
   end;
```

```
run;

ods tagsets.myhtml file='not_stacked.html';
proc standard print data=sashelp.class;
run;


ods _all_ close;
```

Program Description

Create a table template. The DEFINE TABLE statement creates the table template.

```
proc template;
   define table Base.Standard;
      notes 'Table template for PROC Standard.';
      column name (mean std) n label;
      define name;  header='Name' varname='Name' style=RowHeader;
      end;
      define mean;  header='Mean/Std Dev' varname='Mean' format=D12.;
      end;
      define std;   header='/Standard/Deviation'
         varname='stdDev' format=D12.;
      end;
      define n;     header='N' format=best.;
      end;
      define label; header='Label' varname='Label';
      end;
      byline wrap required_space=3;
   end;
run;
proc template;
   define tagset tagsets.myhtml;
        parent=tagsets.phtml;
        stacked_columns=no;
   end;
run;
```

Customize the tagset by stacking the values side by side. This customized tagset has STACKED_COLUMNS= NO. Note that the SAS tagset, Tagsets.Phtml, has STACKED_COLUMNS=YES.

```
proc template;
   define tagset tagsets.myhtml;
        parent=tagsets.phtml;
        stacked_columns=no;
   end;
run;
```

Create HTML output and specify the location for storing the HTML output. The ODS TAGSETS.MYHTML statement opens the markup language destination and creates the HTML output. The output objects are sent to the external file not_stacked.html in the current directory. The PROC STANDARD statement generates the statistics for the sashelp.class data set. The PRINT option prints the report.

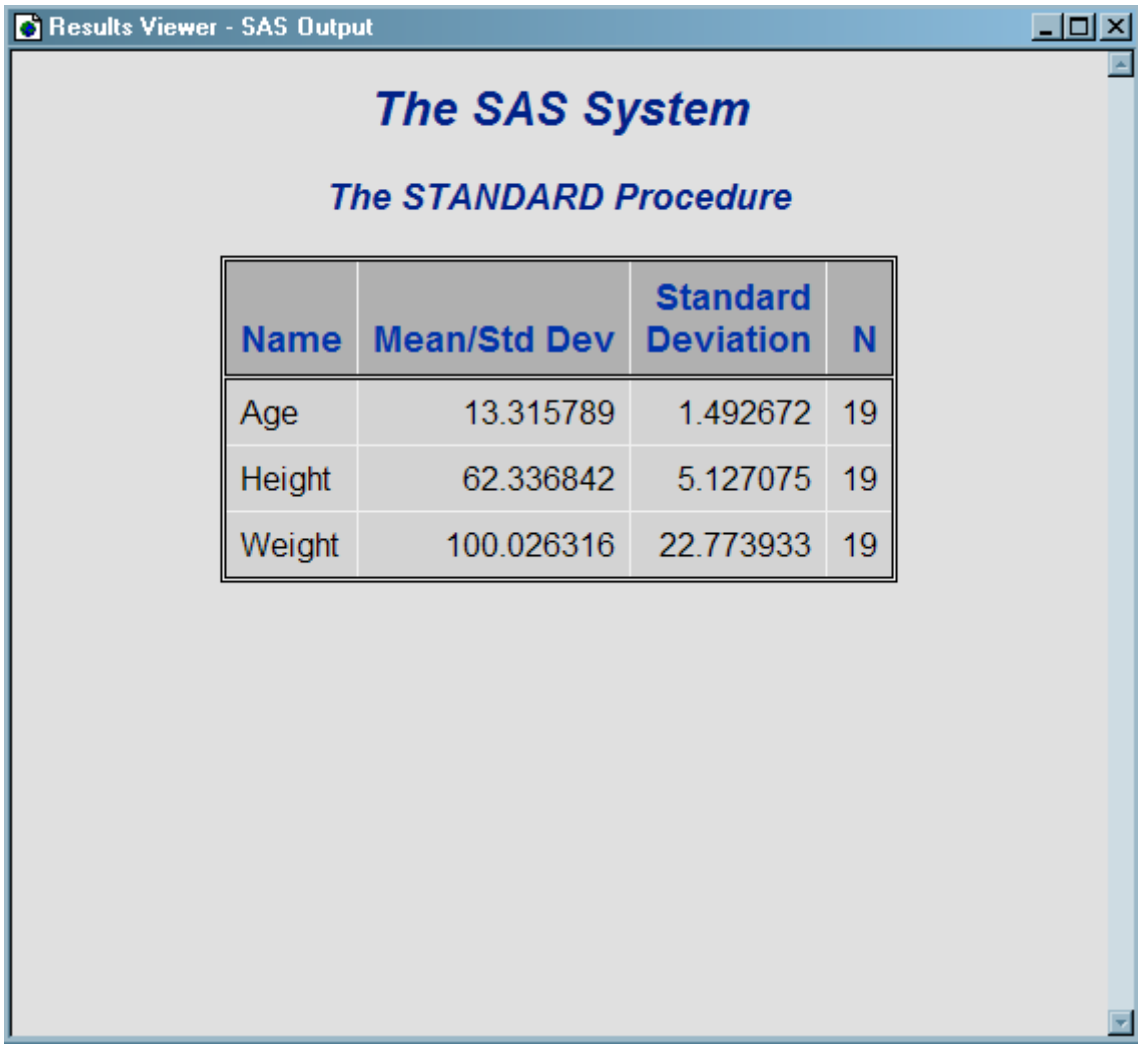
```
ods tagsets.myhtml file='not_stacked.html';
proc standard print data=sashelp.class;
run;
```

Stop the creation of the HTML output. The ODS _ALL_ CLOSE statement closes all open destinations and all files associated with them. For HTML output, close the HTML destination so that you can view the output with a browser.

```
ods _all_ close;
```

Output

Output 17.7 *Output with Values Side by Side*



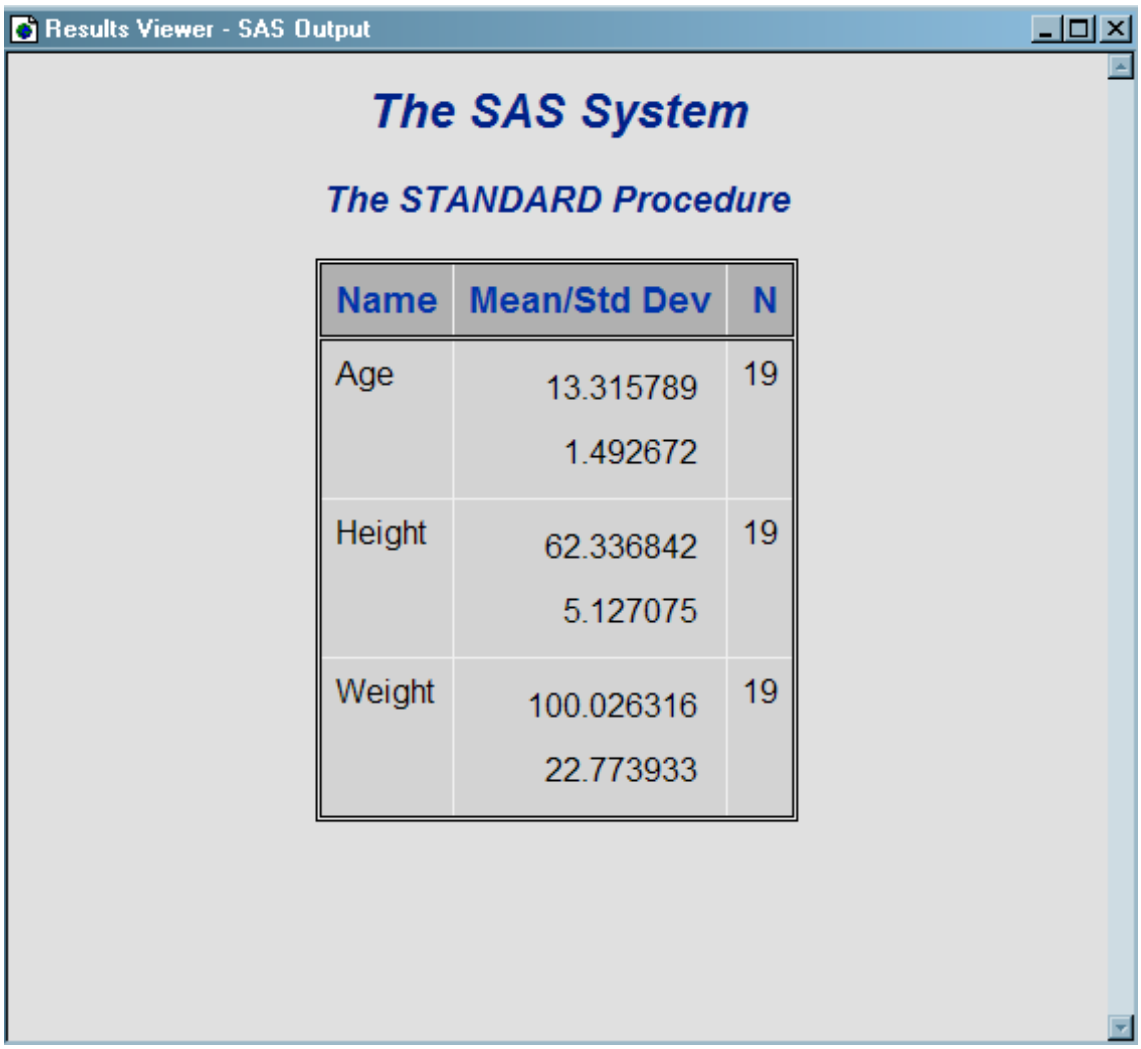
Program

Create the same file but with stacked values. The STACKED_COLUMNS=YES statement shows the same values stacked in the SAS tagset PHTML.

```
ods phtml file='stacked.html';
proc standard print data=sashelp.class;
run;
ods _all_ close;
```

Output

Output 17.8 *Output with Values Stacked One on Top of Another*



PART 8

ODS Styles Reference

Chapter 18
Overview 715

Chapter 19
Style Templates 723

Chapter 20
Style Elements 777

Chapter 21
Style Attributes 807

18

Overview

Understanding Styles, Style Elements, and Style Attributes 715
Using Styles with Base SAS Procedures 721

Understanding Styles, Style Elements, and Style Attributes

The appearance of SAS output is controlled by ODS style templates (ODS styles). ODS styles are produced from compiled STYLE templates written in PROC TEMPLATE style syntax. An ODS style template is a collection of style elements that provides specific visual attributes for your SAS output.

- A style element is a named collection of style attributes that apply to a particular part of the output. Each area of ODS output has a style element name that is associated with it. The style element name specifies where the style attributes are applied. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside the cells. Style elements might also specify default colors and fonts for output that uses the style.
- A style attribute is a visual property, such as color, font properties, and line characteristics, that is defined in ODS with a reserved name and value. Style attributes are collectively referenced by a style element within a style template. Each *style attribute* specifies a value for one aspect of the presentation. For example, the BACKGROUNDCOLOR= attribute specifies the color for the background of an HTML table or for a colored table in printed output. The FONTSTYLE= attribute specifies whether to use a Roman font or an italic font.

Note: Because styles control the presentation of the data, they have no effect on output objects that go to the DOCUMENT or OUTPUT destination.

Available styles are in the SASHELP.TMPLMST item store.In batch mode or SAS Studio, you can display the list of available style templates by using the LIST statement in PROC TEMPLATE:

```
proc template;
   list styles / store=sashelp.tmplmst;
run;
```

For complete information about viewing ODS styles, see “Viewing ODS Styles Supplied by SAS” on page 723.

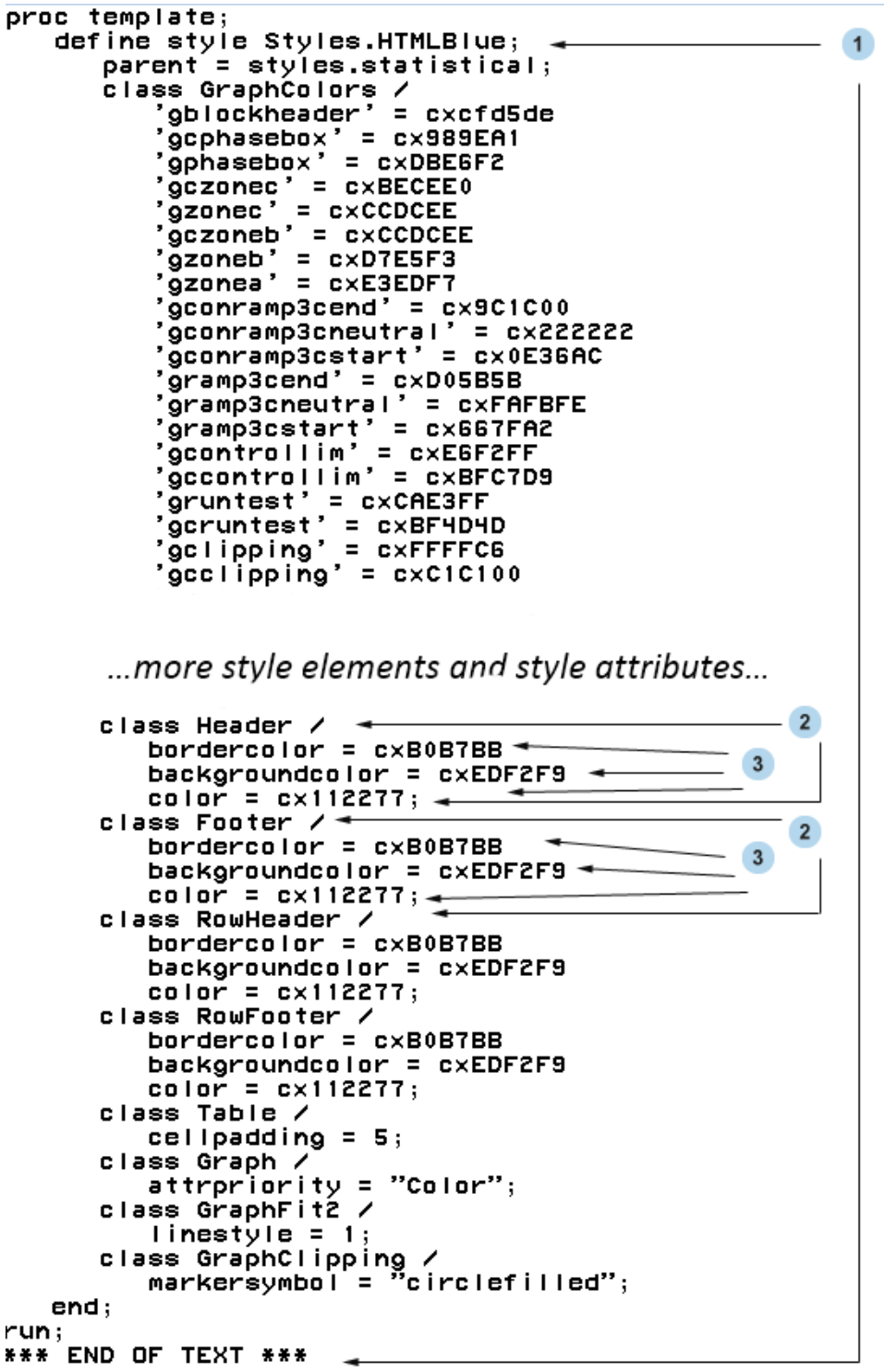
HTML5 output uses the HTMLEncore style template. To help you become familiar with styles, style elements, and style attributes, look at the relationship between them.

You can use the SOURCE statement in PROC TEMPLATE to display the structure of a style template. The following code prints the structure of the HTMLBlue style template to the SAS log:

```
proc template;
   source styles.HTMLBlue;
run;
```

The following figure illustrates the structure of a style. The figure shows the relationship between the style, the style elements, and the style attributes.

Figure 18.1 *Diagram of the HtmlBlue Style*



The following list corresponds to the numbered items in the preceding figure:

1 Styles.HtmlBlue is the *style*. Styles describe how to display presentation aspects (color, font, font size, and so on) of the SAS output. A style determines the overall appearance of the ODS documents that use it. The default style for HTML output is HtmlBlue. Each style consists of style elements.

You can create new styles with the “DEFINE STYLE Statement” on page 454. New styles can be created independently or from an existing style. You can use “PARENT= Statement” on page 461 to create a new style from an existing style. For complete documentation about ODS styles, see Chapter 19, “Style Templates,” on page 723.

2 Header and Footer are examples of *style elements*. A style element is a collection of style attributes that apply to a particular part of the output for a SAS program. For example, a style element might contain instructions for the presentation of column headings or for the presentation of the data inside table cells. Style elements might also specify default colors and fonts for output that uses the style. Style elements exist inside styles and consist of one or more style attributes. Style elements can be user-defined or supplied by SAS. User-defined style elements can be created by the “STYLE Statement” on page 462.

Note: For a list of the default style elements used for HTML and markup languages and their inheritance, see Chapter 20, “Style Elements,” on page 777.

3 BORDERCOLOR=, BACKGROUNDCOLOR=, and COLOR= are examples of *style attributes*. Style attributes specify a value for one aspect of the area of the output that its style element applies to. For example, the COLOR= attribute specifies the value `cx112277` for the font color. For a list of style attributes supplied by SAS, see Chapter 21, “Style Attributes,” on page 807.

Style attributes can be referenced with style references. See “*style-reference*” on page 879 for more information about style references.

The following table shows commonly used style attributes that you can set with the STYLE= option in PROC PRINT, PROC TABULATE, and PROC REPORT. Most of these attributes apply to parts of the table other than cells (for example, table borders and the lines between columns and rows). Note that not all attributes are valid in all destinations. For more information about these style attributes, their valid values, and their applicable destinations, see “Style Attributes Tables” on page 808.

Table 18.1 *Style Attributes for PROC REPORT, PROC TABULATE, and PROC PRINT*

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
ASIS=	X	X		X		X
BACKGROUNDCOLOR=	X	X	X	X	X	X
BACKGROUNDIMAGE=	X	X	X	X	X	X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
BORDERBOTTOMCOLOR=	X	X		X		
BORDERBOTTOMSTYLE=	X	X	X	X		
BORDERBOTTOMWIDTH=	X	X	X	X		
BORDERLEFTCOLOR=	X	X		X		
BORDERLEFTSTYLE=	X	X	X	X		
BORDERLEFTWIDTH=	X	X	X	X		
BORDERCOLOR=	X	X		X	X	X
BORDERCOLORDARK=	X	X	X	X	X	X
BORDERCOLORLIGHT=	X	X	X	X	X	X
BORDERRIGHTCOLOR=	X	X		X		
BORDERRIGHTSTYLE=	X	X	X	X		
BORDERRIGHTWIDTH=	X	X	X	X		
BORDERTOPCOLOR=	X	X		X		
BORDERTOPSTYLE=	X	X	X	X		
BORDERTOPWIDTH=	X	X	X	X		
BORDERWIDTH=	X	X	X	X	X	X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
CELLPADDING=	X		X		X	
CELLSPACING=	X		X		X	
CELLWIDTH=	X	X	X	X		X
CLASS=	X	X	X	X	X	X
COLOR=	X	X	X			
FLYOVER=	X	X		X		X
FONT=	X	X	X	X	X	X
FONTFAMILY=	X	X	X	X	X	X
FONTSIZE=	X	X	X	X	X	X
FONTSTYLE=	X	X	X	X	X	X
FONTWEIGHT=	X	X	X	X	X	X
FONTWIDTH=	X	X	X	X		X
FRAME=	X		X		X	
HEIGHT=	X	X		X	X	X
HREFTARGET=		X		X		X
HTMLSTYLE=	X	X	X	X	X	
NOBREAKSPACE=[2]	X	X		X		X
OUTPUTWIDTH=	X	X	X	X	X	
POSTHTML=[1]	X	X	X	X	X	X
POSTIMAGE=	X	X	X	X	X	X
POSTTEXT=[1]	X	X	X	X	X	X
PREHTML=[1]	X	X	X	X	X	X
PREIMAGE=	X	X	X	X	X	X

Attribute	PROC REPORT STATEMENT REPORT Area	PROC REPORT Areas: CALLDEF, COLUMN, HEADER, LINES, SUMMARY	PROC TABULATE STATEMENT TABLE	PROC TABULATE STATEMENTS VAR, CLASS, BOX, CLASSLEV, KEYWORD	PROC PRINT TABLE location	PROC PRINT: all locations other than TABLE
PRETEXT=[1]	X	X	X	X	X	X
PROTECTSPECIALCHARS=		X		X	X	X
RULES=	X		X		X	
TAGATTR=	X	X	X	X	X	X
TEXTALIGN=	X	X	X	X	X	X
URL=		X		X		X
VERTICALALIGN=		X		X		X
WIDTH=	X	X	X	X	X	

1 When you use these attributes in this location, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table. For complete documentation about style attributes and their values, see Chapter 21, “Style Attributes,” on page 807.

2 To help prevent unexpected wrapping of long text strings when using PROC REPORT with the ODS RTF destination, set NOBREAKSPACE=OFF in a location that affects the LINE statement. The NOBREAKSPACE=OFF attribute must be set in the PROC REPORT code either on the LINE statement or on the PROC REPORT statement where style(lines) is specified.

Using Styles with Base SAS Procedures

Most Base SAS procedures that support ODS use one or more table templates to produce output objects. These table templates include templates for table elements: columns, headers, and footers. Each table element can specify the use of one or more style elements for various parts of the output. These style elements cannot be specified within the syntax of the procedure, but you can use customized styles for the ODS destinations that you use. For more information about customizing tables and styles, see Chapter 14, “TEMPLATE Procedure,” on page 433.

The Base SAS reporting procedures, PROC PRINT, PROC REPORT, and PROC TABULATE, enable you to quickly analyze your data and organize it into easy-to-read tables. You can use the STYLE= option with these procedure statements to modify the appearance of your report. The STYLE= option enables you to make changes in sections of output without changing the default style for all of the output.

You can customize specific sections of procedure output by specifying the STYLE= option in specific statements within the procedure.

19

Style Templates

Viewing ODS Styles Supplied by SAS 723
Program for Viewing Multiple Styles 725
Table of Suggested ODS Styles 726
ODS Styles Gallery 729
EPUB Daisy Style 729
HTML Styles 731
PDF Styles 754
Styles for the ODS Destination for Excel 756
Styles for the ODS Destination for PowerPoint 759
Styles for the ODS Destination for Word 768

Viewing ODS Styles Supplied by SAS

Over thirty ODS styles are available for use with ODS. To view the names of all of the style templates that are shipped with SAS, submit the following program. The style templates are all located in the Styles folder by default.

```
proc template;
   path sashelp.tmplmst;
   list styles;
run;
```

Figure 19.1 *Styles Supplied by SAS: Partial Listing*

Listing of: SASHELP.TMPL_EN
Path Filter is: Styles
Sort by: PATH/ASCENDING

Obs	Path	Type
1	Styles	Dir
2	Styles.Aqua	Style
3	Styles.Daisy	Style
4	Styles.Default	Style
5	Styles.Dove	Style
6	Styles.EGDefault	Style
7	Styles.Excel	Style
8	Styles.ExcelAqua	Style
9	Styles.ExcelHighContrast	Style
10	Styles.ExcelIgnite	Style
11	Styles.ExcelIlluminate	Style
12	Styles.ExcelMarine	Style
13	Styles.ExcelMidnight	Style
14	Styles.ExcelOpal	Style
15	Styles.ExcelUmstead	Style
16	Styles.HTMLBlue	Style
17	Styles.HTMLEncore	Style
18	Styles.HighContrast	Style
19	Styles.HighContrastLarge	Style
20	Styles.Ignite	Style
21	Styles.Illuminate	Style
22	Styles.Journal	Style

To view the source code of a specified style, submit the following code:

```
proc template;
   source styles.style-name;
run;
```

Example Code 19.1 *Partial Source Code for the HTMLBlue Style*

```
proc template;
79      source styles.htmlblue;
define style Styles.Htmlblue;
   parent = styles.statistical;
   class GraphColors /
      'gndata12' = cxECE8C4
      'gndata11' = cxDBD8F8
      'gndata10' = cxC6E4BF
      'gndata9' = cxE6CEAD
      'gndata8' = cxE5C1D4
      'gndata7' = cxCCDFF0
      'gndata6' = cxDDDEB5
      'gndata5' = cxDBC7E7
      'gndata4' = cxD5C6B4
      'gndata3' = cxB7D4D3
      'gndata2' = cxE7B3B4
      'gndata1' = cxBBC2DC
      'gndata' = cxBBC2DC
```

Note: If you are using SAS Studio, you do not need to specify the STYLE= option. You can go to **Options** ⇨ **Preferences** ⇨ **Results** and change the style from the drop-down list for your selected destination.

Program for Viewing Multiple Styles

This program creates a sample report in HTML5, PDF, and RTF of every style supplied by SAS. The output appears in your working directory. Although you can apply most SAS styles to any destination, SAS supplies one or more styles that are optimized to work with the output the destination creates. For a table of suggested ODS styles for each destination, see Table 19.1 on page 727.

The table Gallery is created and is used in a subsequent DATA step to generate a list of each style for each destination. The destination link is a hyperlink.

```
ods _all_ close;

proc template;
   define table gallery;
      column libname memname style links;
      define libname ;
      blank_dups=on;
end;
define links;
   header = 'Samples';
   compute as '<a href="' || trim(style) ||'.html">HTML</a> ' ||
   '<a href="' || trim(style) ||'.pdf">PDF</a> ' ||
   '<a href="' || trim(style) ||'.rtf">RTF</a>';
        end;
end;
run;
```

The DATA step creates an index of all available styles supplied by SAS from the template store Sashelp.Tmplmst.

```
ods html5 file="/tmp/index.html";
title "Index of all styles";
data _null_;
        set sashelp.vstyle(where=(libname="SASHELP"));
        file print ods=(template='Gallery');
        put _ods_;
run;

ods html close;
```

The ODS destination statements create the output. You can add additional destinations by specifying the following statement for each destination: `ods` *`destination`* `file="&style..`*`destination-extension`*`" style=&style;`

```
%macro generateods();
options nodate;
        ods html5 file="&style..html" style=&style;
        ods pdf file="&style..pdf" style=&style;
        ods rtf file="&style..rtf" style=&style;
        title "Style is: &style";
```

The ODS NOPTITLE statement removes the procedure title.

```
ods noptitle;
```

The ODS SELECT statement selects the Variables table for the gallery.

```
ods select variables;
   proc contents data=sashelp.class;
run;
ods _all_ close;
%mend;
```

This DATA step creates a sample of each style.

```
data _null_;
   set sashelp.vstyle(where=(libname="SASHELP"));
   call symputx('style', style);
   call execute('%generateods');
run;
```

The ODS PREFERENCES statements set the ODS options back to defaults.

```
ods preferences;
```

Table of Suggested ODS Styles

The default style for ODS output in SAS Studio corresponds to the theme that SAS Studio is actively using. Depending on the theme, the default style is either Illuminate, Ignite, or HighContrast. You can use any style with any output destination. If your SAS Studio session is not actively using a theme, each destination has one or more styles that are optimized to work with the output the destination creates.

For more information, see "Customizing SAS Studio" in *SAS Studio: User's Guide*.

Table 19.1 *Recommended Styles for ODS Destinations*

Destination	Default Style	Recommended Styles
EPUB	Daisy	Daisy Moonflower[1]
ODS destination for Excel	Excel	Excel ExcelAqua ExcelHighContrast[4] ExcelIgnite[3] ExcelIlluminate[2] ExcelMarine[6] ExcelMidnight ExcelOpal ExcelUmstead[5]
PDF	Pearl	Pearl PearlJ Sapphire
RTF	RTF	RTF
TAGSETS.RTF	RTF	RTF
ODS destination for PowerPoint	PowerPointLight	PowerPointAqua PowerPointDark PowerPointHighContrast[4] PowerPointIgnite[3] PowerPointIlluminate[2] PowerPointLight ExcelMarine[6] ExcelMidnight PowerPointOpal ExcelUmstead[5]
HTML5	HTMLBlue	Default Dove EGDefault HighContrast HighContrastLarge HTMLBlue

Destination	Default Style	Recommended Styles
		HTMLEncore
		Ignite
		Illuminate
		Journal
		Journal2
		Journal3
		Plateau
		Raven
		Snow
		Statistical
ODS destination for Word	Word	Word
		WordAqua
		WordHighContrast[4]
		WordIgnite[3]
		WordIlluminate[2]
		WordMarine[6]
		WordMidnight
		WordOpal
		WordUmstead[5]

1 The Moonflower style for ODS EPUB is designed for nighttime or low-light reading.

2 The Illuminate style for Microsoft Office destinations is synchronized to the Opal report theme in SAS Visual Analytics.

3 The Ignite style for Microsoft Office destinations is synchronized to the Midnight2020 report theme in SAS Visual Analytics.

4 The HighContrast style for Microsoft Office destinations is synchronized to the HighContrast2020 report theme in SAS Visual Analytics.

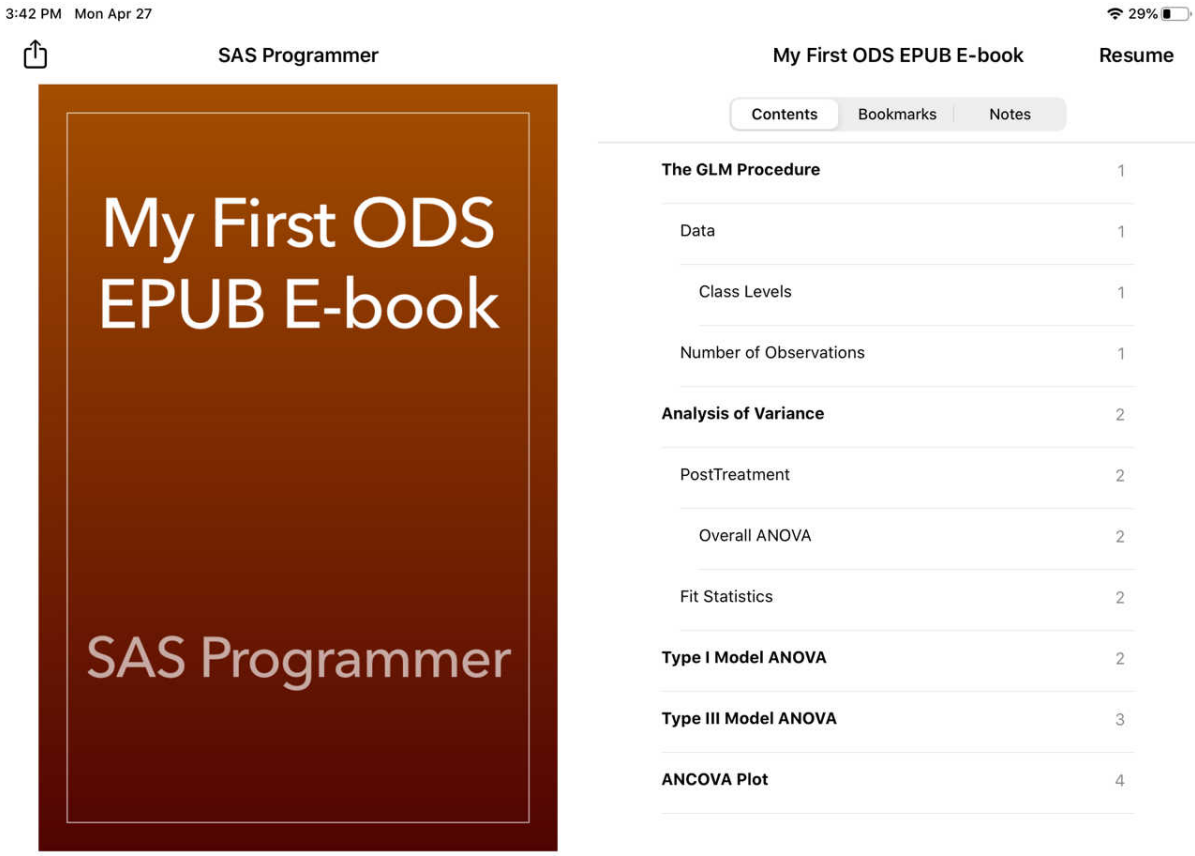
5 The Umstead style for Microsoft Office destinations is synchronized to the Umstead report theme in SAS Visual Analytics.

6 The Marine style for Microsoft Office destinations is synchronized to the Marine report theme in SAS Visual Analytics.

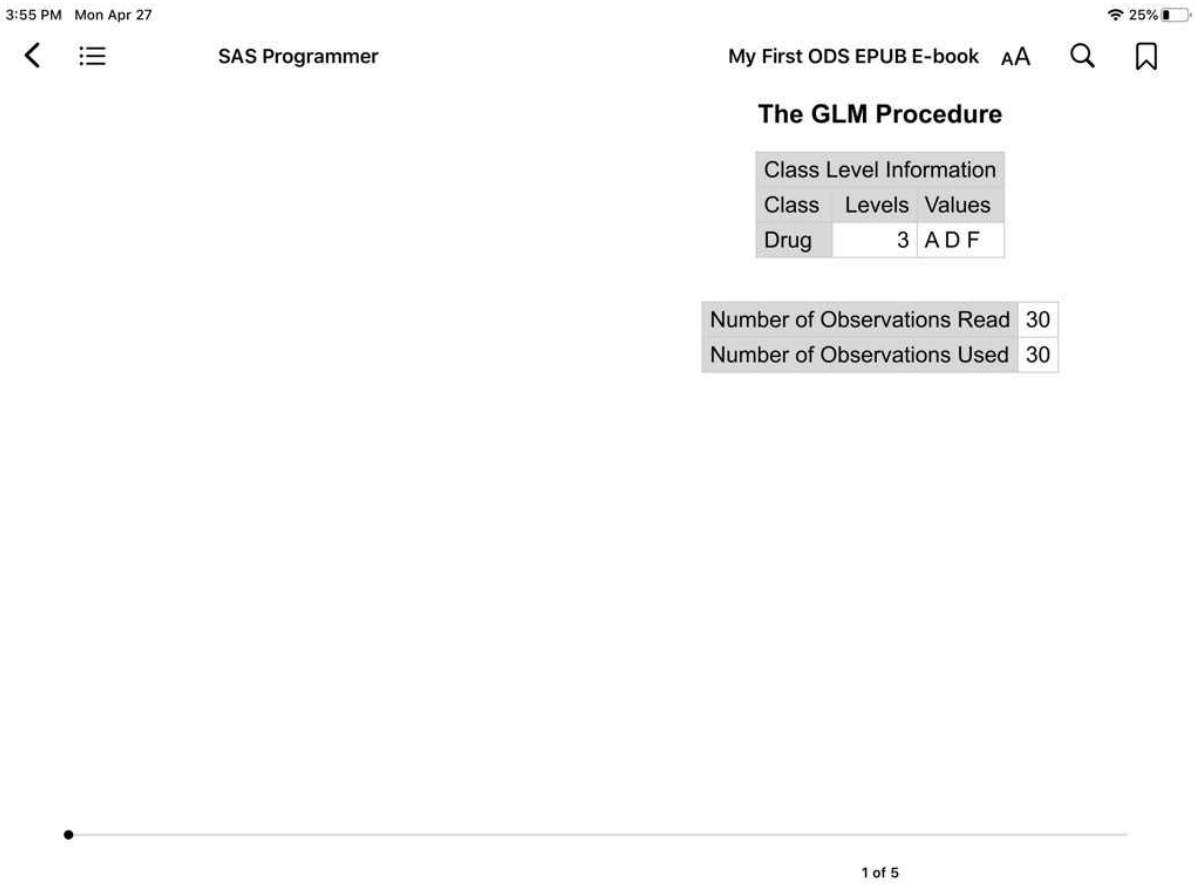
ODS Styles Gallery

EPUB Daisy Style

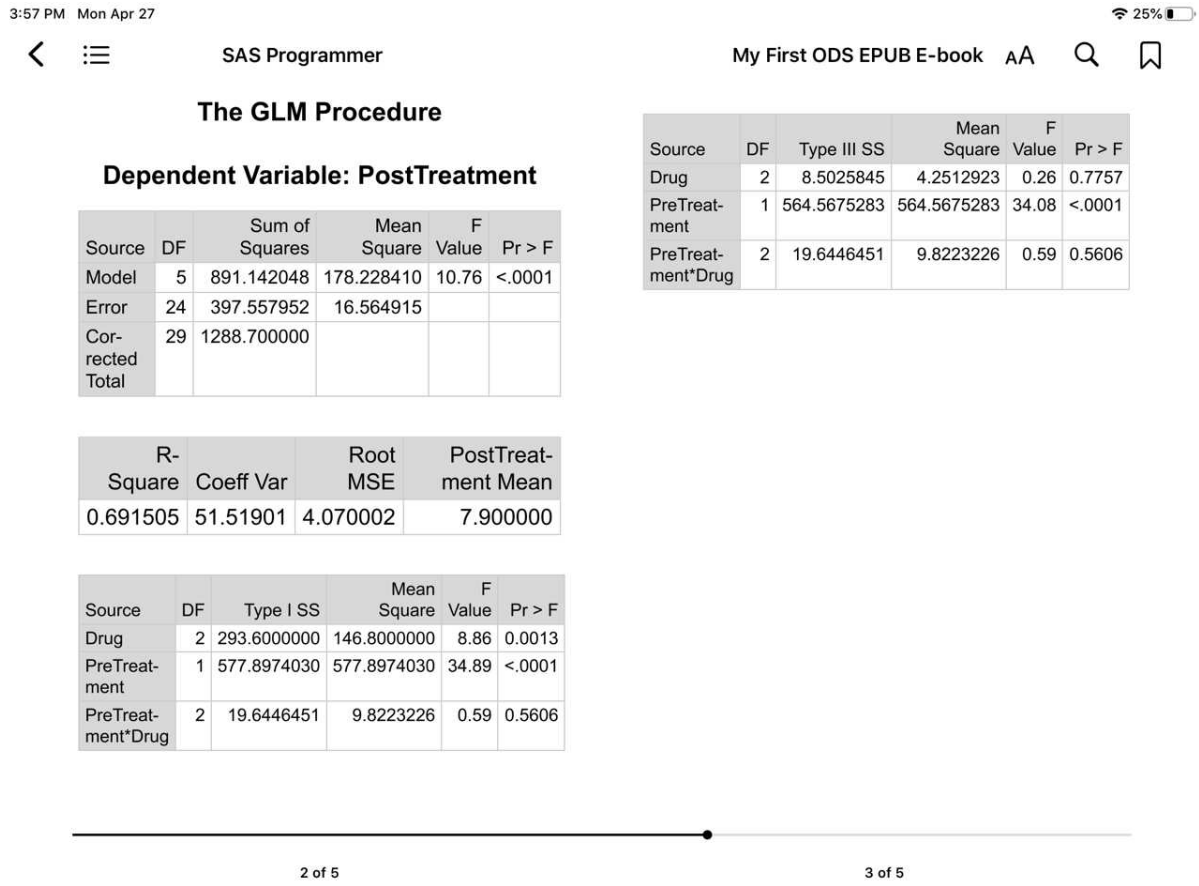
Output 19.1 *EPUB Book Title Page*



Output 19.2 *EPUB Book, Page 1*



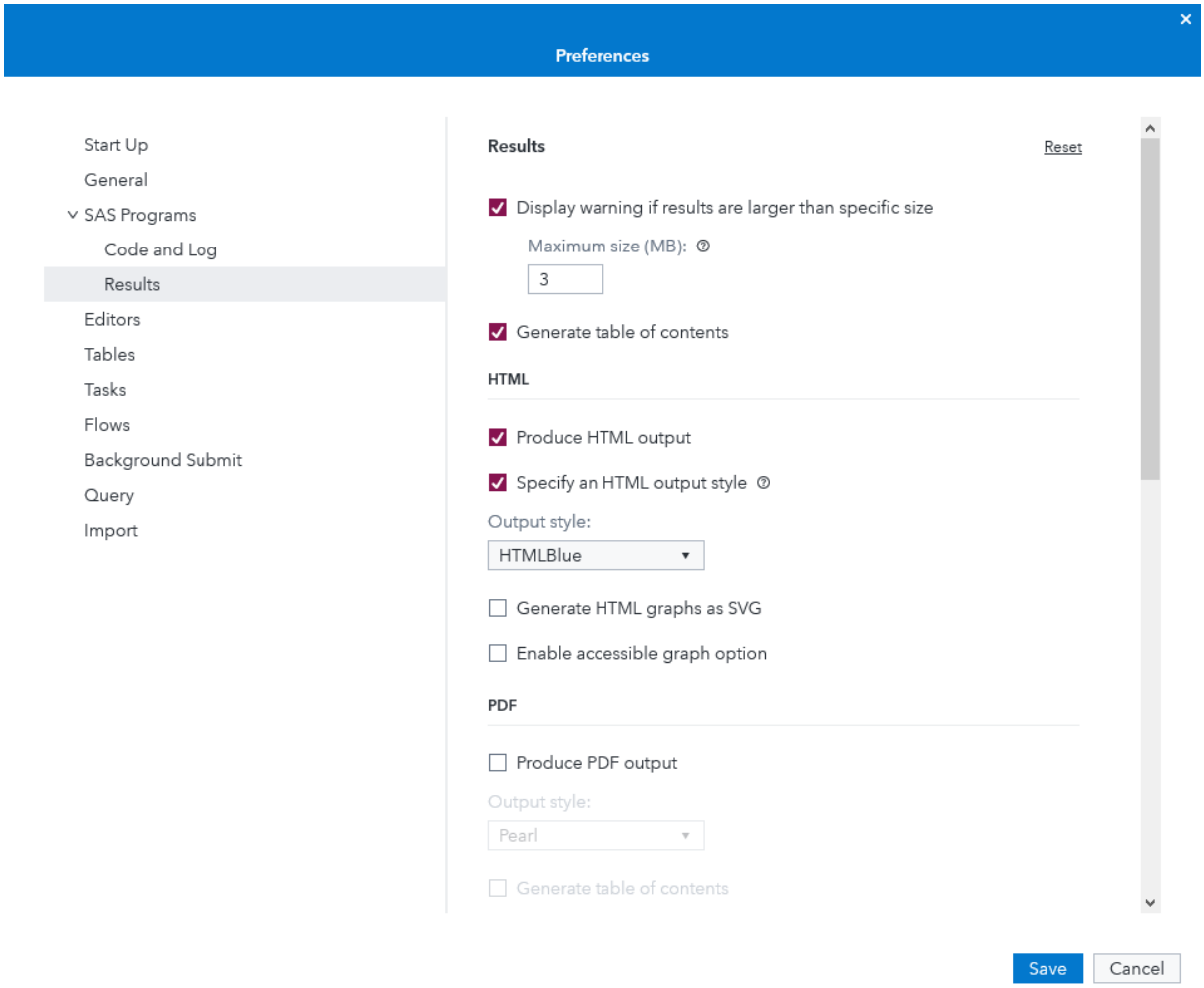
Output 19.3 *EPUB Book, Page 2*



HTML Styles

In SAS Studio, you do not need to specify the STYLE= option. You can go to **Options** ⇨ **Preferences** ⇨ **SAS Programs** ⇨ **Results** and change the style from the drop-down list for your selected destination.

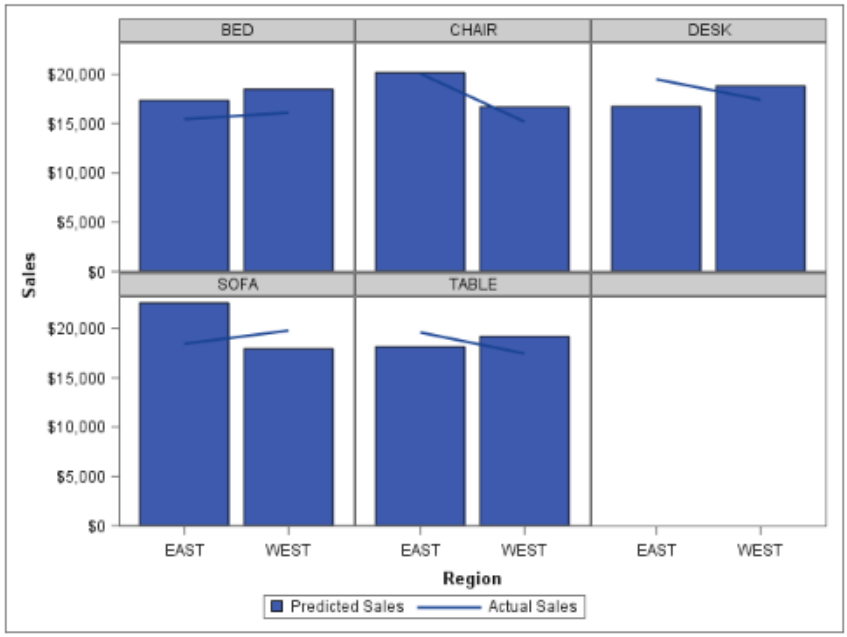
Figure 19.2 Changing the HTML Style with the Preferences in SAS Studio

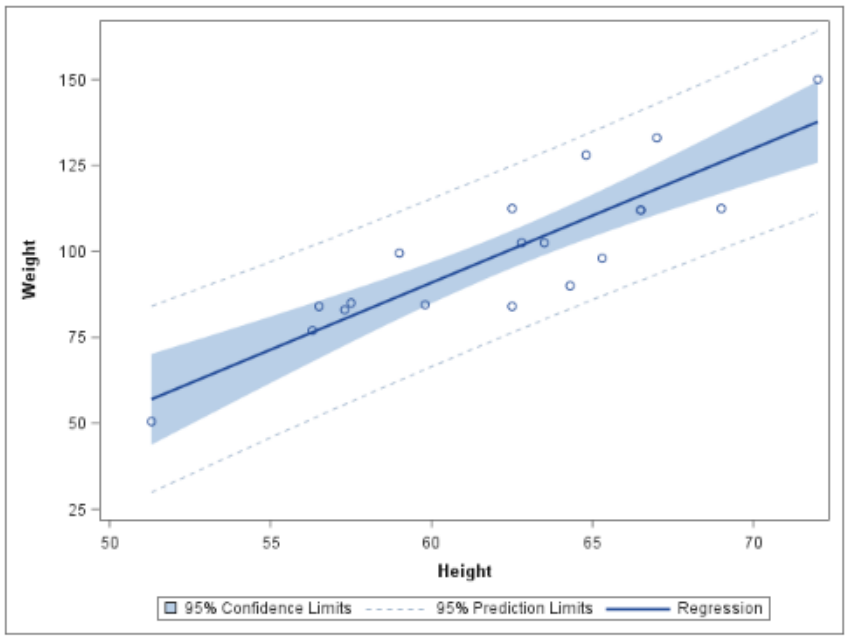


Output 19.4 Daisy Style: Table

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

Output 19.5 *Daisy Style: Graphs*

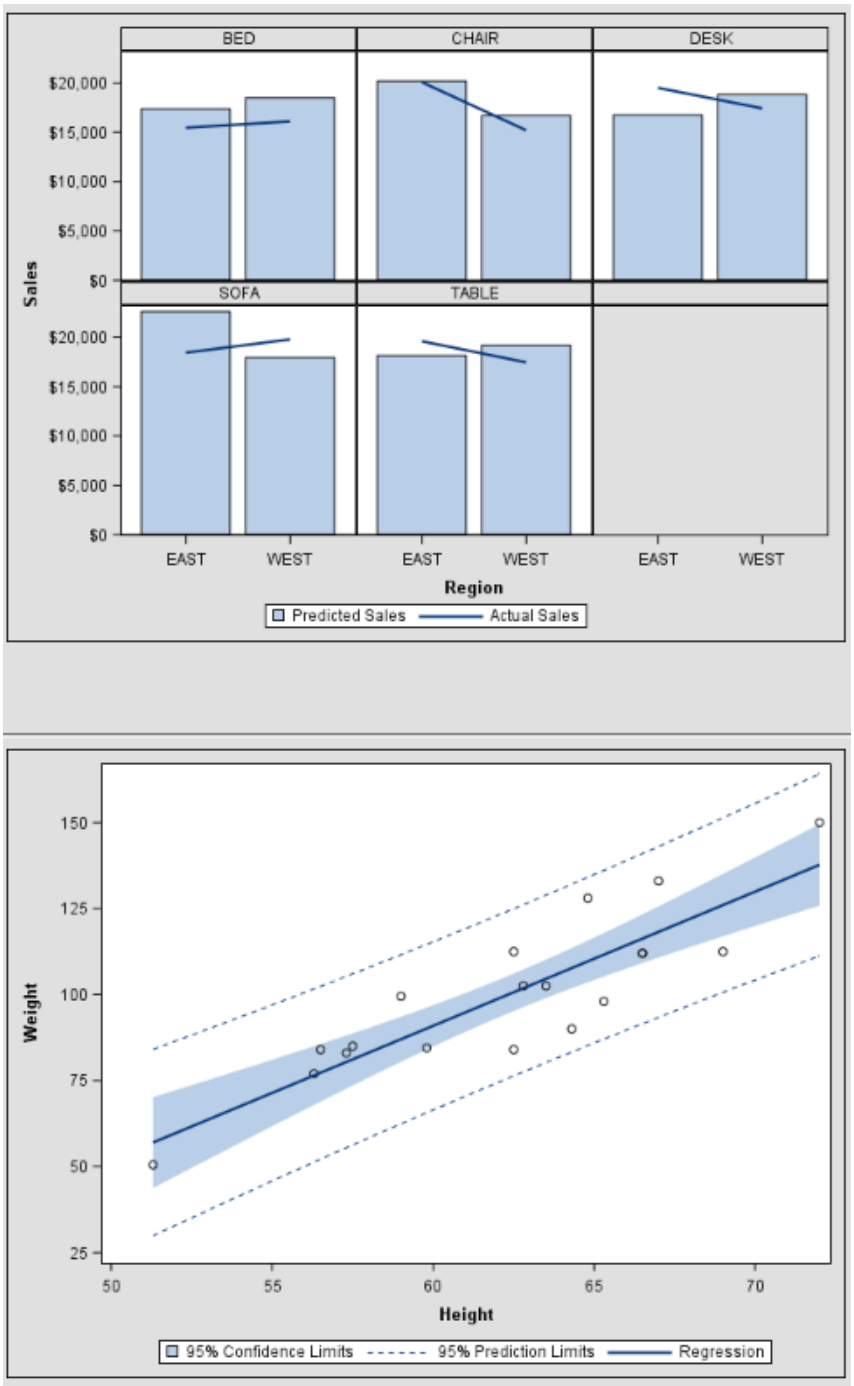




Output 19.6 *Default Style: Table*

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

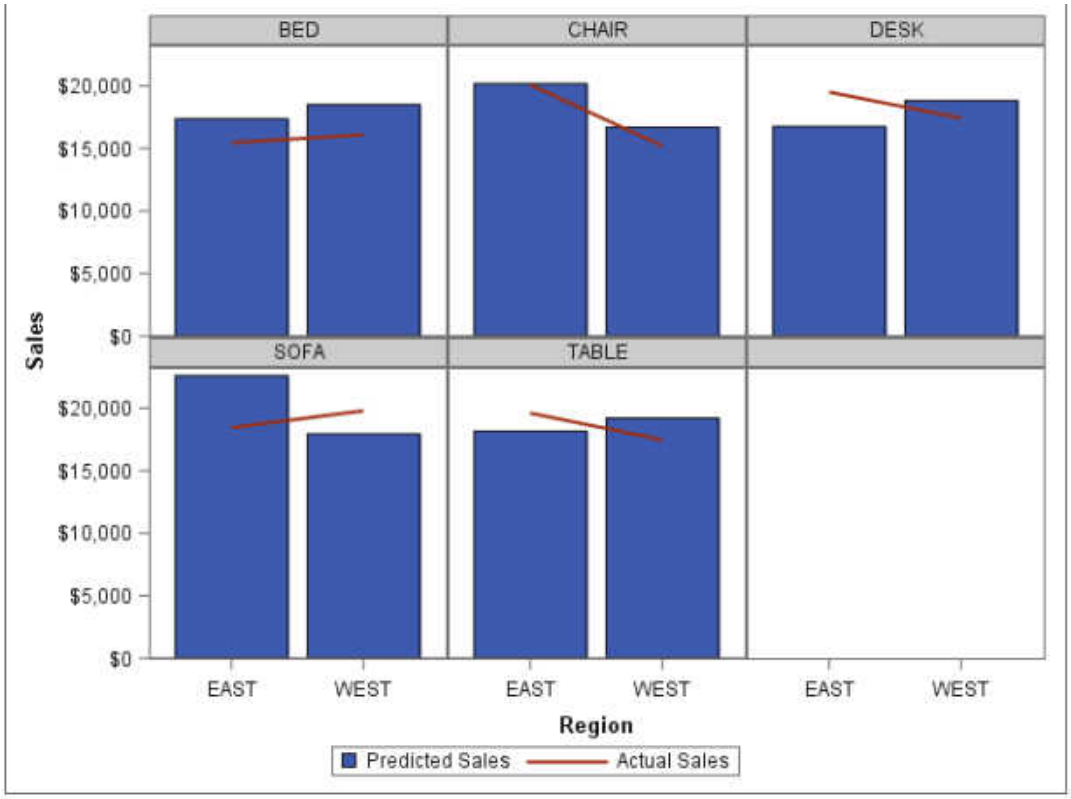
Output 19.7 *Default Style: Graphs*

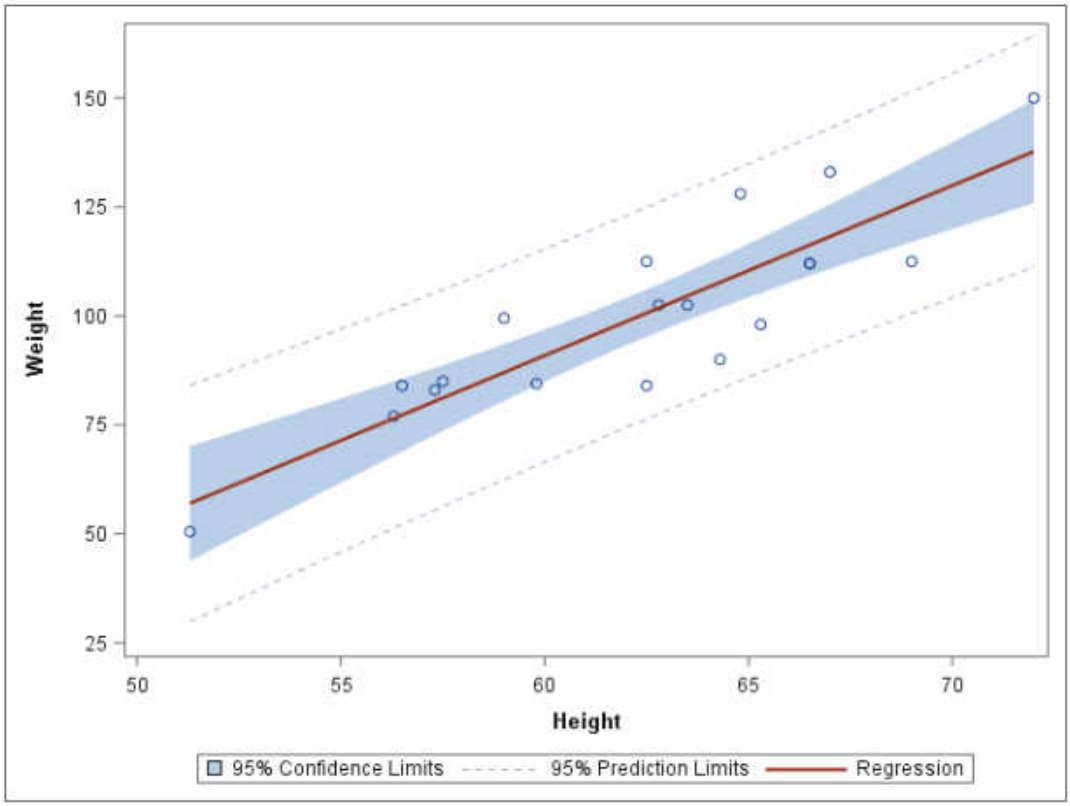


Output 19.8 *Dove Style: Table*

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

Output 19.9 *Dove Style: Graphs*

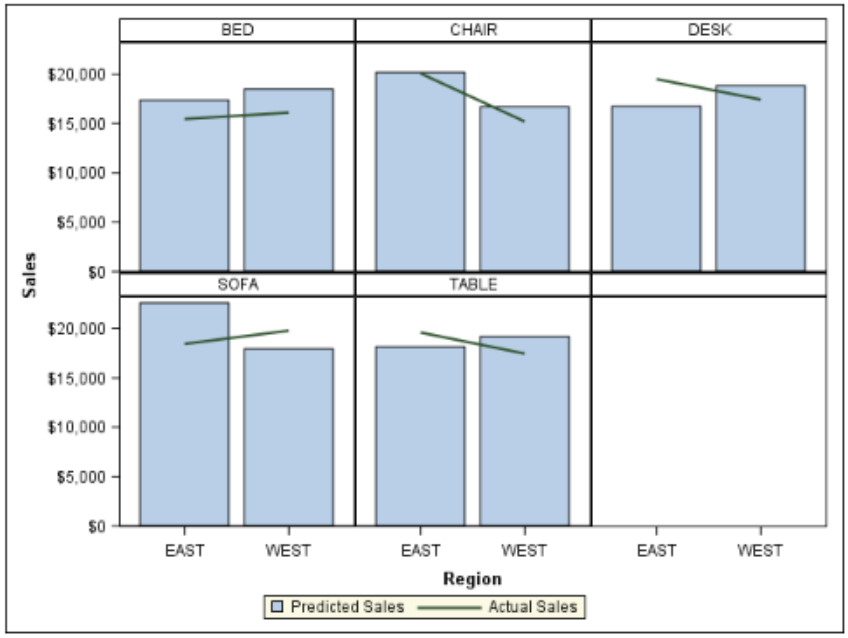


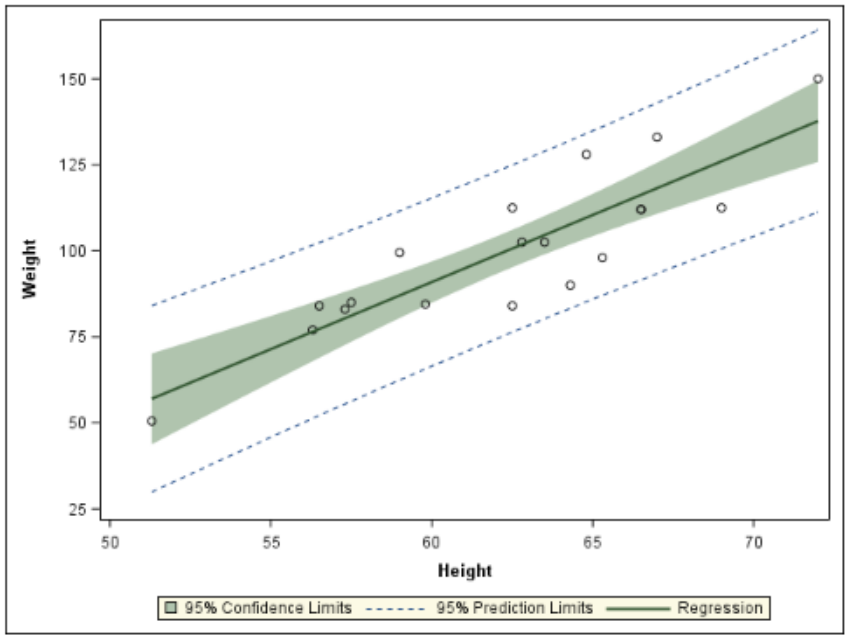


Output 19.10 *EGDefault Style: Table*

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

Output 19.11 *EGDefault Style: Graphs*





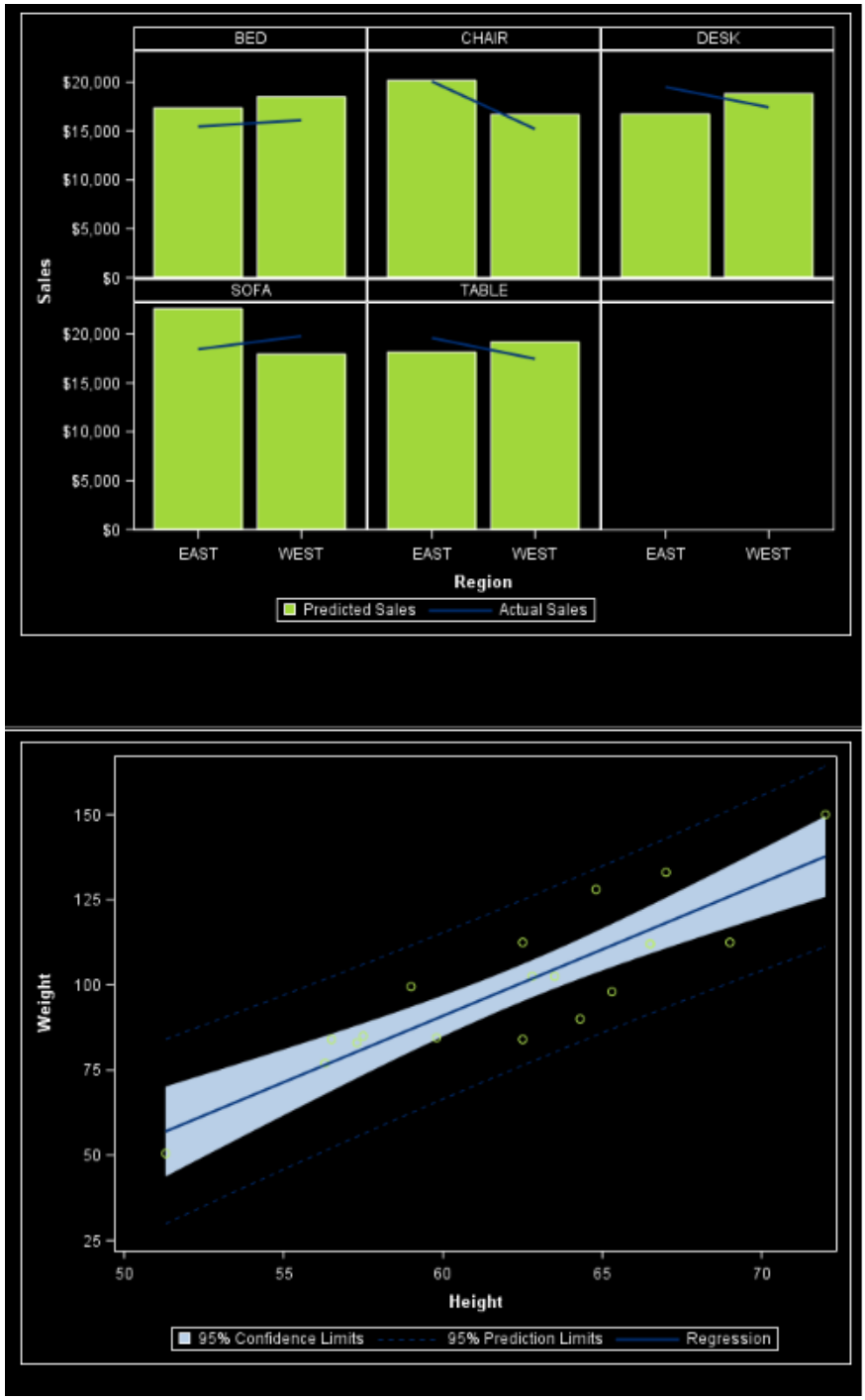
The following style is suitable for generating accessible output when used with the ODS HTML5 destination.

Output 19.12 HighContrast Style: Table

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

The following style is suitable for generating accessible output when used with the ODS HTML5 destination.

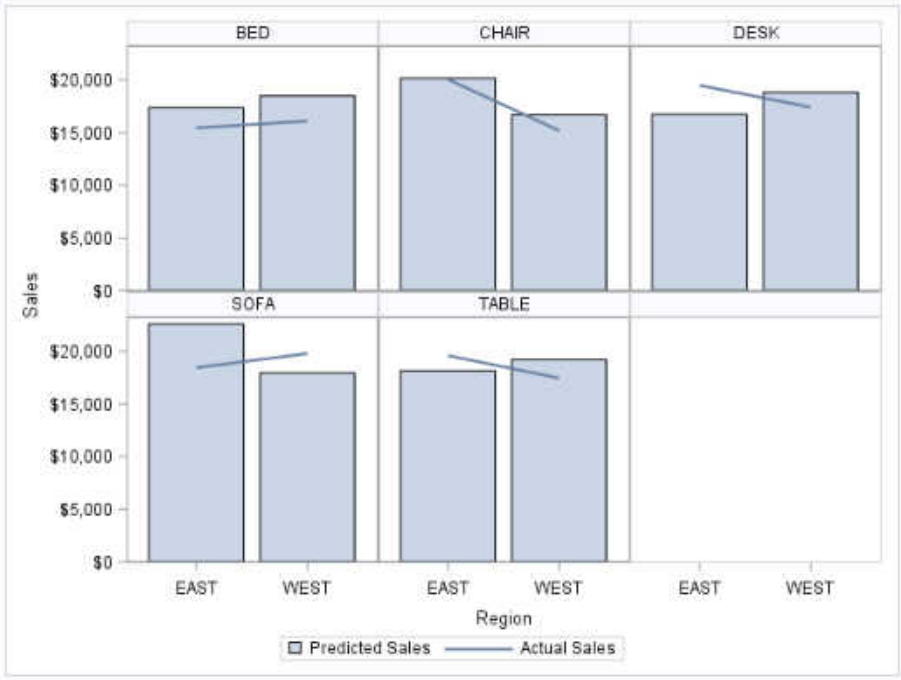
Output 19.13 *HighContrast Style: Graphs*

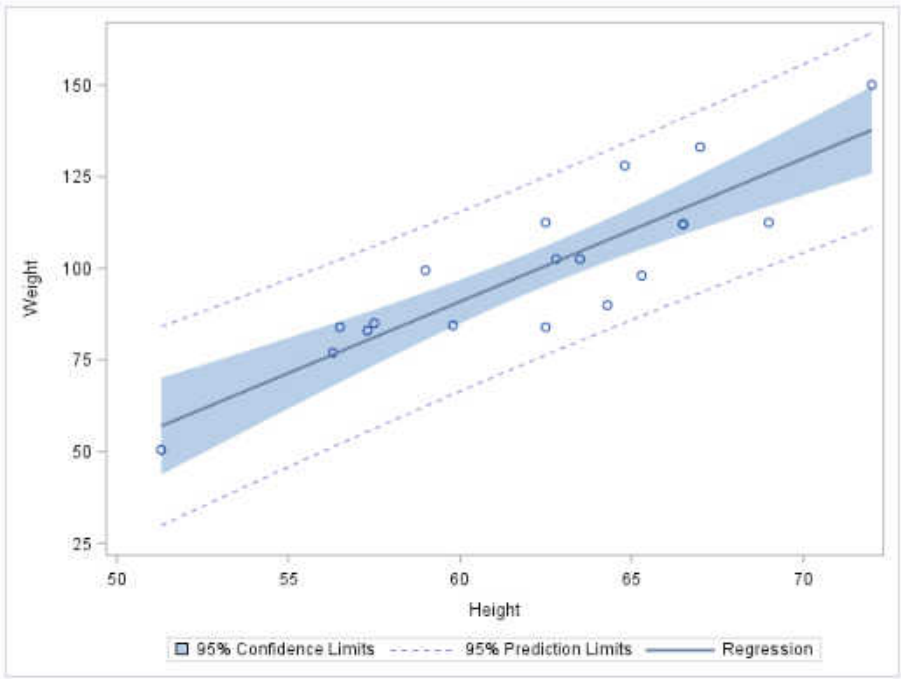


Output 19.14 *HTMLBlue Style: Table*

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

Output 19.15 *HTMLBlue Style: Graphs*

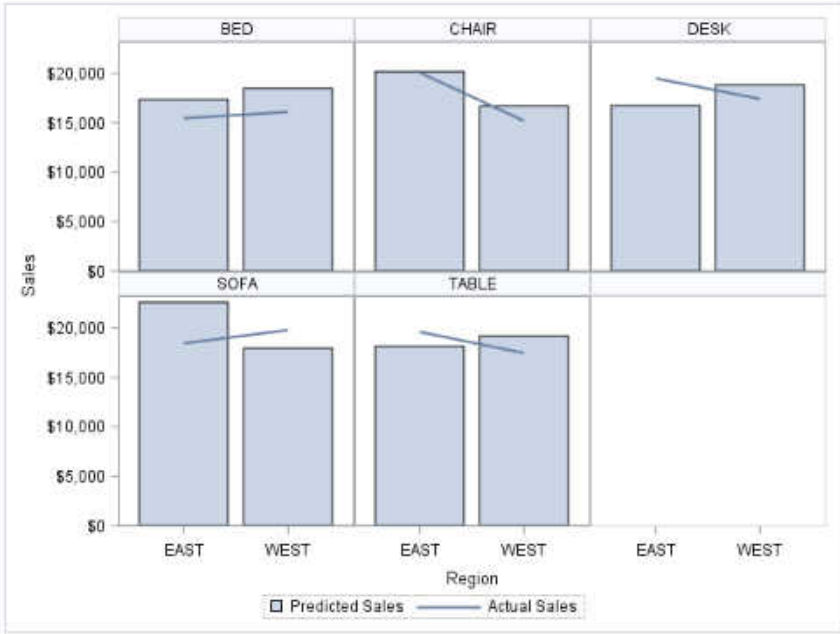


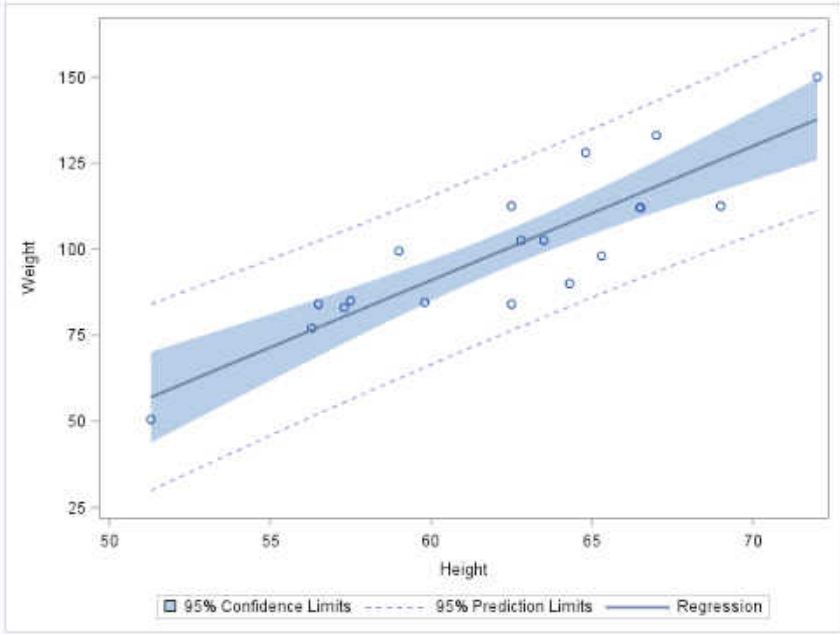


Output 19.16 *HTMLEncore: Table*

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

Output 19.17 *HTMLEncore: Graphs 1*

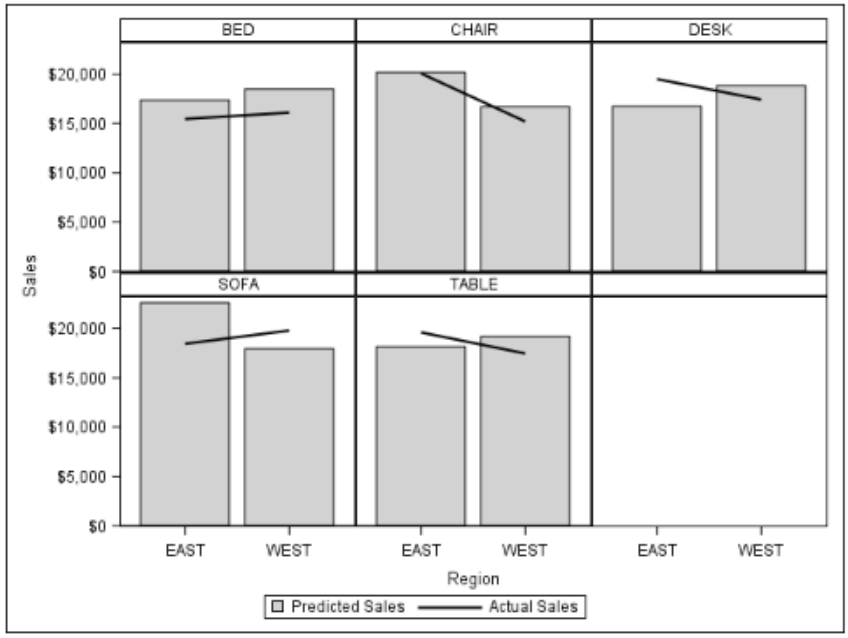


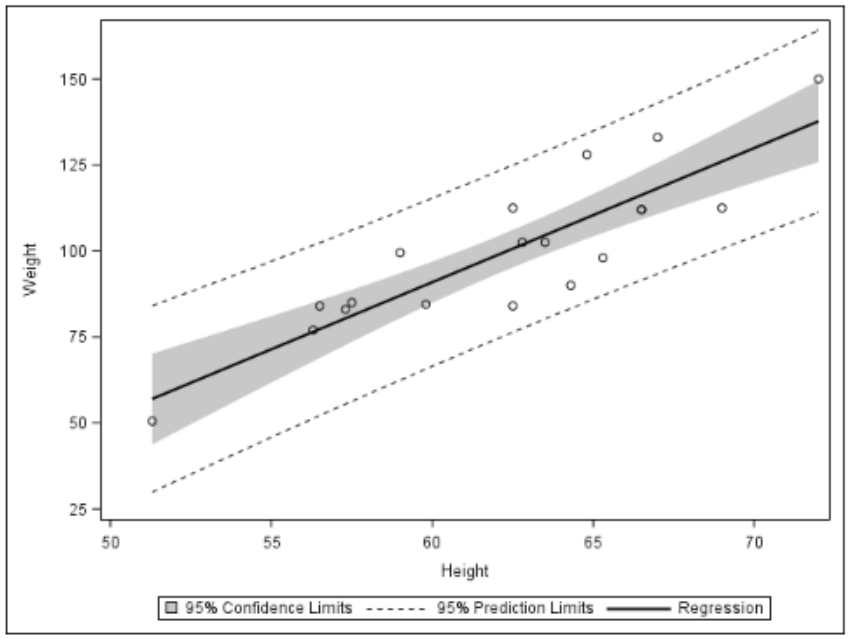


Output 19.18 *Journal Style: Table*

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

Output 19.19 *Journal Style: Graphs*

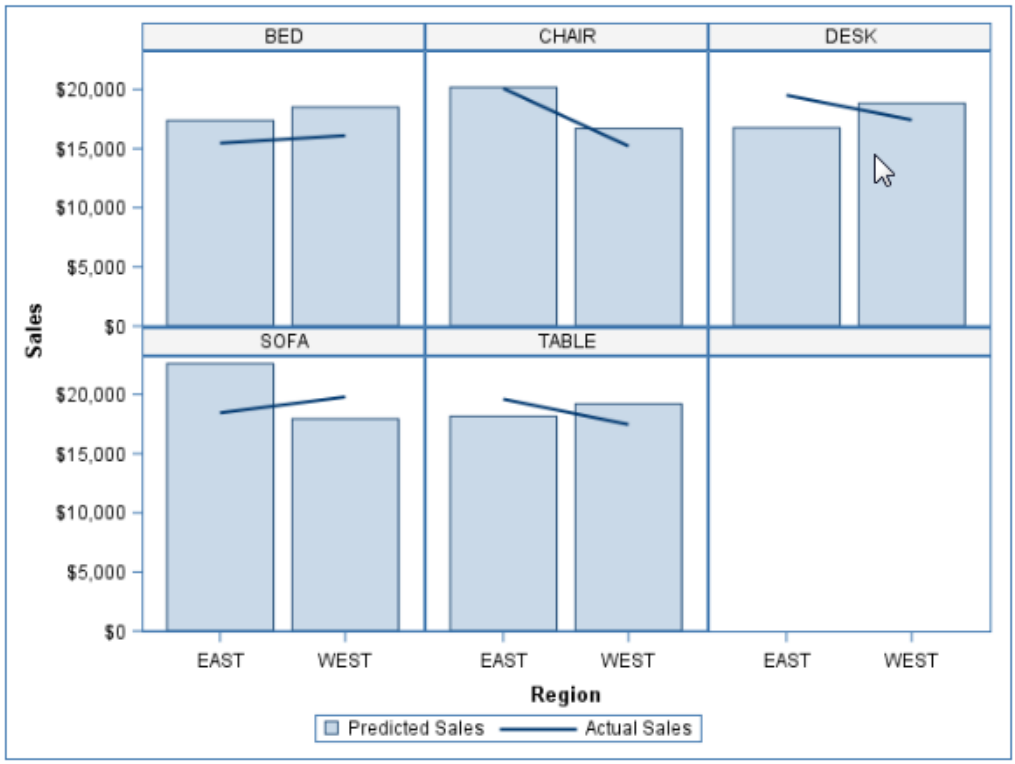


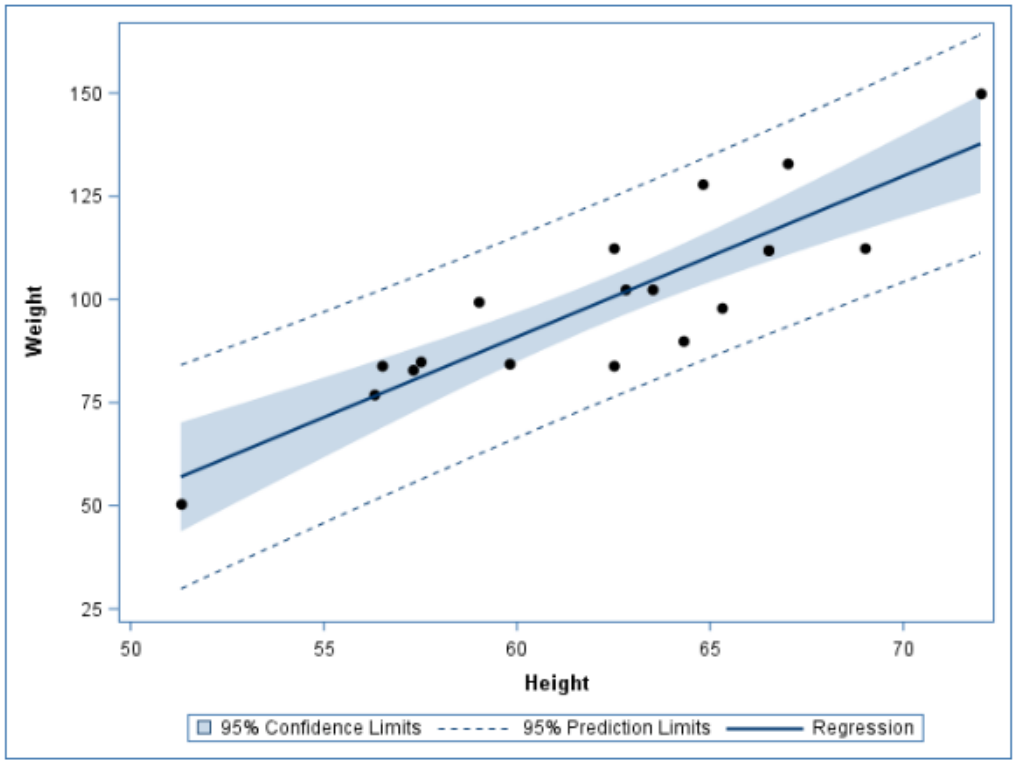


Output 19.20 *Plateau Style: Table*

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

Output 19.21 *Plateau Style: Graphs*

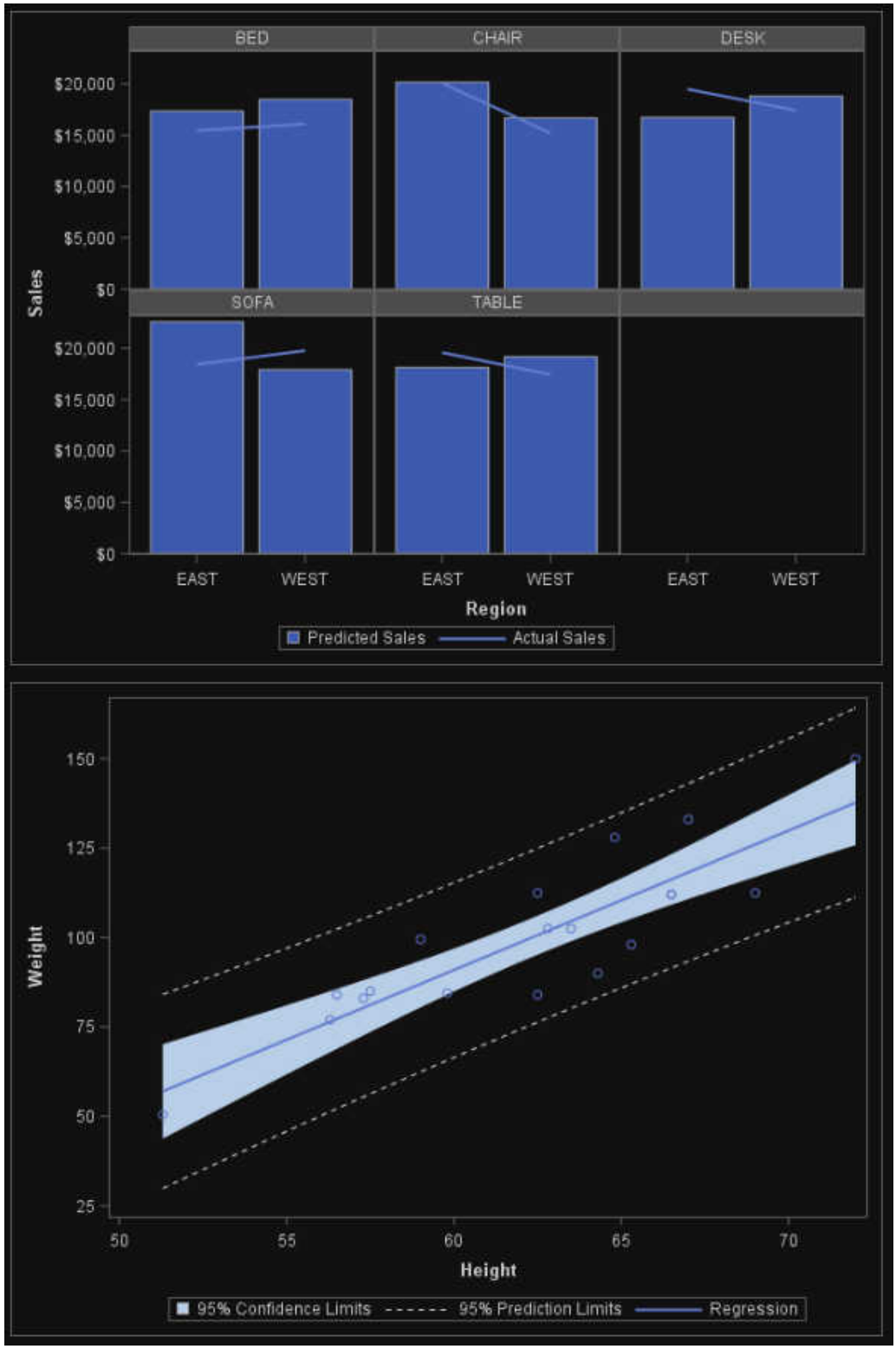




Output 19.22 *Raven Style: Table*

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

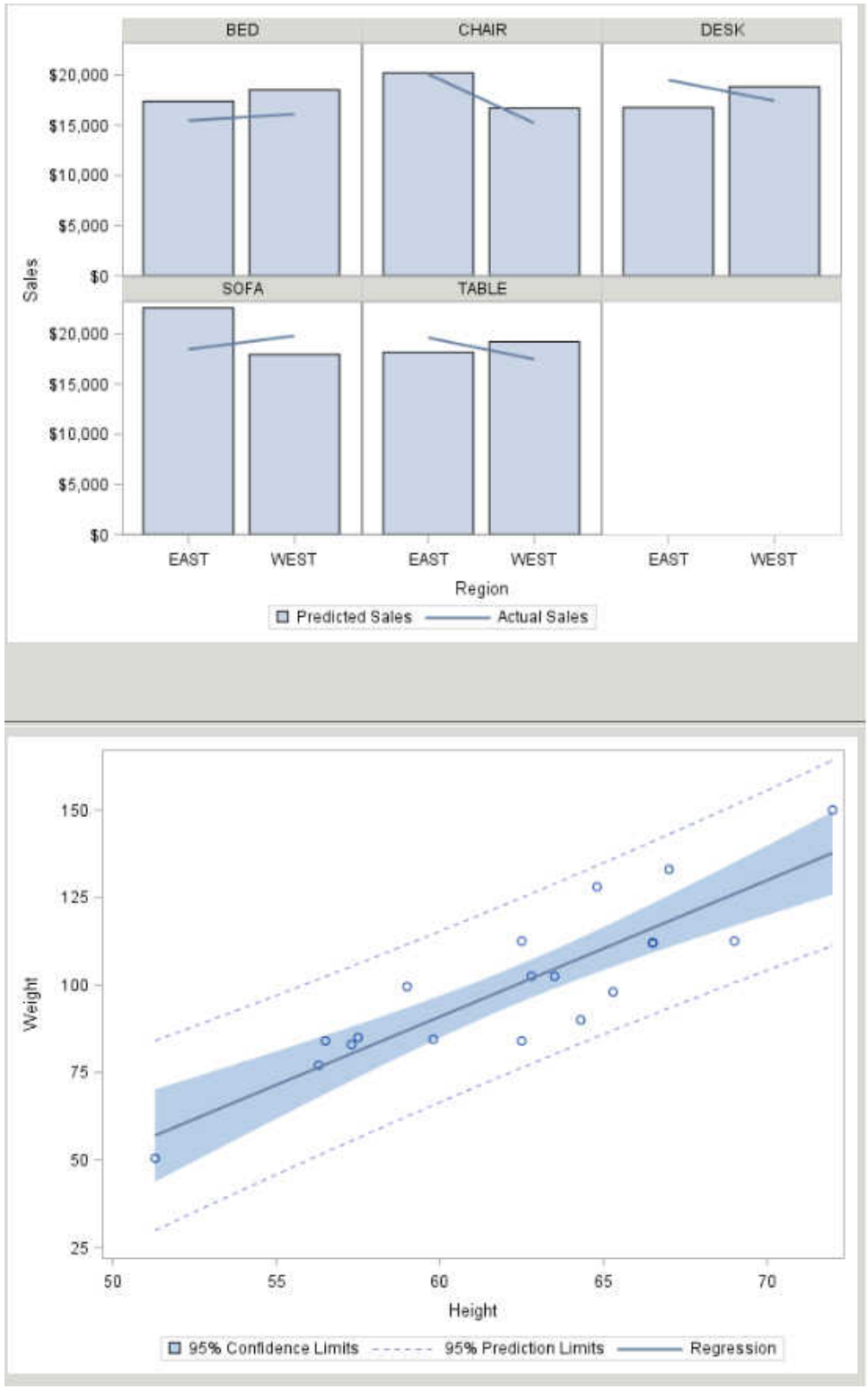
Output 19.23 *Raven Style: Graphs*



Output 19.24 *Statistical Style: Table*

Obs	PRODUCT	REGION	ACTUAL	PREDICT
1	SOFA	EAST	$925.00	$850.00
2	SOFA	EAST	$999.00	$297.00
3	SOFA	EAST	$608.00	$846.00
4	SOFA	EAST	$642.00	$533.00
5	SOFA	EAST	$656.00	$646.00
6	SOFA	EAST	$948.00	$486.00
7	SOFA	EAST	$612.00	$717.00
8	SOFA	EAST	$114.00	$564.00

Output 19.25 *Statistical Style: Graphs*



PDF Styles

Pearl is the default style for PDF output.

Output 19.26 *Pearl Style*

Pearl Style

Obs	Make	Model	Type	Origin	Invoice
1	Acura	MDX	SUV	Asia	$33,337
2	Acura	RSX Type S 2dr	Sedan	Asia	$21,761
3	Acura	TSX 4dr	Sedan	Asia	$24,647
4	Acura	TL 4dr	Sedan	Asia	$30,299
5	Acura	3.5 RL 4dr	Sedan	Asia	$39,014
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	$41,100
7	Acura	NSX coupe 2dr manual S	Sports	Asia	$79,978
8	Audi	A4 1.8T 4dr	Sedan	Europe	$23,508
9	Audi	A41.8T convertible 2dr	Sedan	Europe	$32,506
10	Audi	A4 3.0 4dr	Sedan	Europe	$28,846
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	$30,366
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	$31,388
13	Audi	A6 3.0 4dr	Sedan	Europe	$33,129
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	$35,992
15	Audi	A4 3.0 convertible 2dr	Sedan	Europe	$38,325

Output 19.27 *Sapphire Style*

Sapphire Style

Obs	Make	Model	Type	Origin	Invoice
1	Acura	MDX	SUV	Asia	$33,337
2	Acura	RSX Type S 2dr	Sedan	Asia	$21,761
3	Acura	TSX 4dr	Sedan	Asia	$24,647
4	Acura	TL 4dr	Sedan	Asia	$30,299
5	Acura	3.5 RL 4dr	Sedan	Asia	$39,014
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	$41,100
7	Acura	NSX coupe 2dr manual S	Sports	Asia	$79,978
8	Audi	A4 1.8T 4dr	Sedan	Europe	$23,508
9	Audi	A41.8T convertible 2dr	Sedan	Europe	$32,506
10	Audi	A4 3.0 4dr	Sedan	Europe	$28,846
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	$30,366
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	$31,388
13	Audi	A6 3.0 4dr	Sedan	Europe	$33,129
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	$35,992
15	Audi	A4 3.0 convertible 2dr	Sedan	Europe	$38,325

Styles for the ODS Destination for Excel

Output 19.28 *Excel Style*

	A	B	C	D	E
1	Country=CANADA				
2					
3	1993			Total Predicted Sales	Total Actual Sales
4	Region	Division	Product type		
5	EAST	CONSUMER	FURNITURE	$11,081	$12,483
6			OFFICE	$21,939	$16,991
7		EDUCATION	FURNITURE	$12,972	$14,467
8			OFFICE	$16,434	$20,189
9		Division Total		$62,426	$64,130
10	WEST	Division	Product type		
11		CONSUMER	FURNITURE	$10,286	$10,380
12			OFFICE	$16,042	$16,371
13		EDUCATION	FURNITURE	$12,816	$11,234
14			OFFICE	$17,759	$18,905
15		Division Total		$56,903	$56,890
16	Grand Total			$119,329	$121,020

Output 19.29 *ExcelAqua and ExcelIlluminate Styles*

	A	B	C	D	E
1	Country=CANADA				
2					
3	1993			Total Predicted Sales	Total Actual Sales
4	Region	Division	Product type		
5	EAST	CONSUMER	FURNITURE	$11,081	$12,483
6			OFFICE	$21,939	$16,991
7		EDUCATION	FURNITURE	$12,972	$14,467
8			OFFICE	$16,434	$20,189
9		Division Total		$62,426	$64,130
10	WEST	Division	Product type		
11		CONSUMER	FURNITURE	$10,286	$10,380
12			OFFICE	$16,042	$16,371
13		EDUCATION	FURNITURE	$12,816	$11,234
14			OFFICE	$17,759	$18,905
15		Division Total		$56,903	$56,890
16	Grand Total			$119,329	$121,020

Output 19.30 *ExcelIgnite Style*

	A	B	C	D	E
1	Country=CANADA				
2					
3	1993			Total Predicted Sales	Total Actual Sales
4	Region	Division	Product type		
5	EAST	CONSUMER	FURNITURE	$11,081	$12,483
6			OFFICE	$21,939	$16,991
7		EDUCATION	FURNITURE	$12,972	$14,467
8			OFFICE	$16,434	$20,189
9		Division Total		$62,426	$64,130
10	WEST	Division	Product type		
11		CONSUMER	FURNITURE	$10,286	$10,380
12			OFFICE	$16,042	$16,371
13		EDUCATION	FURNITURE	$12,816	$11,234
14			OFFICE	$17,759	$18,905
15		Division Total		$56,903	$56,890
16	Grand Total			$119,329	$121,020

The following style is recommended for generating accessible table output when used with the ODS Excel destination.

Output 19.31 *ExcelHighContrast Style*

	A	B	C	D	E
1	Country=CANADA				
2					
3	1993			Total Predicted Sales	Total Actual Sales
4	Region	Division	Product type		
5	EAST	CONSUMER	FURNITURE	$11,081	$12,483
6			OFFICE	$21,939	$16,991
7		EDUCATION	FURNITURE	$12,972	$14,467
8			OFFICE	$16,434	$20,189
9		Division Total		$62,426	$64,130
10	WEST	Division	Product type		
11		CONSUMER	FURNITURE	$10,286	$10,380
12			OFFICE	$16,042	$16,371
13		EDUCATION	FURNITURE	$12,816	$11,234
14			OFFICE	$17,759	$18,905
15		Division Total		$56,903	$56,890
16	Grand Total			$119,329	$121,020

Output 19.32 *ExcelMarineStyle*

	A	B	C	D	E
1	Country=CANADA				
2					
3	1993			Total Predicted Sales	Total Actual Sales
4	Region	Division	Product type		
5	EAST	CONSUMER	FURNITURE	$11,081	$12,483
6			OFFICE	$21,939	$16,991
7		EDUCATION	FURNITURE	$12,972	$14,467
8			OFFICE	$16,434	$20,189
9		Division Total		$62,426	$64,130
10	WEST	Division	Product type		
11		CONSUMER	FURNITURE	$10,286	$10,380
12			OFFICE	$16,042	$16,371
13		EDUCATION	FURNITURE	$12,816	$11,234
14			OFFICE	$17,759	$18,905
15		Division Total		$56,903	$56,890
16	Grand Total			$119,329	$121,020

Output 19.33 *ExcelMidnight Style*

	A	B	C	D	E
1	Country=CANADA				
2					
3	1993			Total Predicted Sales	Total Actual Sales
4	Region	Division	Product type		
5	EAST	CONSUMER	FURNITURE	$11,081	$12,483
6			OFFICE	$21,939	$16,991
7		EDUCATION	FURNITURE	$12,972	$14,467
8			OFFICE	$16,434	$20,189
9		Division Total		$62,426	$64,130
10	WEST	Division	Product type		
11		CONSUMER	FURNITURE	$10,286	$10,380
12			OFFICE	$16,042	$16,371
13		EDUCATION	FURNITURE	$12,816	$11,234
14			OFFICE	$17,759	$18,905
15		Division Total		$56,903	$56,890
16	Grand Total			$119,329	$121,020

Output 19.34 *ExcelOpal Style*

	A	B	C	D	E
1	Country=CANADA				
2					
3	1993			Total Predicted Sales	Total Actual Sales
4	Region	Division	Product type		
5	EAST	CONSUMER	FURNITURE	$11,081	$12,483
6			OFFICE	$21,939	$16,991
7		EDUCATION	FURNITURE	$12,972	$14,467
8			OFFICE	$16,434	$20,189
9		Division Total		$62,426	$64,130
10	WEST	Division	Product type		
11		CONSUMER	FURNITURE	$10,286	$10,380
12			OFFICE	$16,042	$16,371
13		EDUCATION	FURNITURE	$12,816	$11,234
14			OFFICE	$17,759	$18,905
15		Division Total		$56,903	$56,890
16	Grand Total			$119,329	$121,020

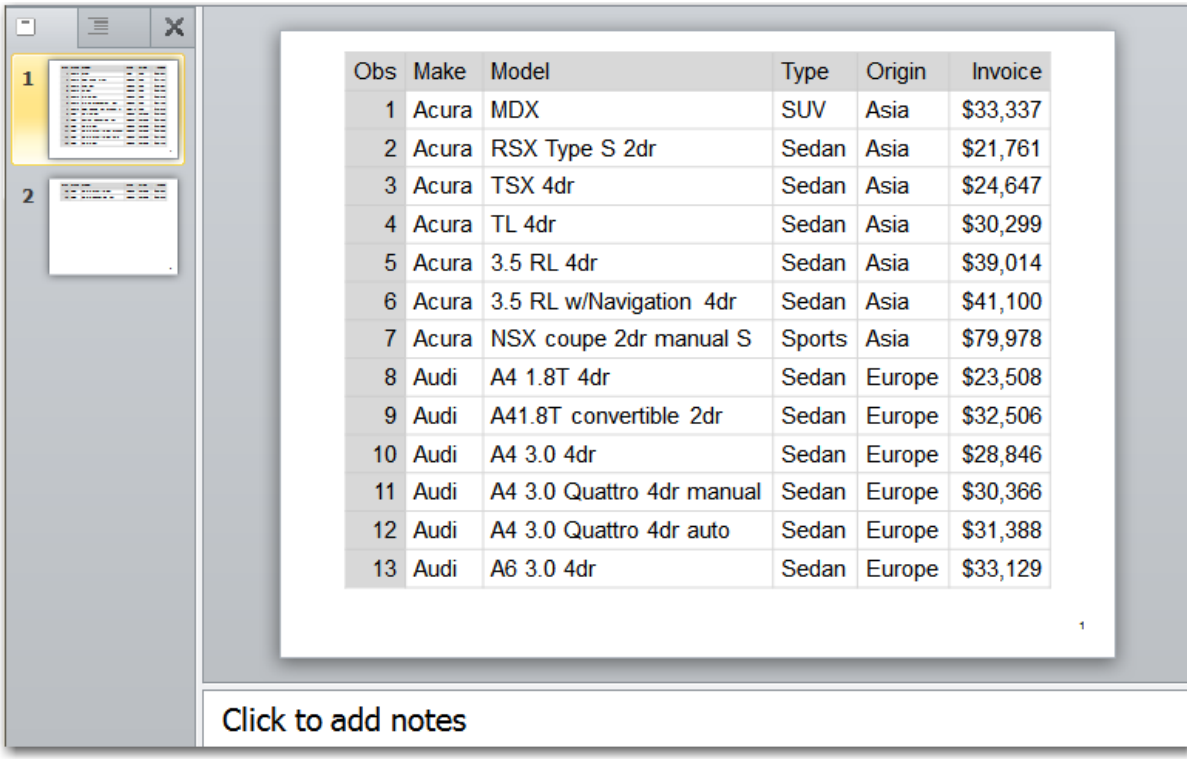
Output 19.35 *ExcelUmstead Style*

	A	B	C	D	E
1	Country=CANADA				
2					
3	1993			Total Predicted Sales	Total Actual Sales
4	Region	Division	Product type		
5	EAST	CONSUMER	FURNITURE	$11,081	$12,483
6			OFFICE	$21,939	$16,991
7		EDUCATION	FURNITURE	$12,972	$14,467
8			OFFICE	$16,434	$20,189
9		Division Total		$62,426	$64,130
10	WEST	Division	Product type		
11		CONSUMER	FURNITURE	$10,286	$10,380
12			OFFICE	$16,042	$16,371
13		EDUCATION	FURNITURE	$12,816	$11,234
14			OFFICE	$17,759	$18,905
15		Division Total		$56,903	$56,890
16	Grand Total			$119,329	$121,020

Styles for the ODS Destination for PowerPoint

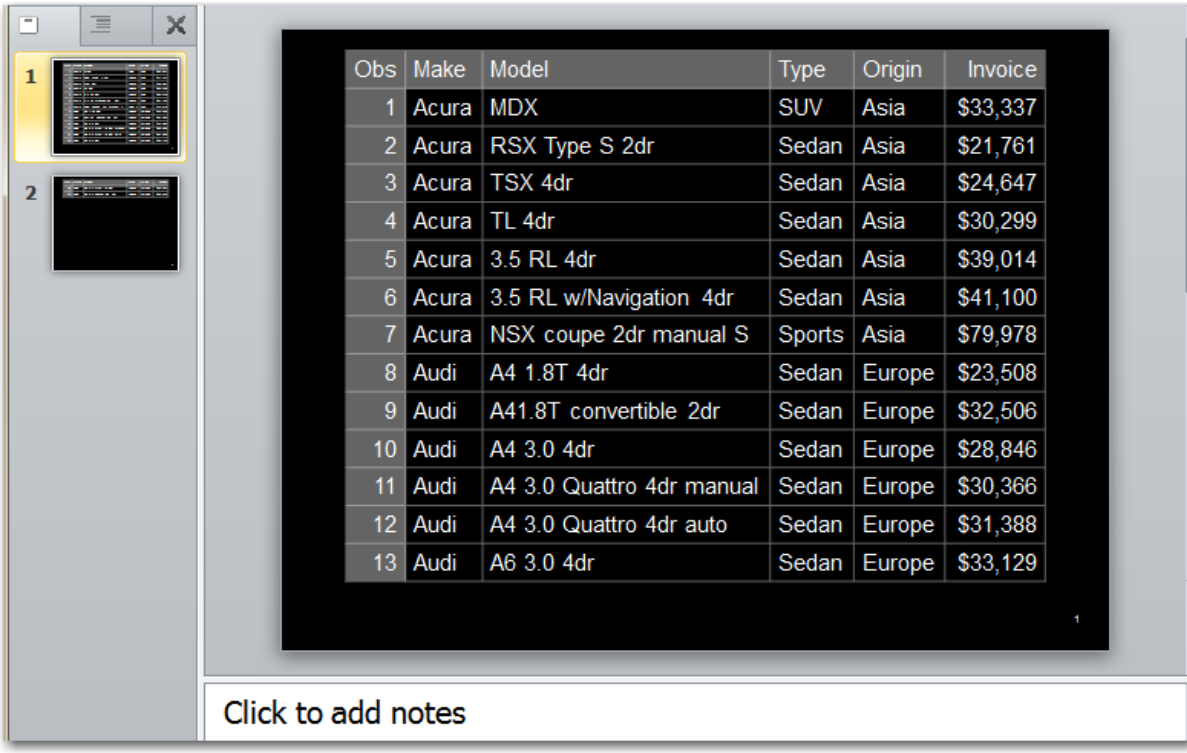
PowerPointLight is the default style for output created by the ODS destination for PowerPoint.

Output 19.36 PowerPointLight Style



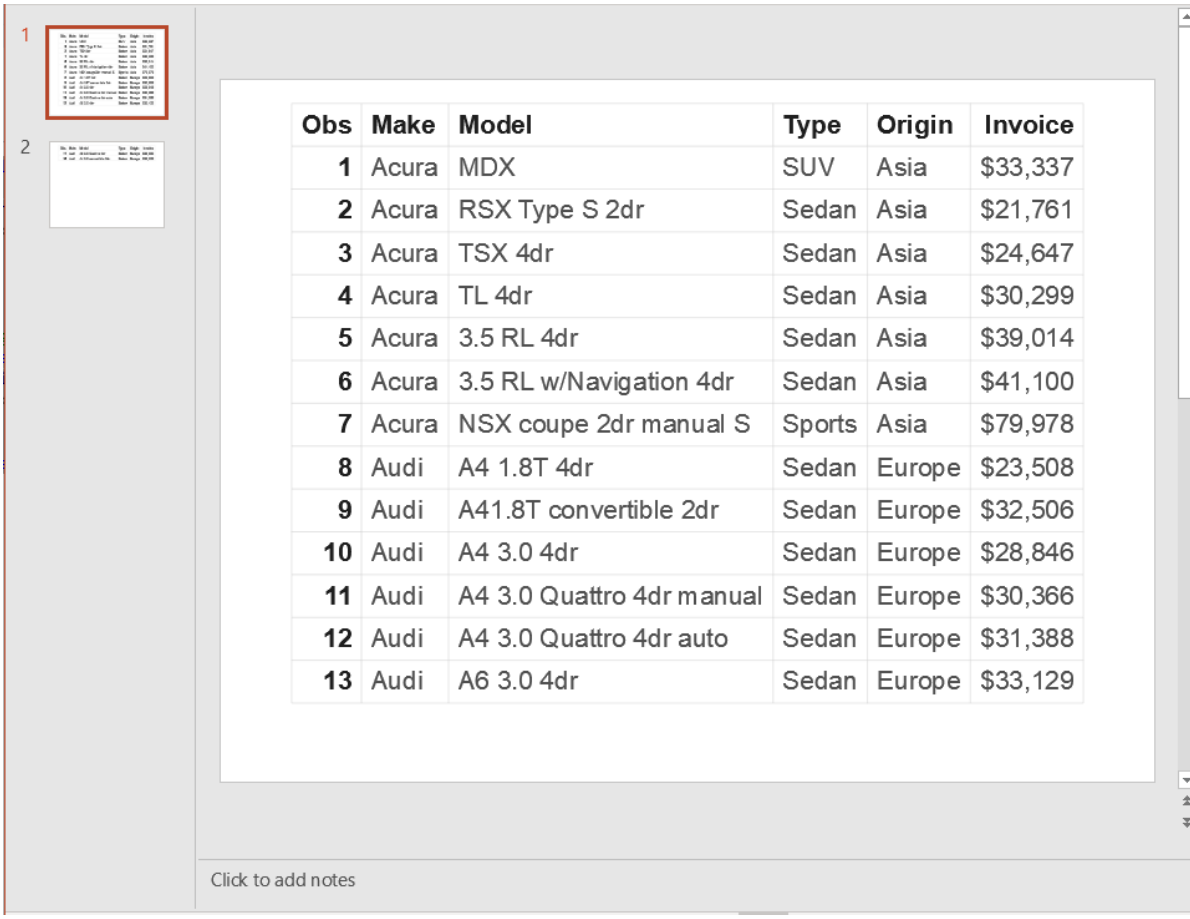
Obs	Make	Model	Type	Origin	Invoice
1	Acura	MDX	SUV	Asia	$33,337
2	Acura	RSX Type S 2dr	Sedan	Asia	$21,761
3	Acura	TSX 4dr	Sedan	Asia	$24,647
4	Acura	TL 4dr	Sedan	Asia	$30,299
5	Acura	3.5 RL 4dr	Sedan	Asia	$39,014
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	$41,100
7	Acura	NSX coupe 2dr manual S	Sports	Asia	$79,978
8	Audi	A4 1.8T 4dr	Sedan	Europe	$23,508
9	Audi	A41.8T convertible 2dr	Sedan	Europe	$32,506
10	Audi	A4 3.0 4dr	Sedan	Europe	$28,846
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	$30,366
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	$31,388
13	Audi	A6 3.0 4dr	Sedan	Europe	$33,129

Output 19.37 PowerPointDark Style



Obs	Make	Model	Type	Origin	Invoice
1	Acura	MDX	SUV	Asia	$33,337
2	Acura	RSX Type S 2dr	Sedan	Asia	$21,761
3	Acura	TSX 4dr	Sedan	Asia	$24,647
4	Acura	TL 4dr	Sedan	Asia	$30,299
5	Acura	3.5 RL 4dr	Sedan	Asia	$39,014
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	$41,100
7	Acura	NSX coupe 2dr manual S	Sports	Asia	$79,978
8	Audi	A4 1.8T 4dr	Sedan	Europe	$23,508
9	Audi	A41.8T convertible 2dr	Sedan	Europe	$32,506
10	Audi	A4 3.0 4dr	Sedan	Europe	$28,846
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	$30,366
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	$31,388
13	Audi	A6 3.0 4dr	Sedan	Europe	$33,129

Output 19.38 *PowerPointAqua Style*



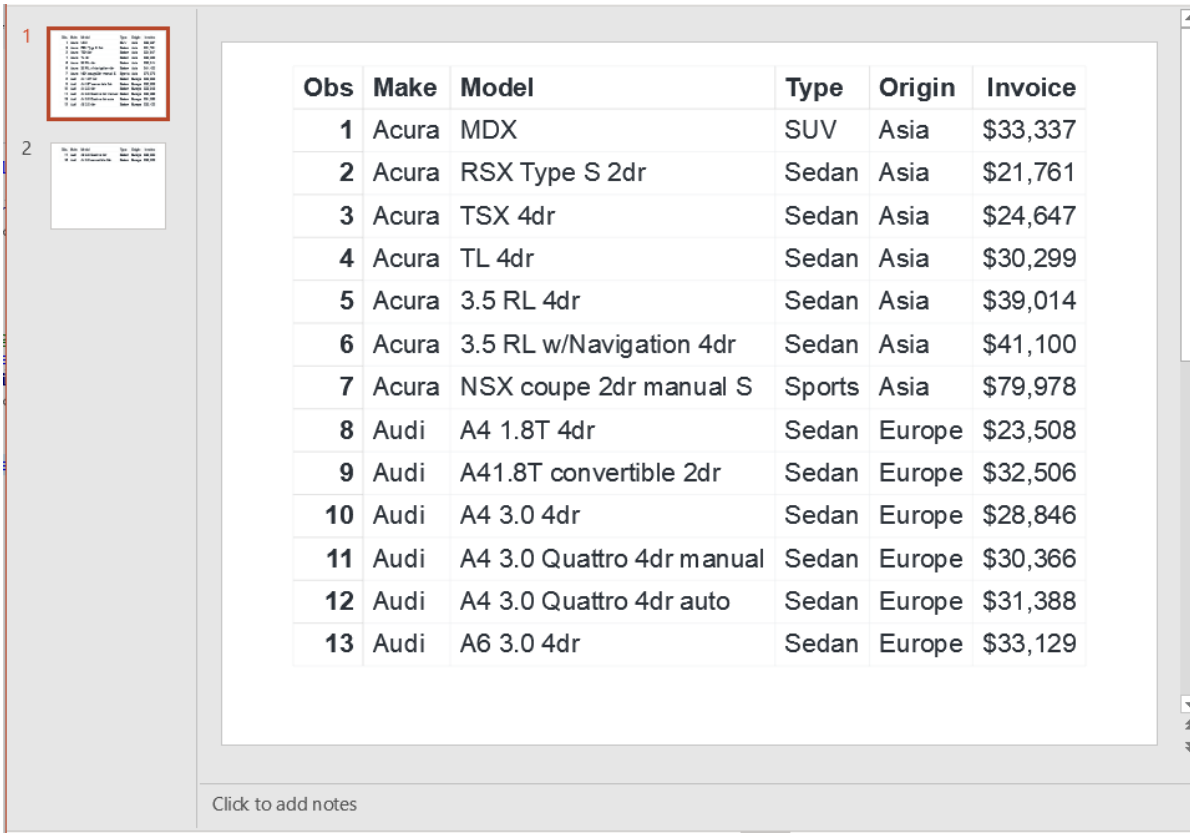
Obs	Make	Model	Type	Origin	Invoice
1	Acura	MDX	SUV	Asia	$33,337
2	Acura	RSX Type S 2dr	Sedan	Asia	$21,761
3	Acura	TSX 4dr	Sedan	Asia	$24,647
4	Acura	TL 4dr	Sedan	Asia	$30,299
5	Acura	3.5 RL 4dr	Sedan	Asia	$39,014
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	$41,100
7	Acura	NSX coupe 2dr manual S	Sports	Asia	$79,978
8	Audi	A4 1.8T 4dr	Sedan	Europe	$23,508
9	Audi	A41.8T convertible 2dr	Sedan	Europe	$32,506
10	Audi	A4 3.0 4dr	Sedan	Europe	$28,846
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	$30,366
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	$31,388
13	Audi	A6 3.0 4dr	Sedan	Europe	$33,129

The following style is recommended for generating accessible table output when used with the ODS PowerPoint destination.

Output 19.39 *PowerPointignite Style*

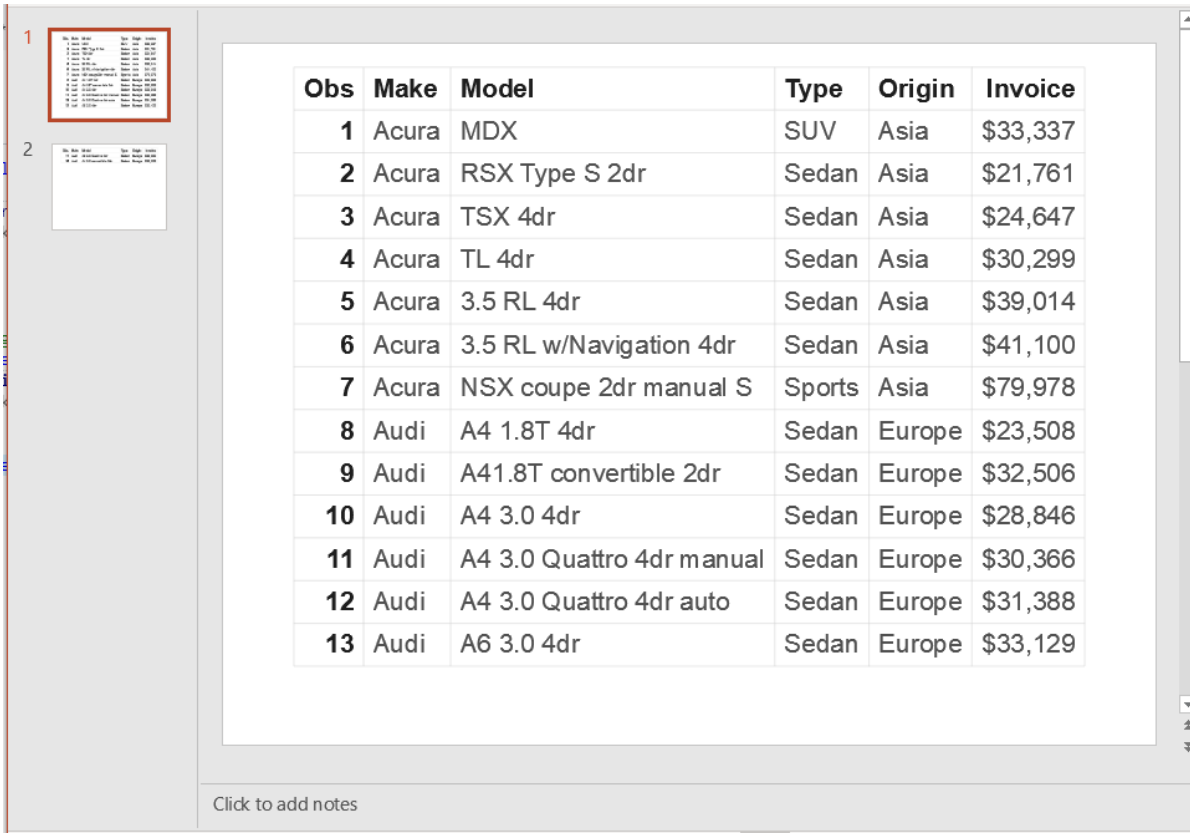


Output 19.40 *PowerPointIlluminate Style*



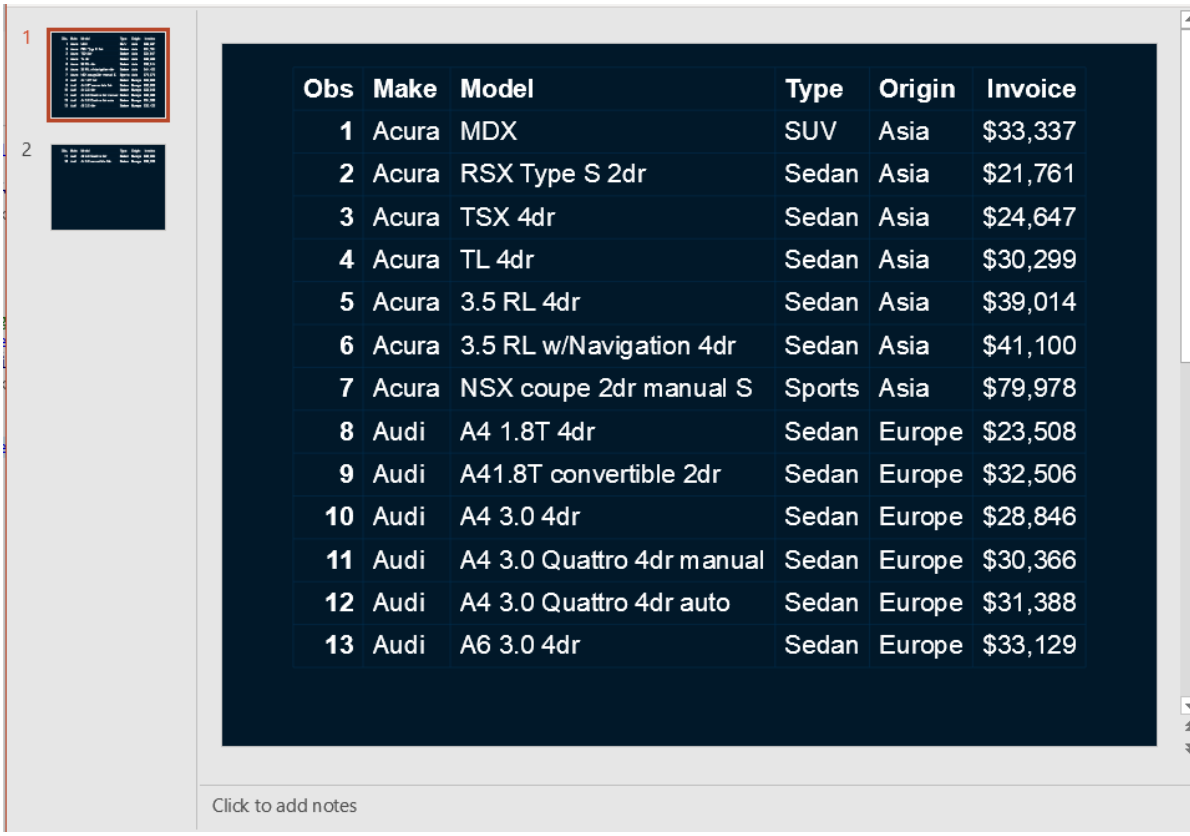
Obs	Make	Model	Type	Origin	Invoice
1	Acura	MDX	SUV	Asia	$33,337
2	Acura	RSX Type S 2dr	Sedan	Asia	$21,761
3	Acura	TSX 4dr	Sedan	Asia	$24,647
4	Acura	TL 4dr	Sedan	Asia	$30,299
5	Acura	3.5 RL 4dr	Sedan	Asia	$39,014
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	$41,100
7	Acura	NSX coupe 2dr manual S	Sports	Asia	$79,978
8	Audi	A4 1.8T 4dr	Sedan	Europe	$23,508
9	Audi	A41.8T convertible 2dr	Sedan	Europe	$32,506
10	Audi	A4 3.0 4dr	Sedan	Europe	$28,846
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	$30,366
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	$31,388
13	Audi	A6 3.0 4dr	Sedan	Europe	$33,129

Output 19.41 *PowerPointiMarine Style*

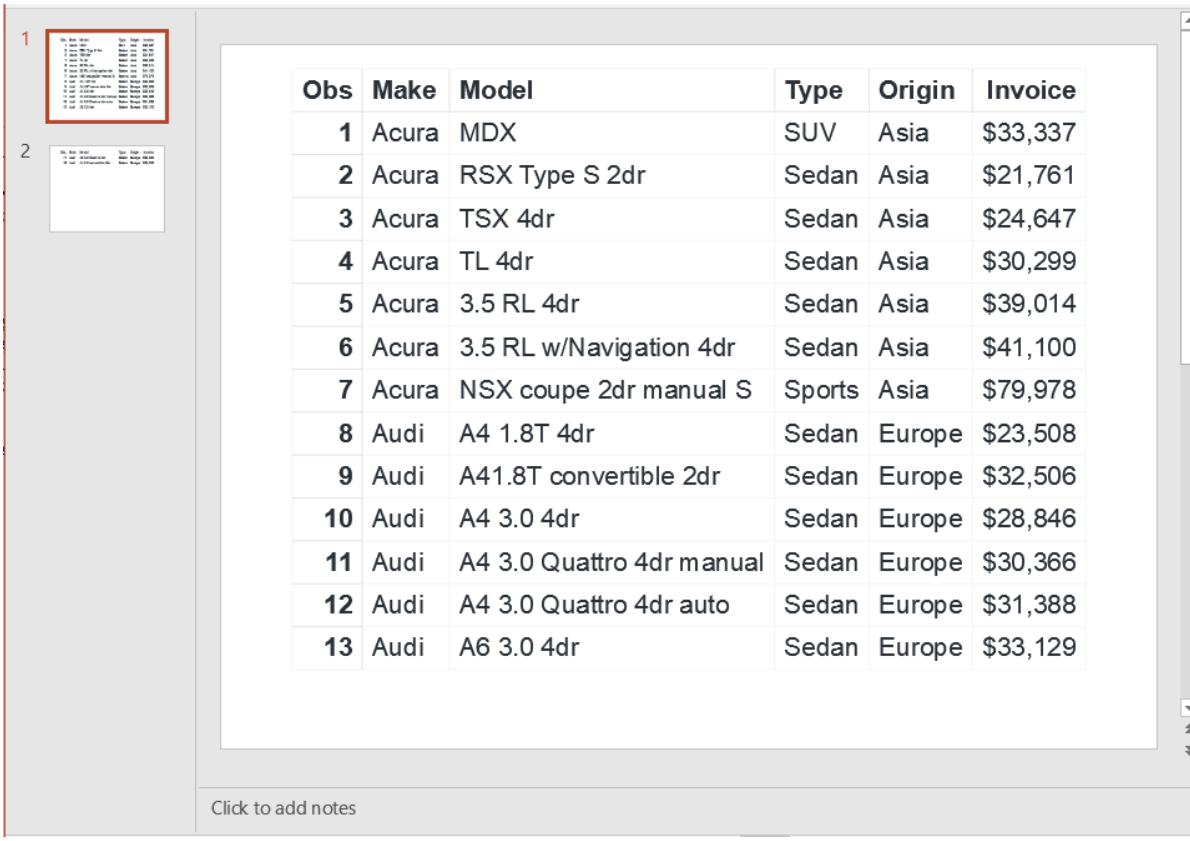


Obs	Make	Model	Type	Origin	Invoice
1	Acura	MDX	SUV	Asia	$33,337
2	Acura	RSX Type S 2dr	Sedan	Asia	$21,761
3	Acura	TSX 4dr	Sedan	Asia	$24,647
4	Acura	TL 4dr	Sedan	Asia	$30,299
5	Acura	3.5 RL 4dr	Sedan	Asia	$39,014
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	$41,100
7	Acura	NSX coupe 2dr manual S	Sports	Asia	$79,978
8	Audi	A4 1.8T 4dr	Sedan	Europe	$23,508
9	Audi	A41.8T convertible 2dr	Sedan	Europe	$32,506
10	Audi	A4 3.0 4dr	Sedan	Europe	$28,846
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	$30,366
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	$31,388
13	Audi	A6 3.0 4dr	Sedan	Europe	$33,129

Output 19.42 *PowerPointiMidnight Style*

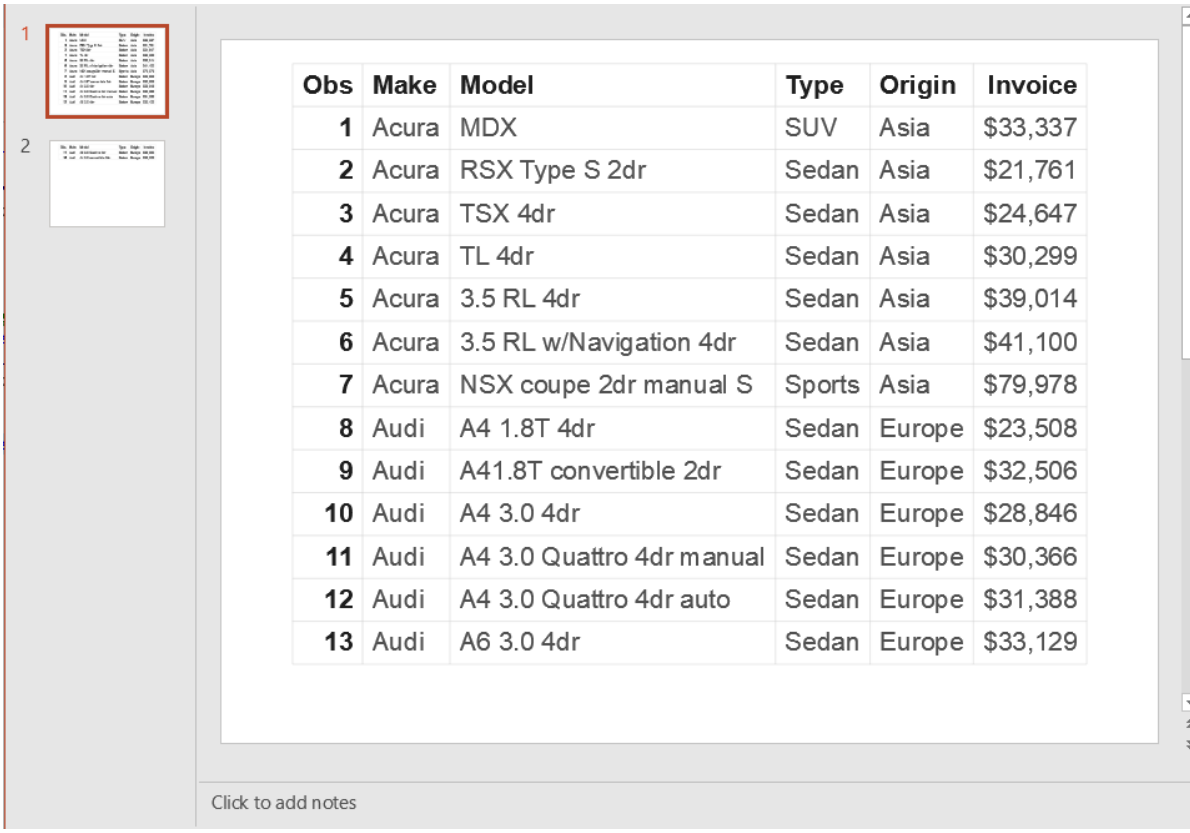


Output 19.43 *PowerPointiOpal Style*



Obs	Make	Model	Type	Origin	Invoice
1	Acura	MDX	SUV	Asia	$33,337
2	Acura	RSX Type S 2dr	Sedan	Asia	$21,761
3	Acura	TSX 4dr	Sedan	Asia	$24,647
4	Acura	TL 4dr	Sedan	Asia	$30,299
5	Acura	3.5 RL 4dr	Sedan	Asia	$39,014
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	$41,100
7	Acura	NSX coupe 2dr manual S	Sports	Asia	$79,978
8	Audi	A4 1.8T 4dr	Sedan	Europe	$23,508
9	Audi	A41.8T convertible 2dr	Sedan	Europe	$32,506
10	Audi	A4 3.0 4dr	Sedan	Europe	$28,846
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	$30,366
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	$31,388
13	Audi	A6 3.0 4dr	Sedan	Europe	$33,129

Output 19.44 *PowerPointiUmstead Style*



Styles for the ODS Destination for Word

Output 19.45 *Word Style*

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
Buick	Rainier	SUV	USA	All	$37,895	$34,357	4.2
Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085	3.4
Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377	5.3
Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523	4.6
Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422	5.3
Chevrolet	Tahoe LT	SUV	USA	All	$41,465	$36,287	5.3
Chevrolet	TrailBlazer LT	SUV	USA	Front	$30,295	$27,479	4.2
Chevrolet	Tracker	SUV	USA	Front	$20,255	$19,108	2.5
Dodge	Durango SLT	SUV	USA	All	$32,235	$29,472	4.7
Ford	Excursion 6.8 XLT	SUV	USA	All	$41,475	$36,494	6.8
Ford	Expedition 4.6 XLT	SUV	USA	Front	$34,560	$30,468	4.6
Ford	Explorer XLT V6	SUV	USA	All	$29,670	$26,983	4.0
Ford	Escape XLS	SUV	USA	All	$22,515	$20,907	3.0
GMC	Envoy XUV SLE	SUV	USA	Front	$31,890	$28,922	4.2
GMC	Yukon 1500 SLE	SUV	USA	Front	$35,725	$31,361	4.8
GMC	Yukon XL 2500 SLT	SUV	USA	All	$46,265	$40,534	6.0
Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5
Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4
Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4
Hummer	H2	SUV	USA	All	$49,995	$45,815	6.0
Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589	$20,201	2.7
Isuzu	Ascender S	SUV	Asia	All	$31,849	$29,977	4.2
Isuzu	Rodeo S	SUV	Asia	Front	$20,449	$19,261	3.2

Output 19.46 *WordAqua Style*

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
Buick	Rainier	SUV	USA	All	$37,895	$34,357	4.2
Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085	3.4
Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377	5.3
Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523	4.6
Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422	5.3
Chevrolet	Tahoe LT	SUV	USA	All	$41,465	$36,287	5.3
Chevrolet	Tracker	SUV	USA	Front	$20,255	$19,108	2.5
Chevrolet	TrailBlazer LT	SUV	USA	Front	$30,295	$27,479	4.2
Dodge	Durango SLT	SUV	USA	All	$32,235	$29,472	4.7
Ford	Escape XLS	SUV	USA	All	$22,515	$20,907	3.0
Ford	Excursion 6.8 XLT	SUV	USA	All	$41,475	$36,494	6.8
Ford	Expedition 4.6 XLT	SUV	USA	Front	$34,560	$30,468	4.6
Ford	Explorer XLT V6	SUV	USA	All	$29,670	$26,983	4.0
GMC	Envoy XUV SLE	SUV	USA	Front	$31,890	$28,922	4.2
GMC	Yukon 1500 SLE	SUV	USA	Front	$35,725	$31,361	4.8
GMC	Yukon XL 2500 SLT	SUV	USA	All	$46,265	$40,534	6.0
Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4
Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4
Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5
Hummer	H2	SUV	USA	All	$49,995	$45,815	6.0
Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589	$20,201	2.7
Isuzu	Ascender S	SUV	Asia	All	$31,849	$29,977	4.2
Isuzu	Rodeo S	SUV	Asia	Front	$20,449	$19,261	3.2
Jeep	Grand Cherokee Laredo	SUV	USA	Front	$27,905	$25,686	4.0

Output 19.47 *WordHighContrast Style*

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
Buick	Rainier	SUV	USA	All	$37,895	$34,357	4.2
Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085	3.4
Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377	5.3
Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523	4.6
Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422	5.3
Chevrolet	Tahoe LT	SUV	USA	All	$41,465	$36,287	5.3
Chevrolet	Tracker	SUV	USA	Front	$20,255	$19,108	2.5
Chevrolet	TrailBlazer LT	SUV	USA	Front	$30,295	$27,479	4.2
Dodge	Durango SLT	SUV	USA	All	$32,235	$29,472	4.7
Ford	Escape XLS	SUV	USA	All	$22,515	$20,907	3.0
Ford	Excursion 6.8 XLT	SUV	USA	All	$41,475	$36,494	6.8
Ford	Expedition 4.6 XLT	SUV	USA	Front	$34,560	$30,468	4.6
Ford	Explorer XLT V6	SUV	USA	All	$29,670	$26,983	4.0
GMC	Envoy XUV SLE	SUV	USA	Front	$31,890	$28,922	4.2
GMC	Yukon 1500 SLE	SUV	USA	Front	$35,725	$31,361	4.8
GMC	Yukon XL 2500 SLT	SUV	USA	All	$46,265	$40,534	6.0
Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4
Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4
Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5
Hummer	H2	SUV	USA	All	$49,995	$45,815	6.0
Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589	$20,201	2.7
Isuzu	Ascender S	SUV	Asia	All	$31,849	$29,977	4.2
Isuzu	Rodeo S	SUV	Asia	Front	$20,449	$19,261	3.2

Output 19.48 *WordIgnite Style*

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
Buick	Rainier	SUV	USA	All	$37,895	$34,357	4.2
Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085	3.4
Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377	5.3
Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523	4.6
Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422	5.3
Chevrolet	Tahoe LT	SUV	USA	All	$41,465	$36,287	5.3
Chevrolet	Tracker	SUV	USA	Front	$20,255	$19,108	2.5
Chevrolet	TrailBlazer LT	SUV	USA	Front	$30,295	$27,479	4.2
Dodge	Durango SLT	SUV	USA	All	$32,235	$29,472	4.7
Ford	Escape XLS	SUV	USA	All	$22,515	$20,907	3.0
Ford	Excursion 6.8 XLT	SUV	USA	All	$41,475	$36,494	6.8
Ford	Expedition 4.6 XLT	SUV	USA	Front	$34,560	$30,468	4.6
Ford	Explorer XLT V6	SUV	USA	All	$29,670	$26,983	4.0
GMC	Envoy XUV SLE	SUV	USA	Front	$31,890	$28,922	4.2
GMC	Yukon 1500 SLE	SUV	USA	Front	$35,725	$31,361	4.8
GMC	Yukon XL 2500 SLT	SUV	USA	All	$46,265	$40,534	6.0
Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4
Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4
Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5
Hummer	H2	SUV	USA	All	$49,995	$45,815	6.0
Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589	$20,201	2.7
Isuzu	Ascender S	SUV	Asia	All	$31,849	$29,977	4.2
Isuzu	Rodeo S	SUV	Asia	Front	$20,449	$19,261	3.2
Jeep	Grand Cherokee Laredo	SUV	USA	Front	$27,995	$25,686	4.0

Output 19.49 *WordIlluminate Style*

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
Buick	Rainier	SUV	USA	All	$37,895	$34,357	4.2
Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085	3.4
Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377	5.3
Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523	4.6
Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422	5.3
Chevrolet	Tahoe LT	SUV	USA	All	$41,465	$36,287	5.3
Chevrolet	Tracker	SUV	USA	Front	$20,255	$19,108	2.5
Chevrolet	TrailBlazer LT	SUV	USA	Front	$30,295	$27,479	4.2
Dodge	Durango SLT	SUV	USA	All	$32,235	$29,472	4.7
Ford	Escape XLS	SUV	USA	All	$22,515	$20,907	3.0
Ford	Excursion 6.8 XLT	SUV	USA	All	$41,475	$36,494	6.8
Ford	Expedition 4.6 XLT	SUV	USA	Front	$34,560	$30,468	4.6
Ford	Explorer XLT V6	SUV	USA	All	$29,670	$26,983	4.0
GMC	Envoy XUV SLE	SUV	USA	Front	$31,890	$28,922	4.2
GMC	Yukon 1500 SLE	SUV	USA	Front	$35,725	$31,361	4.8
GMC	Yukon XL 2500 SLT	SUV	USA	All	$46,265	$40,534	6.0
Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4
Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4
Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5
Hummer	H2	SUV	USA	All	$49,995	$45,815	6.0
Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589	$20,201	2.7
Isuzu	Ascender S	SUV	Asia	All	$31,849	$29,977	4.2
Isuzu	Rodeo S	SUV	Asia	Front	$20,449	$19,261	3.2

Output 19.50 *WordMarine Style*

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
Buick	Rainier	SUV	USA	All	$37,895	$34,357	4.2
Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085	3.4
Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377	5.3
Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523	4.6
Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422	5.3
Chevrolet	Tahoe LT	SUV	USA	All	$41,465	$36,287	5.3
Chevrolet	Tracker	SUV	USA	Front	$20,255	$19,108	2.5
Chevrolet	TrailBlazer LT	SUV	USA	Front	$30,295	$27,479	4.2
Dodge	Durango SLT	SUV	USA	All	$32,235	$29,472	4.7
Ford	Escape XLS	SUV	USA	All	$22,515	$20,907	3.0
Ford	Excursion 6.8 XLT	SUV	USA	All	$41,475	$36,494	6.8
Ford	Expedition 4.6 XLT	SUV	USA	Front	$34,560	$30,468	4.6
Ford	Explorer XLT V6	SUV	USA	All	$29,670	$26,983	4.0
GMC	Envoy XUV SLE	SUV	USA	Front	$31,890	$28,922	4.2
GMC	Yukon 1500 SLE	SUV	USA	Front	$35,725	$31,361	4.8
GMC	Yukon XL 2500 SLT	SUV	USA	All	$46,265	$40,534	6.0
Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4
Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4
Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5
Hummer	H2	SUV	USA	All	$49,995	$45,815	6.0
Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589	$20,201	2.7
Isuzu	Ascender S	SUV	Asia	All	$31,849	$29,977	4.2
Isuzu	Rodeo S	SUV	Asia	Front	$20,449	$19,261	3.2

Output 19.51 *WordMidnight Style*

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
Buick	Rainier	SUV	USA	All	$37,895	$34,357	4.2
Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085	3.4
Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377	5.3
Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523	4.6
Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422	5.3
Chevrolet	Tahoe LT	SUV	USA	All	$41,465	$36,287	5.3
Chevrolet	Tracker	SUV	USA	Front	$20,255	$19,108	2.5
Chevrolet	TrailBlazer LT	SUV	USA	Front	$30,295	$27,479	4.2
Dodge	Durango SLT	SUV	USA	All	$32,235	$29,472	4.7
Ford	Escape XLS	SUV	USA	All	$22,515	$20,907	3.0
Ford	Excursion 6.8 XLT	SUV	USA	All	$41,475	$36,494	6.8
Ford	Expedition 4.6 XLT	SUV	USA	Front	$34,560	$30,468	4.6
Ford	Explorer XLT V6	SUV	USA	All	$29,670	$26,983	4.0
GMC	Envoy XUV SLE	SUV	USA	Front	$31,890	$28,922	4.2
GMC	Yukon 1500 SLE	SUV	USA	Front	$35,725	$31,361	4.8
GMC	Yukon XL 2500 SLT	SUV	USA	All	$46,265	$40,534	6.0
Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4
Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4
Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5
Hummer	H2	SUV	USA	All	$49,995	$45,815	6.0
Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589	$20,201	2.7
Isuzu	Ascender S	SUV	Asia	All	$31,849	$29,977	4.2
Isuzu	Rodeo S	SUV	Asia	Front	$20,449	$19,261	3.2

Output 19.52 *WordOpal Style*

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
Buick	Rainier	SUV	USA	All	$37,895	$34,357	4.2
Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085	3.4
Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377	5.3
Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523	4.6
Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422	5.3
Chevrolet	Tahoe LT	SUV	USA	All	$41,465	$36,287	5.3
Chevrolet	Tracker	SUV	USA	Front	$20,255	$19,108	2.5
Chevrolet	TrailBlazer LT	SUV	USA	Front	$30,295	$27,479	4.2
Dodge	Durango SLT	SUV	USA	All	$32,235	$29,472	4.7
Ford	Escape XLS	SUV	USA	All	$22,515	$20,907	3.0
Ford	Excursion 6.8 XLT	SUV	USA	All	$41,475	$36,494	6.8
Ford	Expedition 4.6 XLT	SUV	USA	Front	$34,560	$30,468	4.6
Ford	Explorer XLT V6	SUV	USA	All	$29,670	$26,983	4.0
GMC	Envoy XUV SLE	SUV	USA	Front	$31,890	$28,922	4.2
GMC	Yukon 1500 SLE	SUV	USA	Front	$35,725	$31,361	4.8
GMC	Yukon XL 2500 SLT	SUV	USA	All	$46,265	$40,534	6.0
Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4
Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4
Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5
Hummer	H2	SUV	USA	All	$49,995	$45,815	6.0
Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589	$20,201	2.7
Isuzu	Ascender S	SUV	Asia	All	$31,849	$29,977	4.2
Isuzu	Rodeo S	SUV	Asia	Front	$20,449	$19,261	3.2

Output 19.53 *WordUmstead Style*

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
Buick	Rainier	SUV	USA	All	$37,895	$34,357	4.2
Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085	3.4
Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377	5.3
Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523	4.6
Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422	5.3
Chevrolet	Tahoe LT	SUV	USA	All	$41,465	$36,287	5.3
Chevrolet	Tracker	SUV	USA	Front	$20,255	$19,108	2.5
Chevrolet	TrailBlazer LT	SUV	USA	Front	$30,295	$27,479	4.2
Dodge	Durango SLT	SUV	USA	All	$32,235	$29,472	4.7
Ford	Escape XLS	SUV	USA	All	$22,515	$20,907	3.0
Ford	Excursion 6.8 XLT	SUV	USA	All	$41,475	$36,494	6.8
Ford	Expedition 4.6 XLT	SUV	USA	Front	$34,560	$30,468	4.6
Ford	Explorer XLT V6	SUV	USA	All	$29,670	$26,983	4.0
GMC	Envoy XUV SLE	SUV	USA	Front	$31,890	$28,922	4.2
GMC	Yukon 1500 SLE	SUV	USA	Front	$35,725	$31,361	4.8
GMC	Yukon XL 2500 SLT	SUV	USA	All	$46,265	$40,534	6.0
Honda	CR-V LX	SUV	Asia	All	$19,860	$18,419	2.4
Honda	Element LX	SUV	Asia	All	$18,690	$17,334	2.4
Honda	Pilot LX	SUV	Asia	All	$27,560	$24,843	3.5
Hummer	H2	SUV	USA	All	$49,995	$45,815	6.0
Hyundai	Santa Fe GLS	SUV	Asia	Front	$21,589	$20,201	2.7
Isuzu	Ascender S	SUV	Asia	All	$31,849	$29,977	4.2
Isuzu	Rodeo S	SUV	Asia	Front	$20,449	$19,261	3.2

20

Style Elements

ODS Style Elements 777
Overview 777
Style Elements Affecting Documents 778
Style Elements Affecting Tables of Contents 778
Style Elements Affecting Pages 779
Style Elements Affecting Frames 780
Style Elements Affecting the Body File 780
Style Elements Affecting Tables 781
Style Elements Affecting Text 783
Style Elements Affecting Layout 787
Style Elements Affecting Graphs 787
Miscellaneous Elements 787
Style Elements Affecting Template-Based Graphics 788
Style Elements Affecting Device-Based Graphics 797

ODS Style Elements

Overview

The following tables lists all the style elements available for ODS style definitions. The tables provides a brief description of each style element and indicates the style elements from which it inherits its attributes.

Abstract style elements are not explicitly used in the ODS output. They are used for inheritance purposes only. Because of this, abstract styles do not appear in the output of destinations that generate a style sheet.

Style Elements Affecting Documents

Table 20.1 Style Elements Affecting Documents

Style Element	Description	Inherits from
Document	Controls the various document bodies. This generally includes things like the page background color and page margins. **Note:** This is an abstract style element.	Container
Body	Controls the Body file. The Body file is created by the BODY= option in any ODS Markup Family Destination.	Document
Frame	Controls the Frame file for HTML. The Frame file is created by the FRAME= option in any ODS Markup Family Destination.	Document
Contents	Controls the Contents file. The Contents file is created by the CONTENTS= option in any ODS Markup Family Destination.	Document
Pages	Controls the Page file. The Pages file is created by the PAGE= option in any ODS Markup Family Destination.	Document

Style Elements Affecting Tables of Contents

Table 20.2 Style Elements Affecting Tables of Contents and Tables of Pages

Style Element	Description	Inherits from
Contents	Controls the Contents file	Document

Style Element	Description	Inherits from
IndexItem	Controls list items and folders for Contents and Pages **Note:** This is an abstract style element.	Container
ContentFolder	Controls the folders in the Contents file	IndexItem
ByContentFolder	Controls the bygroup folders in the Contents file	ContentFolder
ContentItem	Controls the items in the Contents file	IndexItem
Index	Controls miscellaneous Contents and Pages components **Note:** This is an abstract style element.	Container
IndexProcName	Controls the PROC name in the Contents and Pages files **Note:** This is an abstract style element.	Index
ContentProcName	Controls the PROC name in the Contents file	IndexProcName
ContentProcLabel	Controls the PROC label in the Contents file	ContentProcName
IndexAction	Determines what happens on mouse-over events for folders and items (HTML only)	IndexItem
FolderAction	Determines what happens on mouse-over events for folders (HTML only)	IndexAction
IndexTitle	Controls the title of Contents and Pages files **Note:** This is an abstract style element.	Index
ContentTitle	Controls the title of the Contents file.	IndexTitle

Style Elements Affecting Pages

The PAGE= option in applicable MARKUP family destinations creates the page file. See PAGE= option.

Table 20.3 Style Elements Affecting Pages

Style Element	Description	Inherits from
Pages	Controls the Page file. The Pages file is created by the PAGE= option.	Document
Date	Controls how date fields look **Note:** This is an abstract style element.	Container
PagesDate	Controls the date in the Pages file	Date
PagesTitle	Controls the title of the Pages file	IndexTitle
PagesItem	Controls the leafnode item in the Pages file	IndexItem
PagesProcName	Controls the proc name in the Pages file	IndexProcName
PagesProcLabel	Controls the proc label in the Pages file	PagesProcName

Style Elements Affecting Frames

The FRAME= option in ODS MARKUP destinations creates the frame file. See FRAME= option.

Table 20.4 Style Elements Affecting Frame

Style Element	Description	Inheritance
Frame	Controls the Frame file for HTML	Document

Style Elements Affecting the Body File

The BODY style element inherits from the DOCUMENT style element. The BODY= option in applicable Markup Family destinations creates the body file. See BODY= option.

The following style elements affect the body of an HTML document:

- Tables on page 781
- Text on page 783
- Layouts on page 787

Style Elements Affecting Tables

Table 20.5 Style Elements Affecting Data Cells in Tables

Style Element	Description	Inherits from
Data	Default style for data cells	Cell
Cell	Controls data, header, and footer cells **Note:** This is an abstract style element.	Container
DataFixed	Default style for data cells that request a fixed font	Data
DataEmpty	Controls empty data cells	Data
DataEmphasis	Controls emphasized data cells	Data
DataEmphasisFixed	Controls emphasized data cells that request a fixed font	DataEmphasis
DataStrong	Controls strong (more emphasized) data cells	Data
DataStrongFixed	Controls strong (more emphasized) data cells that request a fixed font	DataStrong

Table 20.6 Style Elements Affecting Header Cells

Style Element	Description	Inherits from
Header	Controls the headers of a table	HeadersAndFooters
HeaderFixed	Controls the header of a table that request a fixed font	Header
HeaderEmphasis	Controls emphasized table header cells that request a fixed font	Header
HeaderEmphasisFixed	Controls emphasized table header cells that request a fixed font	HeaderEmphasis
HeaderStrong	Controls strong (more emphasized) table header cells	Header

Style Element	Description	Inherits from
HeaderStrongFixed	Controls strong (more emphasized) table header cells	HeaderStrong
HeaderEmpty	Controls empty table header cells	Header
RowHeader	Controls row headers	Header
RowHeaderFixed	Controls row headers that request a fixed font	RowHeader
RowHeaderEmpty	Controls empty row headers	RowHeader
RowHeaderEmphasis	Controls emphasized row headers	RowHeader
RowHeaderEmphasisFixed	Controls emphasized row headers that request a fixed font	RowHeaderEmphasis
RowHeaderStrong	Controls strong (more emphasized) row headers	RowHeader
RowHeaderStrongFixed	Controls strong (more emphasized) row headers that request a fixed font	RowHeaderStrong

Table 20.7 *Style Elements Affecting Footer Cells*

Style Element	Description	Inherits from
Footer	Controls table footers	HeadersAndFooters
FooterFixed	Controls table footers that request a fixed font	Footer
FooterEmpty	Controls empty table footers	Footer
FooterEmphasis	Controls emphasized table footers	Footer
FooterEmphasisFixed	Controls emphasized table footers that request a fixed font	FooterEmphasis
FooterStrong	Controls strong (more emphasized) table footers	Footer

Style Element	Description	Inherits from
FooterStrongFixed	Controls strong (more emphasized) table footers that request a fixed font	FooterStrong
RowFooter	Controls a row footer (label)	Footer
RowFooterFixed	Controls a row footer (label) that request a fixed font	RowFooter
RowFooterEmpty	Controls an empty row footer (label)	RowFooter
RowFooterEmphasis	Controls an emphasized row footer (label)	RowFooter
RowFooterEmphasisFixed	Controls an emphasized row footer (label) that request a fixed font	RowFooterEmphasis
RowFooterStrong	Controls a strong (more emphasized) row footer (label)	RowFooter
RowFooterStrongFixed	Controls a strong (more emphasized) row footer (label) that requests a fixed font	RowFooterStrong

Style Elements Affecting Text

Table 20.8 *Elements affecting Text*

Style Element	Description	Inherits from
BodyDate	Controls the date field in the Body file	ContentsDate
SASDate	Controls the date field in the Body file when SASDATE option is in effect. This Element is valid only in the ODS destination for PowerPoint.	BodyDate
PageNo	Controls page numbers for paginated destinations	TitlesAndFooters

Style Element	Description	Inherits from
	(RTF, TAGSETS.RTF, PDF, PowerPoint)	
SysTitleAndFooterContainer	Controls the container for system page title and system page footer. This element is usually used to add borders around a title.	Container
TitlesAndFooters	Controls system page title text and system page footer text **Note:** This is an abstract style element.	Container
SystemTitle	Controls system title text	TitlesAndFooters
SystemTitle2-10	Controls SystemTitle text for the corresponding SystemTitle*n* value. For example, SystemTitle3 controls SystemTitle3 text.	SystemTitle-SystemTitle9
SystemFooter	Controls system footer text	TitlesAndFooters
SystemFooter2-10	Controls SystemFooter text for the corresponding SystemFooter*n* value. For example, SystemFooter3 controls SystemFooter3 text.	SystemFooter-SystemFooter9
BylineContainer	Controls the container for the byline. This is generally used to add borders to a byline.	Container
Byline	Controls byline text	TitlesAndFooters
Parskip	Controls space between tables in RTF output	TitlesAndFooters
ProcTitle	Controls procedure title text	TitlesAndFooters
ProcTitleFixed	Controls procedure title text that requests a fixed font	ProcTitle

Style Element	Description	Inherits from
UserText	Controls the ODS TEXT= style	Note
PrePage	Controls the ODS RTF/ MEASURED PREPAGE= style	Note
Note	Controls the container for note banners and note contents **Note:** This is an abstract style element.	Container
NoteBanner	Controls the banner for Note	Note
NoteContent	Controls the contents for Note	Note
NoteContentFixed	Controls the contents for Note. Fixed font.	NoteContent
WarnBanner	Controls the banner for Warnings	Note
WarnContent	Controls the contents of Warnings	Note
WarnContentFixed	Controls the contents for Warnings. Fixed font.	WarnContent
ErrorBanner	Controls the banner for Errors	Note
ErrorContent	Controls the contents of Errors	Note
ErrorContentFixed	Controls the contents for Errors. Fixed font.	ErrorContent
FatalBanner	Controls the banner for Fatal	Note
FatalContent	Controls the contents of Fatal	Note
FatalContentFixed	Controls the contents for Fatal. Fixed font.	FatalContent

Style Element	Description	Inherits from
Paragraph	Controls paragraphs in text	Container
Heading1-Heading10	Controls Heading text for the corresponding Heading*n* value. For example, Heading2 controls the Heading2 text.	NormalText
List	Controls both unordered and ordered lists	Container
List2-List10	Controls nested bulleted lists at the corresponding List*n* level. For example, List3 controls the third level bulleted list.	List-List9
ListItem	Controls bulleted list items	Container
ListItem2-ListItem10	Controls the nested bulleted list items at the corresponding ListItem*n* level. For example, ListItem3 controls the third level bulleted list item.	ListItem-ListItem9
Pagebreak	Controls page breaks in the text	No inheritance
LineContent	Controls the text generated by the LINE statement in PROC REPORT	No inheritance
Link	Controls the color of un-visited link text	No inheritance
ActiveLink	Controls the color of active (clicked) link text	No inheritance
FocusLink	Controls the color of the link-focus indicator **Accessibility note:** For an example that uses the FocusLink style element, see Enhancing the Appearance of the Link-Focus Indicator.	No inheritance

Style Element	Description	Inherits from
VisitedLink	Controls the color of visited link text	No inheritance

Style Elements Affecting Layout

Table 20.9 *Elements Affecting Layout*

Style Element	Description	Inheritance
LayoutContainer	Container for Layout	No inheritance
LayoutRegion	Region style for Layout cells	LayoutContainer

Note: Layouts and regions are created by the ODS LAYOUT and ODS REGION statements. See ODS LAYOUT ABSOLUTE Statement, and ODS REGION Statement, Absolute

Style Elements Affecting Graphs

For more information about Style Elements that affect graphs, see "Style Elements for Use with ODS Graphics" in *SAS ODS Graphics: Procedures Guide*.

Miscellaneous Elements

Table 20.10 *Miscellaneous Style Elements*

Style Element	Description	Inherits from
Container	Controls all container-oriented elements	
Continued	Controls continued flag when a table breaks across a page (paginated destinations only)	TitlesAndFooters

Style Element	Description	Inherits from
ExtendedPage	Displays a message when the page does not fit (PDF only)	TitlesAndFooters
StartUpFunction	This is a Javascript function that is added to the HTML output. Any Javascript code in the TAGATTR= attribute is executed when the page is loaded.	No inheritance
ShutDownFunction	Controls the Shut-Down function. This is a Javascript function that is added to the HTML output. Any Javascript code in the TAGATTR= attribute is executed when the page is exited.	No inheritance
UserText	Controls the ODS TEXT= style	Note

Style Elements Affecting Template-Based Graphics

The following style elements affect template-based graphics and can be specified by Graph Template Language appearance options or used in styles. Template-based graphics include all ODS Graphics output where a compiled ODS template of type STATGRAPH is used to produce graphical output. Supplied templates are stored in Sashelp.Tmplmst. SAS/GRAPH device drivers and some global statements such as SYMBOL, PATTERN, AXIS, and LEGEND have no affect. Common ODS Graphics procedures that produce template-based graphics are SGPLOT, SGPANEL, SGPIE, SGMAP, SGSCATTER, and SGRENDER in addition to many SAS/STAT, SAS/ETS, and SAS/QC procedures. ODS graphics always produce output as image files and use the ODS GRAPHICS statement to control the graphical environment.

Certain style elements were created to be used with specific plots or graphs. For example, the style element GraphFit2 is best used to modify secondary fit lines. The style element GraphConfidence2 was created to modify secondary confidence bands. The table below lists each style element, the portion of the graph that it affects or was created to use with, and the default attribute values. Attribute values can be changed with PROC TEMPLATE, as stated above.

For complete documentation on the style attributes that can be specified in each style element, see “About Style Attributes” on page 465.

Table 20.11 Graph Style Elements: General Graph Appearance

Style Element	Portion of Graph Affected	Recognized Attributes
GraphAnnoLine	Annotation lines and arrows, and outlines for closed annotation shapes such as circles and squares.	ContrastColor LineStyle LineThickness
GraphAnnoShape	Fill for closed annotation shapes such as circles and squares.	Color MarkerSize MarkerSymbol Transparency
GraphAnnoText	Annotation text	Color Font or *font-attributes*[1]
GraphAxisLines	X-, Y-, and Z-axis lines	ContrastColor LineStyle LineThickness TickDisplay
GraphBackground	Background of the graph	Color Transparency
GraphBaseMap[2]	Appearance of the tiles in the base map that is generated by the SGMAP procedure OPENSTREETMAP statement.	TileSet[3]
GraphBorderLines	Border line around the graph and around the graph legend	ContrastColor LineStyle LineThickness
GraphDataText	Text font and color for point and line labels	Color Font or *font-attributes*[1]
GraphFootnoteText	Text font and color for footnote(s)	Color Font or *font-attributes*[1]
GraphGridLines	Horizontal and vertical grid lines drawn at major tick marks	ContrastColor DisplayOpts LineStyle LineThickness

Style Element	Portion of Graph Affected	Recognized Attributes
		Transparency
GraphHeaderBackground	Background color of the cell headers in a data-driven lattice or data-driven panel graph	Color Transparency
GraphLabelText	Text font and color for axis labels and legend titles	Color Font or *font-attributes*[1]
GraphLegendBackground	Background color of the legend	Color Transparency
GraphMinorGridLines	Appearance of the horizontal and vertical minor grid lines.	ContrastColor DisplayOpts LineStyle LineThickness
GraphOutlines	Outline properties for fill areas such as bars, pie slices, box plots, ellipses, and histograms	Color ContrastColor LineStyle LineThickness
GraphReference	Horizontal and vertical reference lines and drop lines	ContrastColor LineStyle LineThickness
GraphTitleText	Text font and color for title(s)	Color Font or *font-attributes*[1]
GraphUnicodeText	Text font for Unicode values	Color Font or *font-attributes*[1]
GraphValueText	Text font and color for axis tick values and legend values	Color Font or *font-attributes*[1]
GraphWalls	Vertical wall(s) bounded by axes	Color ContrastColor FrameBorder LineStyle

Style Element	Portion of Graph Affected	Recognized Attributes
		LineThickness

1 *Font-attributes* can be one or more of the following: FONTFAMILY=, FONTSIZE=, FONTSTYLE=, FONTWEIGHT=.
2 This style element is valid starting with 2023.03.
3 *TileSet* can be LIGHT, DARK, or HIGHCONTRAST. The default is LIGHT. See "Controlling the Theme of an OpenStreetMap Base Map" in *SAS/GRAPH and Base SAS: Mapping Reference* for details.

Table 20.12 *Style Elements Affecting Graphical Data Representation*

Style Element	Portion of Graph Affected	Recognized Attributes
GraphBoxMean	Marker for mean	ContrastColor MarkerSize MarkerSymbol
GraphBoxMedian	Line for median	ContrastColor LineStyle LineThickness
GraphBoxWhisker	Box whiskers and serifs	ConstrastColor LineStyle LineThickness
GraphConfidence	Primary confidence lines and bands, colors for bands and lines	ContrastColor Color LineStyle LineThickness MarkerSize MarkerSymbol Transparency
GraphConfidence2	Secondary confidence lines and bands, color for bands, and contrast color for lines	ContrastColor Color LineStyle LineThickness MarkerSize MarkerSymbol Transparency
GraphConnectLine	Line for connecting boxes or bars	ContrastColor LineStyle LineThickness

Style Element	Portion of Graph Affected	Recognized Attributes
GraphCutLine	Cutline attributes for a dendogram	Color LineStyle
GraphDataDefault	Primitives related to non-grouped data items, colors for filled areas, markers, and lines	Color ContrastColor EndColor LineStyle LineThickness NeutralColor MarkerSymbol MarkerSize StartColor
GraphError	Error line or error bar fill, ContrastColor for lines, Color for bar fill	CapStyle Color ContrastColor LineStyle Transparency
GraphFit	Primary fit lines such as a normal density curve	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol
GraphFit2	Secondary fit lines such as a kernel density curve	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol
GraphFinal	Final data for the waterfall chart. Color applies to filled areas.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol

Style Element	Portion of Graph Affected	Recognized Attributes
GraphInitial	Initial data for the waterfall chart. Color applies to filled areas.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol
GraphMissing	Properties for graph items representing missing values	Color ContrastColor FillPattern LineStyle LineThickness MarkerSymbol MarkerSize Transparency
GraphOther	Other data for the graph. Color applies to filled areas.	Color ContrastColor FillPattern LineStyle LineThickness MarkerSize MarkerSymbol
GraphOverflow	Overflow data for the graph. Color applies to filled areas. ContrastColor applies to markers and lines.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol
GraphOutlier	Outlier data for the graph	ContrastColor Color MarkerSize MarkerSymbol LineStyle LineThickness
GraphPrediction	Prediction lines	ContrastColor Color

Style Element	Portion of Graph Affected	Recognized Attributes
		LineStyle LineThickness MarkerSize MarkerSymbol Transparency
GraphPredictionLimits	Fills for prediction limits	ContrastColor Color LineStyle LineThickness MarkerSize MarkerSymbol Transparency
GraphSelection	For interactive graphs, visual properties of selected item. Color for selected fill area, ContrastColor for selected marker or line.	ContrastColor Color LineStyle LineThickness MarkerSymbol MarkerSize
GraphUnderflow	Underflow data for the graph. Color applies to filled areas. ContrastColor applies to markers and lines.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol
ThreeColorAltRamp	Line contours, markers, and data labels with segmented range color response	EndColor NeutralColor StartColor
ThreeColorRamp	Gradient contours, surfaces, markers, and data labels with continuous color response	EndColor NeutralColor StartColor
TwoColorAltRamp	Line contours, markers, and data labels with segmented range color response	EndColor StartColor

Style Element	Portion of Graph Affected	Recognized Attributes
TwoColorRamp	Gradient contours, surfaces, markers, and data labels with continuous color response	EndColor StartColor

Table 20.13 *Graphical Style Elements: Data Related (Grouped)*

Style Elements	Portion of Graph Affected	Attributes Defined
GraphData1 GraphData2 GraphData3 GraphData4 GraphData5 GraphData6 GraphData7	Primitives related to the first seven grouped data items. Color applies to filled areas. ContrastColor applies to markers and lines.	Color ContrastColor FillPattern[1] LineStyle MarkerSymbol
GraphData8 GraphData9 GraphData10 GraphData11	Primitives related to the 8th through 11th grouped data items.	Color ContrastColor FillPattern[1] LineStyle MarkerSymbol[2]
GraphData12	Primitives related to the 12th grouped data item.	Color ContrastColor FillPattern[1] MarkerSymbol[2]
GraphData13[3] GraphData14 GraphData15	Primitives related to the 13th through 15th grouped data items.	FillPattern[1] MarkerSymbol

1 Style attribute FillPattern in GraphData1–GraphData11 is also available with the DEFAULT ODS style and all styles that are derived from it.

2 Style attribute MarkerSymbol in these style elements is defined for styles JOURNAL2 and JOURNAL3 only.

3 Style elements GraphData13–GraphData15 are available only with the JOURNAL2 and JOURNAL3 styles.

Table 20.14 *Display Style Elements*

Style Element	Portion of Graph Affected	Recognized Attributes	Possible Values
GraphAltBlock	Alternate fill color for block plots	Color	GraphColors("gablock")

Style Element	Portion of Graph Affected	Recognized Attributes	Possible Values
GraphBand	Display options for confidence bands	DisplayOpts	"Fill fillpattern outline"
GraphBar	Display options for bar charts	DisplayOpts	"Connect fill fillpattern outline"
GraphBlock	Fill color for block plots	Color	GraphColors("gblock")
GraphBox	Display options for box plots	DisplayOpts	"Caps connect fill fillpattern mean median notches outliers"
		CapStyle	"Serif"
		Connect	"Mean"
GraphContour	Display options for contours	DisplayOpts	"LabeledLineGradient"
		EndColor	GraphColors('gramp3cend')
		NeutralColor	GraphColors('gramp3cneutral')
		StartColor	GraphColors('gramp3cstart')
GraphEllipse	Display options for confidence ellipses	DisplayOpts	"Fill fillpattern outline"
GraphHighLow	Display options for high-low plots	DisplayOpts	"Fill fillpattern outline"
GraphHistogram	Display options for histograms	DisplayOpts	"Fill fillpattern outline"
GraphPolygon	Display options for polygon plots	DisplayOpts	"Fill fillpattern outline"
GraphSkins	Display skins	DataSkin	"Crisp" "Gloss" "Matte" "None" "Pressed"
		KpiSkin	"Basic" "Modern"

Style Element	Portion of Graph Affected	Recognized Attributes	Possible Values
			"None"
			"Onyx"
			"Satin"

Style Elements Affecting Device-Based Graphics

Device-based graphics are all SAS/GRAPH output where there is a user-specified or default device (DEVICE= option) that controls certain aspects of the graphical output. Supplied device drivers are stored in the Sashelp.Devices catalog. Examples of device drivers are SASPRTC, GIF, WIN, ACTIVEX, PDF, and SVG. Common SAS/GRAPH procedures that produce device-based graphics are GPLOT, GCHART, and GMAP. Most device-based graphics produce a GRSEG catalog entry as output and use the GOPTIONS statement to control the graphical environment.

For complete documentation on the style attributes that can be specified in each style element, see “About Style Attributes” on page 465.

Table 20.15 *Device-Based Graph Style Elements: General Graph Appearance*

Style Element	Portion of Graph Affected	Recognized Attributes
DropShadowStyle	Used with text types	Color
Graph	Graph size and outer border appearance	OutputWidth OutputHeight BorderColor BorderWidth CellPadding CellSpacing
GraphAxisLines	X, Y, and Z-axis lines	Color LineStyle LineThickness
GraphBackground	Background of the graph	Transparency BackgroundColor Gradient_Direction StartColor

Style Element	Portion of Graph Affected	Recognized Attributes
		EndColor BackgroundImage Image VerticalAlign TextAlign
GraphBorderLines	Border around graph wall, legend border, borders to complete axis frame	Color LineThickness LineStyle
GraphCharts	All charts within the graph	Transparency BackgroundColor Gradient_Direction StartColor EndColor BackgroundImage Image VerticalAlign TextAlign
GraphDataText	Text font and color for point and line labels	Font or *font-attributes*[1] Color
GraphFloor	3-D floor	BackgroundColor Transparency Gradient_Direction StartColor EndColor BackgroundImage Image VerticalAlign TextAlign
GraphFootnoteText	Text font and color for footnotes	Font or *font-attributes*[1] Color
GraphGridLines	Horizontal and vertical grid lines drawn at major tick marks	Color LineStyle LineThickness Transparency displayopts

Style Element	Portion of Graph Affected	Recognized Attributes
GraphGridLines	Horizontal and vertical grid lines drawn at major tick marks	Color LineStyle LineThickness Transparency displayopts
GraphLegendBackground	Background color of the legend	Color FrameBorder Transparency
GraphOutlines	Outline properties for fill areas such as bars, pie slices, and box plots.	Color LineStyle LineThickness
GraphTitle1Text	Text font and color for the first title	Font or *font-attributes*[1] Color
GraphTitleText	Text font and color for titles subsequent to the first title	Font or *font-attributes*[1] Color
GraphValueText	Text font and color for axis tick values and legend values	Font or *font-attributes*[1] Color
GraphWalls	Vertical walls bounded by axes	Transparency BackgroundColor Gradient_Direction StartColor EndColor BackgroundImage Image

1 *Font-attributes* can be one of the following: FONTFAMILY=, FONTSIZE=, FONTSTYLE=, FONTWEIGHT=.

Table 20.16 *Style Elements Affecting Device-Based Non-Grouped Graphical Data Representation*

Style Element	Portion of Graph Affected	Default Attributes
GraphCutLine	Cutline attributes for a dendogram	Color LineStyle
GraphFinal	Final data for the waterfall chart. Color applies to filled areas.	Color ContrastColor LineStyle

Style Element	Portion of Graph Affected	Default Attributes
		LineThickness MarkerSize MarkerSymbol TextColor
GraphInitial	Initial data for the waterfall chart. Color applies to filled areas.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol TextColor
GraphOther	Other data for the graph. Color applies to filled areas.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol TextColor
GraphOverflow	Overflow data for the graph. Color applies to filled areas. ContrastColor applies to markers and lines.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol TextColor
GraphUnderflow	Underflow data for the graph. Color applies to filled areas. ContrastColor applies to markers and lines.	Color ContrastColor LineStyle LineThickness MarkerSize MarkerSymbol TextColor
ThreeColorAltRamp	Line contours, markers, and data labels with segmented range color response	StartColor NeutralColor EndColor

Style Element	Portion of Graph Affected	Default Attributes
ThreeColorRamp	Gradient contours, surfaces, markers, and data labels with continuous color response	StartColor NeutralColor EndColor
TwoColorAltRamp	Line contours, markers, and data labels with segmented range color response	StartColor EndColor
TwoColorRamp	Gradient contours, surfaces, markers, and data labels with continuous color response	StartColor EndColor

Table 20.17 *Style Elements Affecting Device-Based Grouped Graphical Data Representation*

Style Element	Portion of Graph Affected	Default Attributes
GraphData1	Primitives related to 1st grouped data items. Color applies to filled areas. ContrastColor applies to markers and lines.	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor
GraphData2	Primitives related to 2nd grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor

Style Element	Portion of Graph Affected	Default Attributes
GraphData3	Primitives related to 3rd grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor
GraphData4	Primitives related to 4th grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor
GraphData5	Primitives related to 5th grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor
GraphData6	Primitives related to 6th grouped data items	BackGroundImage ContrastColor Color

Style Element	Portion of Graph Affected	Default Attributes
		EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor
GraphData7	Primitives related to 7th grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor
GraphData8	Primitives related to 8th grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor
GraphData9	Primitives related to 9th grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle

Style Element	Portion of Graph Affected	Default Attributes
		LineThickness MarkerSize MarkerSymbol StartColor
GraphData10	Primitives related to 10th grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor
GraphData11	Primitives related to 11th grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol StartColor
GraphData12	Primitives related to 12th grouped data items	BackGroundImage ContrastColor Color EndColor Gradient_Direction Image LineStyle LineThickness MarkerSize MarkerSymbol

Style Element	Portion of Graph Affected	Default Attributes
		StartColor

21

Style Attributes

Overview 807

Style Attributes Tables 808
- Overview 808
- EPUB Destination 808
- Excel Destination 809
- HTML5 Destination 812
- Markup Family of Destinations 817
- PowerPoint Destination 820
- PDF Family of Destinations 824
- RTF Family of Destinations 828
- Word Destination 833

Style Attributes Detailed Information 838

Style Attributes Values 874

Overview

Style attributes influence the characteristics of individual cells, tables, documents, graphs, and HTML frames. Style attributes exist within style elements and are specified by the STYLE statement on page 462 or the CLASS statement on page 456. The default value for an attribute depends on the style that is in use. For information about styles, style elements, and style attributes, see "Understanding Styles, Style Elements, and Style Attributes" on page 441. For information about using style attributes with ODS Statistical Graphics, see the chapter on controlling the appearance of your graphics in *SAS Graph Template Language: User's Guide*.

Style attributes can be supplied by SAS or user-defined. Style attributes can be referenced with a style reference. See "Understanding Style References " on page 448 and "*style-reference*" on page 879 for more information.

The implementation of an attribute depends on the ODS destination that formats the output. When creating HTML output, the implementation of an attribute depends on the browser that is used.

For a list of the values that style attributes can specify, see Chapter 21, “Style Attributes,” on page 807. For a list of style elements that you can specify style attributes in, see Chapter 20, “Style Elements,” on page 777.

Style Attributes Tables

Overview

For usage information about these style attributes, such as aliases, restrictions, and examples, see “Style Attributes Detailed Information” on page 838.

EPUB Destination

The EPUB destination includes the ODS EPUB Statement.

Table 21.1 *Documents*

Task	Attribute
Specify the minimum number of lines of text that must appear in a paragraph before it is forced to move to another page	“ORPHANS=integer”
Specify the number of lines of text that must appear at the top of a page if a paragraph is separated by a page break	“WINDOWS=integer”

Table 21.2 *Tables*

Task	Attribute
Specify whether the border is collapsed or separated	“BORDERCOLLAPSE=COLLAPSE | SEPARATE”
Specify the text to show in a data tip for various reporting elements, including tables and table cells.	“FLYOVER="string"”
Specify an ID for the table or table cell	“HTMLID="string"”

Task	Attribute
Specify the minimum number of lines of text that must appear in a paragraph before it is forced to move to another page	"ORPHANS=integer"
Specify the number of lines of text that must appear at the top of a page if a paragraph is separated by a page break	"WINDOWS=integer"

Excel Destination

The ODS destination for Excel includes the ODS EXCEL Statement.

Table 21.3 Document

Task	Attribute
Specify how to handle leading spaces and line breaks in table cells and HTML documents	"ASIS=ON | OFF"
Specify the string to use for the bullets in the contents file	"LISTSTYLETYPE=bullet-type"
Specify how to handle space characters in table cells	"NOBREAKSPACE=ON | OFF "
Specify how less-than signs (<), greater-than signs (>), and ampersands (&) are interpreted in table cells	"PROTECTSPECIALCHARS=ON | OFF | AUTO"

Table 21.4 Graphs

Task	Attribute
Specify the size of the font for tables, table cells, and graphs	"FONTSIZE=dimension | size"
Specify the style of the font for tables, table cells, and graphs. In many cases, italic and slant map to the same font.	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify justification in tables, table cells, and graphs	"TEXTALIGN=alignment"

Task	Attribute
Change the visual presentation of the text	"TEXTDECORATION=presentation–options"
Specify the number of spaces that the first line of output is indented	"TEXTINDENT=n"

Table 21.5 Tables

Task	Attribute
Specify how to handle leading spaces and line breaks in table cells and HTML documents	"ASIS=ON | OFF"
Specify the color of the background of tables, table cells, or graphs	"BACKGROUNDCOLOR=color"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"
Specify the color of the bottom border of the table	"BORDERBOTTOMCOLOR=color "
Specify the line style of the bottom border of the selected cell	"BORDERBOTTOMSTYLE=line-style"
Specify the width of the bottom border of a table or table cell	"BORDERBOTTOMWIDTH=dimension "
Specify the color of the border in a table or table cell if the border is just one color	"BORDERCOLOR=color"
Specify the color of the left border of a table	"BORDERLEFTCOLOR=color"
Specify the line style of the left border of the specified table cell	"BORDERLEFTSTYLE=line-style"
Specify the width of the left border of the table	"BORDERLEFTWIDTH=dimension"
Specify the color of the right border of the table	"BORDERRIGHTCOLOR=color "
Specify the line style of the right border of the selected cell	"BORDERRIGHTSTYLE=line-style"

Task	Attribute
Specify the width of the right border of the table	"BORDERRIGHTWIDTH=dimension "
Specify the border style of one or more borders	"BORDERSTYLE=line-style-1 <, ..., line-style-4 >"
Specify the color of the top border of a table or table cell	"BORDERTOPCOLOR=color"
Specify the line style of the top border of the specified table cell	"BORDERTOPSTYLE=line-style"
Specify the width of the top border of the table	"BORDERTOPWIDTH=dimension"
Specify the width of the border of the table	"BORDERWIDTH=dimension"
Specify the text to show in a data tip for table cells	"FLYOVER="string""
Specify the font to use in table cells and graphs	"FONTFAMILY="string-1<…, string-n>""
Specify the size of the font for tables, table cells, and graphs	"FONTSIZE=dimension | size"
Specify the style of the font for tables, table cells, and graphs. In many cases, italic and slant map to the same font.	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"
Specify how to handle space characters in table cells	"NOBREAKSPACE=ON | OFF "
Specify text to place after the table cell or table	"POSTTEXT="string""
Specify text to place before the table cell or table	"PRETEXT="string""
Specify how less-than signs (<), greater-than signs (>), and ampersands (&) are interpreted in table cells	"PROTECTSPECIALCHARS=ON | OFF | AUTO"
Specify a string which contains one or more of the following: format, formula, mergeacross, rotate, hidden, wrap	"TAGATTR="string""

Task	Attribute
Specify justification in tables, table cells, and graphs	"TEXTALIGN=alignment"
Change the visual presentation of the text	"TEXTDECORATION=presentation–options"
Specify the number of spaces that the first line of output is indented	"TEXTINDENT=n"
Specify a URL to link to from various reporting elements, including table cells	"URL="uniform-resource-locator""
Specify vertical justification	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "
Specify the width of a table cell, table, line, or a graph	"WIDTH=dimension "

HTML5 Destination

The ODS HTML5 Statement belongs to the Markup Family of destinations.

Table 21.6 *HTML Document*

Task	Attribute
Specify the position of the background of the tables, table cells, or graphs	"BACKGROUNDPOSITION=position"
Specify whether to put a scroll bar in the frame that references the body file	"BODYSCROLLBAR=YES | NO | AUTO"
Specify the width of the frame that displays the body file in the HTML frame file	"BODYSIZE=dimension | dimension% | * "
Specify whether to put a scroll bar in the frames in the frame file that display the contents and the page files	"CONTENTSCROLLBAR=YES | NO | AUTO"
Specify whether to put a border around the frame for an HTML file that uses frames	"FRAMEBORDER=ON | OFF"
Specify the height of a table cell, graph, or graphics in an HTML document [1]	"HEIGHT=dimension "

Task	Attribute
Specify whether to make the entry in the table of contents a link to the body file	"LISTENTRYANCHOR=ON | OFF "
Specify a path or URL to an image file to be used as the bullet for list items.	"LISTSTYLEIMAGE=string"
Specify the bottom margin for the HTML document	"MARGINBOTTOM=dimension"
Specify the left margin for the HTML document	"MARGINLEFT=dimension"
Specify the right margin for the HTML document	"MARGINRIGHT=dimension"
Specify the top margin for the HTML document	"MARGINTOP=dimension"
Specify the color of the link-focus indicator outline	"OUTLINECOLOR=color"
Specify the line style of the link-focus indicator outline	"OUTLINESTYLE=line-style"
Specify the line width of the link-focus indicator outline	"OUTLINEWIDTH=dimension"
Specify HTML to place at page breaks in an HTML document	"PAGEBREAKHTML="string""
Specify the width of a table cell, table, line, or a graph	"WIDTH=dimension "

Table 21.7 *Output and Text*

Task	Attribute
Specify the position of the background of the tables, table cells, or graphs	"BACKGROUNDPOSITION=position"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"
Specify whether an image is repeated horizontally, vertically, both, or not repeated	"BACKGROUNDREPEAT=option"

Task	Attribute
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"
Specify the height of a table cell, graph, or graphics in an HTML document [1]	"HEIGHT=dimension "
Specify justification in tables, table cells, and graphs	"TEXTALIGN=alignment"

Table 21.8 Tables

Task	Attribute
Specify how to handle leading spaces and line breaks in table cells and HTML documents	"ASIS=ON | OFF"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"
Specify the position of the background of the tables, table cells, or graphs	"BACKGROUNDPOSITION=position"
Specify whether an image is repeated horizontally, vertically, both, or not repeated	"BACKGROUNDREPEAT=option"
Specify the color of the bottom border of the table	"BORDERBOTTOMCOLOR=color "
Specify the line style of the bottom border of the selected cell	"BORDERBOTTOMSTYLE=line-style"
Specify the width of the bottom border of the table	"BORDERBOTTOMWIDTH=dimension "
Specify whether the border is collapsed or separated	"BORDERCOLLAPSE=COLLAPSE | SEPARATE"
Specify the color of the border in a table or table cell if the border is just one color	"BORDERCOLOR=color"
Specify the lighter color to use in a border that uses two colors to create a three-dimensional effect	"BORDERCOLORLIGHT=color"
Specify the color of the left border of a table	"BORDERLEFTCOLOR=color"

Task	Attribute
Specify the line style of the left border of the specified table cell	"BORDERLEFTSTYLE=line-style"
Specify the width of the left border of the table	"BORDERLEFTWIDTH=dimension"
Specify the color of the right border of the table	"BORDERRIGHTCOLOR=color "
Specify the line style of the right border of the selected cell	"BORDERRIGHTSTYLE=line-style"
Specify the width of the right border of the table	"BORDERRIGHTWIDTH=dimension "
Specify the thickness of the spacing between cells in a table	"BORDERSPACING=dimension"
Specify the border style of one or more borders	"BORDERSTYLE=line-style-1 <, ..., line-style-4 >"
Specify the color of the top border of the table	"BORDERTOPCOLOR=color"
Specify the line style of the top border of the specified table cell	"BORDERTOPSTYLE=line-style"
Specify the width of the top border of the table	"BORDERTOPWIDTH=dimension"
Specify the width of the border of the table	"BORDERWIDTH=dimension"
Specify the amount of white space on each of the four sides of the content in a table cell	"CELLPADDING=dimension | dimension%"
Specify the font to use in table cells and graphs	"FONTFAMILY="string-1<…, string-n>""
Specify the size of the font for tables, table cells, and graphs	"FONTSIZE=dimension | size"
Specify the style of the font for tables, table cells, and graphs	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"

Task	Attribute
Specify the text to show in a data tip for various reporting elements, including tables and table cells.	“FLYOVER="string"”
Specify the height of a table cell, graph, or graphics in an HTML document [1]	“HEIGHT=dimension ”
Specify the window or frame in which to open the target of the link	“HREFTARGET="target" ”
Specify individual attributes and values for a table or table cell in an HTML document	“HTMLSTYLE="string"”
Specify the amount of white space between the content of the table cell and the border	“PADDING=dimension | dimension%”
Specify the amount of white space on the bottom of the content of the table cell	“PADDINGBOTTOM=dimension | dimension %”
Specify the amount of white space on the left side of the content of the table cell	“PADDINGLEFT=dimension | dimension%”
Specify the amount of white space on the right side of the content of the table cell	“PADDINGRIGHT=dimension | dimension%”
Specify the amount of white space on the top of the content of the table cell	“PADDINGTOP=dimension | dimension%”
Specify the HTML code to place after the table or table cell	“POSTHTML="string"”
Specify an image to place after the table or table cell	“POSTIMAGE="external-file" | fileref | POSTIMAGE="external-file? DESC=alternative-text””
Specify text to place after the table cell or table	“POSTTEXT="string"”
Specify the HTML code to place before the table or table cell	“PREHTML="string"”
Specify an image to place before the table or table cell	“PREIMAGE="external-file" | fileref | PREIMAGE="external-file?DESC=alternative-text"”
Specify text to place before the table cell or table	“PRETEXT="string"”

Task	Attribute
Specify how less-than signs (<), greater-than signs (>), and ampersands (&) are interpreted in table cells	"PROTECTSPECIALCHARS=ON | OFF | AUTO"
Specify the types of rules to use in tables	"RULES=rule-type"
Specify justification in tables, table cells, and graphs	"TEXTALIGN=alignment"
Change the visual presentation of the text	"TEXTDECORATION=presentation–options"
Specify the number of spaces that the first line of output is indented	"TEXTINDENT=n"
Specify a URL to link to from various reporting elements, including table cells	"URL="uniform-resource-locator""
Specify vertical justification	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "

Markup Family of Destinations

The markup family of destinations includes the following ODS statements:

- ODS CSV Statement
- ODS HTML5 Statement

The following tables contain all of the style attributes for the Markup Family and the style attributes that are specific to the Markup Family of destinations.

Table 21.9 *Document*

Task	Attribute
Specify whether styles used in an HTML document are used in CSS style files	"ABSTRACT=ON | OFF"

Table 21.10 *Output and Text*

Task	Attribute
Specify the alternate colors for maps	"CONTRASTCOLOR=color"
Specify the entire doctype declaration for the HTML document	"DOCTYPE="string""

Task	Attribute
Place a rule of the specified width into the space around the text (or entire cell if there is no text) in a table where white space would otherwise appear	"FILLRULEWIDTH=dimension"
Specify the text to show in a data tip for the table cell	"FLYOVER="string""
Specify a font definition to use in tables, table cells, and graphs	"FONT=font-definition|FONT=font-definition | HEADING_FONT | BODY_FONT"
Specify the font to use in table cells and graphs	"FONTFAMILY="string-1<…, string-n>""
Specify the size of the font for tables, table cells, and graphs	"FONTSIZE=*dimension* | *size*" (p. 855)
Specify the style of the font for tables, table cells, and graphs	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"
Specify the font width of tables, table cells, and graphs compared to the width of the usual design of the table, table cell, or graph	"FONTWIDTH=relative-width"
Specify the height of a table cell, graph, or graphics in an HTML document [1]	"HEIGHT=dimension "
Specify the image to appear in a graph	"IMAGE="external-file" |IMAGE="external-file?DESC=alternative-text""
Specify the start fill color for a graph	"STARTCOLOR=color"
Specify justification in tables, table cells, and graphs	"TEXTALIGN=alignment"
Specify a transparency level for graphs	"TRANSPARENCY=dimension"
Specify vertical justification	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "
Specify the width of a table cell, table, line, or a graph	"WIDTH=dimension "

Table 21.11 Tables

Task	Attribute
Specify how to handle leading spaces and line breaks in an HTML document	"ASIS=ON | OFF"
Specify the color of the background of tables, table cells, or graphs	"BACKGROUNDCOLOR=color"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"
Specify the position of the background of the tables, table cells, or graphs	"BACKGROUNDPOSITION=position"
Specify whether an image is repeated horizontally, vertically, both, or not repeated	"BACKGROUNDREPEAT=option"
Specify the color of the border in a table or table cell if the border is just one color	"BORDERCOLOR=color"
Specify the width of the table border	"BORDERWIDTH=dimension"
Specify the color of the bottom of the table	"BORDERBOTTOMCOLOR=color "
Specify the line style of the bottom of the selected cell	"BORDERBOTTOMSTYLE=line-style"
Specify the width of the bottom of the table	"BORDERBOTTOMWIDTH=dimension "
Specify the lighter color to use in a table that uses two colors to create a three-dimensional effect	"BORDERCOLORLIGHT=color"
Specify the color of the left of a table	"BORDERLEFTCOLOR=color"
Specify the line style of the left of the specified table cell	"BORDERLEFTSTYLE=line-style"
Specify the width of the left of the table	"BORDERLEFTWIDTH=dimension"
Specify the color of the right of the table	"BORDERRIGHTCOLOR=color "
Specify the line style of the right of the selected cell	"BORDERRIGHTSTYLE=line-style"
Specify the width of the right of the table	"BORDERRIGHTWIDTH=dimension "

Task	Attribute
Specify the thickness of the spacing between cells in a table	"BORDERSPACING=dimension"
Specify the style of one or more sides of a table	"BORDERSTYLE=line-style-1 <, ..., line-style-4 >"
Specify the color of the top of the table	"BORDERTOPCOLOR=color"
Specify the line style of the top of the specified table cell	"BORDERTOPSTYLE=line-style"
Specify the width of the top of the table	"BORDERTOPWIDTH=dimension"
Specify the amount of white space on each of the four sides of the content in a table cell	"CELLPADDING=dimension | dimension%"
Specify the name of the style sheet class to use in an HTML document for the table or table cell	"CLASS="string""
Specify the window or frame in which to open the target of the link	"HREFTARGET="target" "
Specify an ID for the table or table cell	"HTMLID="string""
Specify an image to place before the table or table cell	"PREIMAGE="external-file" | fileref | PREIMAGE="external-file?DESC=alternative-text""
Specify an image to place after the table or table cell	"POSTIMAGE="external-file" | fileref | POSTIMAGE="external-file?DESC=alternative-text""

PowerPoint Destination

The ODS destination for PowerPoint includes the "ODS POWERPOINT Statement" in *SAS Output Delivery System: User's Guide*.

Table 21.12 *Output and Text*

Task	Attribute
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"

Task	Attribute
Specify whether an image is repeated horizontally, vertically, both, or not repeated	"BACKGROUNDREPEAT=option"
Specify the font to use in table cells and graphs	"FONTFAMILY="string-1<…, string-n>""
Specify the size of the font for tables, table cells, and graphs	"FONTSIZE=dimension | size"
Specify the style of the font for tables, table cells, and graphs	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"
Specify vertical justification	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "

Table 21.13 Document

Task	Attribute
Specify how to handle leading spaces and line breaks in table cells and HTML documents	"ASIS=ON | OFF"
Specify the string to use for the bullets in the contents file	"LISTSTYLETYPE=bullet-type"
Specify the left margin for the PowerPoint slide	"MARGINLEFT=dimension"
Specify the right margin for the PowerPoint slide	"MARGINRIGHT=dimension"

Table 21.14 Tables

Task	Attribute
Specify the color of the background of tables, table cells, or graphs	"BACKGROUNDCOLOR=color"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"
Specify whether an image is repeated horizontally, vertically, both, or not repeated	"BACKGROUNDREPEAT=option"

Task	Attribute
Specify the color of the bottom border of the table	"BORDERBOTTOMCOLOR=color "
Specify the line style of the bottom border of the selected cell	"BORDERBOTTOMSTYLE=line-style"
Specify the width of the bottom border of the table	"BORDERBOTTOMWIDTH=dimension "
Specify the color of the border in a table or table cell if the border is just one color	"BORDERCOLOR=color"
Specify the color of the left border of a table	"BORDERLEFTCOLOR=color"
Specify the line style of the left border of the specified table cell	"BORDERLEFTSTYLE=line-style"
Specify the width of the left border of the table	"BORDERLEFTWIDTH=dimension"
Specify the color of the right border of the table	"BORDERRIGHTCOLOR=color "
Specify the line style of the right border of the selected cell	"BORDERRIGHTSTYLE=line-style"
Specify the width of the right border of the table	"BORDERRIGHTWIDTH=dimension "
Specify the color of the top border of the table	"BORDERTOPCOLOR=color"
Specify the line style of the top border of the specified table cell	"BORDERTOPSTYLE=line-style"
Specify the width of the top border of the table	"BORDERTOPWIDTH=dimension"
Specify the width of the border of the table	"BORDERWIDTH=dimension"
Specify the amount of white space on each of the four sides of the content in a table cell	"CELLPADDING=dimension | dimension%"
Specify the color of the foreground in tables, table cells, or graphs, which is primarily the color of text	"COLOR=color"

Task	Attribute
Specify the hyperlink to show in a data tip for tables and table cells	"FLYOVER="string""
Specify the font to use in table cells and graphs	"FONTFAMILY="string-1<…, string-n>""
Specify the size of the font for tables, table cells, and graphs	"FONTSIZE=dimension | size"
Specify the style of the font for tables, table cells, and graphs	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"
Specify the type of frame to use on a table	"FRAME=frame-type"
Specify the amount of white space on the bottom of the content of the table cell	"PADDINGBOTTOM=dimension | dimension%"
Specify the amount of white space on the left side of the content of the table cell	"PADDINGLEFT=dimension | dimension%"
Specify the amount of white space on the right side of the content of the table cell	"PADDINGRIGHT=dimension | dimension%"
Specify the amount of white space on the top of the content of the table cell	"PADDINGTOP=dimension | dimension%"
Specify the width of the left of the table	"BORDERLEFTWIDTH=dimension"
Specify an image to place after the table	"POSTIMAGE="external-file" | fileref | POSTIMAGE="external-file?DESC=alternative-text""
Specify an image to place before the table	"PREIMAGE="external-file" | fileref | PREIMAGE="external-file?DESC=alternative-text""
Specify justification in tables, table cells, and graphs	"TEXTALIGN=alignment"
Change the visual presentation of the text	"TEXTDECORATION=presentation–options"
Specify the number of spaces that the first line of output is indented	"TEXTINDENT=n"

Task	Attribute
Specify a URL to link to from various reporting elements, including table cells in individual cells	"URL="uniform-resource-locator""
Specify vertical justification in individual cells	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "

PDF Family of Destinations

The PDF family of destinations includes the "ODS PDF Statement" in *SAS Output Delivery System: User's Guide*.

Table 21.15 *Document*

Task	Attribute
Place a rule of the specified width into the space around the text (or entire cell if there is no text) in a table where white space would otherwise appear	"FILLRULEWIDTH=dimension"
Specify the color for the links in an PDF document that have not yet been visited	"LINKCOLOR=color"
Specify the string to use for the bullets in the contents file	"LISTSTYLETYPE=bullet-type"
Specify the bottom margin for the HTML document	"MARGINBOTTOM=dimension"
Specify the left margin for the PDF document	"MARGINLEFT=dimension"
Specify the right margin for the HTML document	"MARGINRIGHT=dimension"
Specify the top margin for the PDF document	"MARGINTOP=dimension"

Table 21.16 *Output and Text*

Task	Attribute
Specify the color of the background of tables, table cells, or graphs	"BACKGROUNDCOLOR=color"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"

Task	Attribute
Specify the position of the background of the tables, table cells, or graphs	"BACKGROUNDPOSITION=position"
Specify the color of the foreground in tables, table cells, or graphs, which is primarily the color of text	"COLOR=color"
Specify the alternate colors for maps	"CONTRASTCOLOR=color"
Specify a font definition to use in tables, table cells, and graphs	"FONT=font-definition|FONT=font-definition | HEADING_FONT | BODY_FONT"
Specify the font to use in table cells and graphs	"FONTFAMILY="string-1<…, string-n>""
Specify the size of the font for tables, table cells, and graphs	"FONTSIZE=dimension | size"
Specify the style of the font for tables, table cells, and graphs	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"
Specify the font width of tables, table cells, and graphs compared to the width of the usual design of the table, table cell, or graph	"FONTWIDTH=relative-width"
Specify the height of a table cell, graph, or graphics in an PDF document [1]	"HEIGHT=dimension "
Specify the image to appear in a graph	"IMAGE="external-file" |IMAGE="external-file?DESC=alternative-text""
Specify justification in tables, table cells, and graphs	"TEXTALIGN=alignment"
Specify vertical justification	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "

Table 21.17 *Tables*

Task	Attribute
Specify how to handle leading spaces and line breaks in an PDF document	"ASIS=ON | OFF"

Task	Attribute
Specify the color of the background of tables, table cells, or graphs	"BACKGROUNDCOLOR=color"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"
Specify the position of the background of the tables, table cells, or graphs	"BACKGROUNDPOSITION=position"
Specify the color of the bottom of the table	"BORDERBOTTOMCOLOR=color "
Specify the line style of the bottom of the selected cell	"BORDERBOTTOMSTYLE=line-style"
Specify the width of the bottom of the table	"BORDERBOTTOMWIDTH=dimension "
Specify the color of the border in a table or table cell if the border is just one color	"BORDERCOLOR=color"
Specify the width of the right of the table	"BORDERRIGHTWIDTH=dimension "
Specify the thickness of the spacing between cells in a table	"BORDERSPACING=dimension"
Specify the color of the top of the table	"BORDERTOPCOLOR=color"
Specify the width of the top of the table	"BORDERTOPWIDTH=dimension"
Specify the width of the table border	"BORDERWIDTH=dimension"
Specify the amount of white space on each of the four sides of the content in a table cell	"CELLPADDING=dimension | dimension%"
Specify the color of the foreground in tables, table cells, or graphs, which is primarily the color of text	"COLOR=color"
Specify the text to show in a data tip for the table cell	"FLYOVER="string""
Specify a font definition to use in tables, table cells, and graphs	"FONT=font-definition|FONT=font-definition | HEADING_FONT | BODY_FONT"
Specify the font to use in table cells and graphs	"FONTFAMILY="string-1<…, string-n>""

Task	Attribute
Specify the size of the font for tables, table cells, and graphs	“FONTSIZE=dimension | size”
Specify the style of the font for tables, table cells, and graphs	“FONTSTYLE=ITALIC | ROMAN | SLANT”
Specify the font weight of tables, table cells, and graphs	“FONTWEIGHT=weight”
Specify the font width of tables, table cells, and graphs compared to the width of the usual design of the table, table cell, or graph	“FONTWIDTH=relative-width”
Specify the type of frame to use on a table	“FRAME=frame-type”
Specify the height of a table cell, graph, or graphics in an PDF document [1]	“HEIGHT=dimension ”
Specify how to handle space characters in table cells	“NOBREAKSPACE=ON | OFF ”
Specify the amount of white space on the bottom of the content of the table cell	“PADDINGBOTTOM=dimension | dimension%”
Specify the amount of white space on the left side of the content of the table cell	“PADDINGLEFT=dimension | dimension%”
Specify the amount of white space on the right side of the content of the table cell	“PADDINGRIGHT=dimension | dimension%”
Specify the amount of white space on the top of the content of the table cell	“PADDINGTOP=dimension | dimension%”
Specify an image to place after the table or table cell	“POSTIMAGE="external-file" | fileref | POSTIMAGE="external-file?DESC=alternative-text””
Specify text to place after the table cell or table	“POSTTEXT="string"”
Specify an image to place before the table or table cell	“PREIMAGE="external-file" | fileref | PREIMAGE="external-file?DESC=alternative-text"”
Specify text to place before the table cell or table	“PRETEXT="string"”

Task	Attribute
Specify how less-than signs (<), greater-than signs (>), and ampersands (&) are interpreted in table cells	"PROTECTSPECIALCHARS=ON | OFF | AUTO"
Specify the types of rules to use in tables	"RULES=rule-type"
Specify justification in tables, table cells, and graphs	"TEXTALIGN=alignment"
Change the visual presentation of the text	"TEXTDECORATION=presentation–options"
Specify the number of spaces that the first line of output is indented	"TEXTINDENT=n"
Specify a URL to link to from various reporting elements, including table cells	"URL="uniform-resource-locator""
Specify vertical justification	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "

RTF Family of Destinations

The RTF family of destinations includes the following statements:

- ODS RTF Statement
- ODS TAGSETS.RTF Statement

Table 21.18 Document

Task	Attribute
Specify the color for the links in an RTF document that have not yet been visited	"LINKCOLOR=color"
Specify the bottom margin for the RTF document	"MARGINBOTTOM=dimension"
Specify the left margin for the RTF document	"MARGINLEFT=dimension"
Specify the right margin for the RTF document	"MARGINRIGHT=dimension"
Specify the top margin for the RTF document	"MARGINTOP=dimension"

Table 21.19 Output and Text

Task	Attribute
Specify the color of the background of tables, table cells, or graphs	"BACKGROUNDCOLOR=color"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"
Specify the position of the background of the tables, table cells, or graphs	"BACKGROUNDPOSITION=position"
Specify the color of the foreground in tables, table cells, or graphs, which is primarily the color of text	"COLOR=color"
Specify the alternate colors for maps	"CONTRASTCOLOR=color"
Specify a font definition to use in tables, table cells, and graphs	"FONT=font-definition|FONT=font-definition | HEADING_FONT | BODY_FONT"
Specify the font to use in table cells and graphs	"FONTFAMILY="string-1<…, string-n>""
Specify the size of the font for tables, table cells, and graphs	"FONTSIZE=dimension | size"
Specify the style of the font for tables, table cells, and graphs	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"
Specify the font width of tables, table cells, and graphs compared to the width of the usual design of the table, table cell, or graph	"FONTWIDTH=relative-width"
Specify the image to appear in a graph	"IMAGE="external-file" |IMAGE="external-file? DESC=alternative-text""
Specify justification in tables, table cells, and graphs	"TEXTALIGN=alignment"
Specify how to evenly distribute text	"TEXTJUSTIFY=INTER_WORD | INTER_CHARACTER"
Specify vertical justification	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "

Table 21.20 Tables

Task	Attribute
Specify how to handle leading spaces and line breaks in an RTF document	"ASIS=ON | OFF"
Specify the color of the background of tables, table cells, or graphs	"BACKGROUNDCOLOR=color"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"
Specify the position of the background of the tables, table cells, or graphs	"BACKGROUNDPOSITION=position"
Specify the color of the bottom border of the table	"BORDERBOTTOMCOLOR=color "
Specify the line style of the bottom border of the selected cell	"BORDERBOTTOMSTYLE=line-style"
Specify the width of the bottom border of the table	"BORDERBOTTOMWIDTH=dimension "
Specify whether the border is collapsed or separated	"BORDERCOLLAPSE=COLLAPSE | SEPARATE"
Specify the color of the border in a table or table cell if the border is just one color	"BORDERCOLOR=color"
Specify the color of the left border of a table	"BORDERLEFTCOLOR=color"
Specify the line style of the left border of the specified table cell	"BORDERLEFTSTYLE=line-style"
Specify the width of the left border of the table	"BORDERLEFTWIDTH=dimension"
Specify the color of the right border of the table	"BORDERRIGHTCOLOR=color "
Specify the line style of the right border of the selected cell	"BORDERRIGHTSTYLE=line-style"
Specify the width of the right border of the table	"BORDERRIGHTWIDTH=dimension "
Specify the thickness of the spacing between cells in a table	"BORDERSPACING=dimension"

Task	Attribute
Specify the border style of one or more borders	"BORDERSTYLE=line-style-1 <, ..., line-style-4 >"
Specify the color of the top border of the table	"BORDERTOPCOLOR=color"
Specify the line style of the top border of the specified table cell	"BORDERTOPSTYLE=line-style"
Specify the width of the top border of the table	"BORDERTOPWIDTH=dimension"
Specify the width of the border of the table	"BORDERWIDTH=dimension"
Specify the amount of white space on each of the four sides of the content in a table cell	"CELLPADDING=dimension | dimension%"
Specify the color of the foreground in tables, table cells, or graphs, which is primarily the color of text	"COLOR=color"
Specify a font definition to use in tables, table cells, and graphs	"FONT=font-definition|FONT=font-definition | HEADING_FONT | BODY_FONT"
Specify the font to use in table cells and graphs	"FONTFAMILY="string-1<…, string-n>""
Specify the size of the font for tables, table cells, and graphs	"FONTSIZE=dimension | size"
Specify the style of the font for tables, table cells, and graphs	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"
Specify the font width of tables, table cells, and graphs compared to the width of the usual design of the table, table cell, or graph	"FONTWIDTH=relative-width"
Specify the type of frame to use on a table	"FRAME=frame-type"
Specify the height of a table cell, graph, or graphics in an RTF document [1]	"HEIGHT=dimension "

Task	Attribute
Specify how to handle space characters in table cells	“NOBREAKSPACE=ON | OFF ”
Specify the amount of white space between the content of the table cell and the border	“PADDING=dimension | dimension%”
Specify the amount of white space on the bottom of the content of the table cell	“PADDINGBOTTOM=dimension | dimension%”
Specify the amount of white space on the left side of the content of the table cell	“PADDINGLEFT=dimension | dimension%”
Specify the amount of white space on the right side of the content of the table cell	“PADDINGRIGHT=dimension | dimension%”
Specify the amount of white space on the top of the content of the table cell	“PADDINGTOP=dimension | dimension%”
Specify an image to place after the table or table cell.	“POSTIMAGE="external-file" | fileref | POSTIMAGE="external-file?DESC=alternative-text””
Specify text to place after the table cell or table.	“POSTTEXT="string"”
Specify an image to place before the table or table cell.	“PREIMAGE="external-file" | fileref | PREIMAGE="external-file?DESC=alternative-text"”
Specify text to place before the table cell or table	“PRETEXT="string"”
Specify how less-than signs (<), greater-than signs (>), and ampersands (&) are interpreted in table cells	“PROTECTSPECIALCHARS=ON | OFF | AUTO”
Specify the types of rules to use in tables	“RULES=rule-type”
Specify justification in tables, table cells, and graphs	“TEXTALIGN=alignment”
Change the visual presentation of the text	“TEXTDECORATION=presentation–options”
Specify the number of spaces that the first line of output is indented	“TEXTINDENT=n”

Task	Attribute
Specify a URL to link to from various reporting elements, including table cells	"URL="uniform-resource-locator""
Specify vertical justification	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "
Specify the width of a table cell, table, line, or a graph	"WIDTH=dimension "

Word Destination

The ODS destination for Word includes the "ODS WORD Statement" in *SAS Output Delivery System: User's Guide*.

Table 21.21 *Documents*

Task	Attribute
Specify the color of the background of the Word document	"BACKGROUNDCOLOR=color"
Specify an image to use as the background	"BACKGROUNDIMAGE="string"| BACKGROUNDIMAGE='gradient' | WATERMARK-IMAGE <?WASHOUT=0 | 1>"
Specify the color of the bottom border of the Word document	"BORDERBOTTOMCOLOR=color "
Specify the line style of the bottom border of the Word document	"BORDERBOTTOMSTYLE=line-style"
Specify the width of the bottom border of the Word document	"BORDERBOTTOMWIDTH=dimension "
Specify the color of the border of the Word document if the border is just one color	"BORDERCOLOR=color"
Specify the color of the left border of the Word document	"BORDERLEFTCOLOR=color"
Specify the line style of the left border of the Word document	"BORDERLEFTSTYLE=line-style"
Specify the width of the left border of the Word document	"BORDERLEFTWIDTH=dimension"

Task	Attribute
Specify the color of the right border of the Word document	"BORDERRIGHTCOLOR=color "
Specify the line style of the right border of the Word document	"BORDERRIGHTSTYLE=line-style"
Specify the width of the right border of the Word document	"BORDERRIGHTWIDTH=dimension "
Specify the border style of one or more borders	"BORDERSTYLE=line-style-1 <, ..., line-style-4 >"
Specify the color of the top border of the Word document	"BORDERTOPCOLOR=color"
Specify the line style of the top border of the Word document	"BORDERTOPSTYLE=line-style"
Specify the width of the top border of the Word document	"BORDERTOPWIDTH=dimension"
Specify the width of the border of the Word document	"BORDERWIDTH=dimension"
Specify how less-than signs (<), greater-than signs (>), and ampersands (&) are interpreted in the Word document	"PROTECTSPECIALCHARS=ON | OFF | AUTO"

Table 21.22 *Cells*

Task	Attribute
Specify the color of the background of table cells	"BACKGROUNDCOLOR=color"
Specify the color of the bottom border of the selected table cell	"BORDERBOTTOMCOLOR=color "
Specify the line style of the bottom border of the selected cell	"BORDERBOTTOMSTYLE=line-style"
Specify the width of the bottom border of a table cell	"BORDERBOTTOMWIDTH=dimension "
Specify the color of the border in a table or table cell if the border is just one color	"BORDERCOLOR=color"
Specify the color of the left border of a table cell	"BORDERLEFTCOLOR=color"

Task	Attribute
Specify the line style of the left border of the specified table cell	"BORDERLEFTSTYLE=line-style"
Specify the width of the left border of the table cell	"BORDERLEFTWIDTH=dimension"
Specify the color of the right border of the table cell	"BORDERRIGHTCOLOR=color "
Specify the line style of the right border of the selected cell	"BORDERRIGHTSTYLE=line-style"
Specify the width of the right border of the table cell	"BORDERRIGHTWIDTH=dimension "
Specify the border style of one or more borders in the table cell	"BORDERSTYLE=line-style-1 <, ..., line-style-4 >"
Specify the color of the top border of a table cell	"BORDERTOPCOLOR=color"
Specify the line style of the top border of the specified table cell	"BORDERTOPSTYLE=line-style"
Specify the width of the top border of the table cell	"BORDERTOPWIDTH=dimension"
Specify the width of the border of the table cell	"BORDERWIDTH=dimension"
Specify the color of the foreground in table cells, which is primarily the color of text	"COLOR=color"
Specify a font definition to use in table cells	"FONT=font-definition|FONT=font-definition | HEADING_FONT | BODY_FONT"
Specify the font to use in table cells	"FONTFAMILY="string-1<…, string-n>""
Specify the size of the font for table cells	"FONTSIZE=dimension | size"
Specify the style of the font for table cells	"FONTSTYLE=ITALIC | ROMAN | SLANT"
Specify the font weight of tables, table cells, and graphs	"FONTWEIGHT=weight"
Specify the bottom margin for the table cell	"MARGINBOTTOM=dimension"
Specify the left margin for the table cell	"MARGINLEFT=dimension"

Task	Attribute
Specify the right margin for the table cell	"MARGINRIGHT=dimension"
Specify the top margin for the table cell	"MARGINTOP=dimension"
Specify aligning the values by the decimal point	TEXTALIGN=DEC
Specify that the words are evenly distributed across the page	"TEXTJUSTIFY=INTER_WORD | INTER_CHARACTER"
Specify vertical justification	"VERTICALALIGN=BOTTOM | MIDDLE | TOP "

Table 21.23 Tables

Task	Attribute
Specify the color of the background of the table	"BACKGROUNDCOLOR=color"
Specify the color of the bottom border of the table	"BORDERBOTTOMCOLOR=color "
Specify the line style of the bottom border of the table	"BORDERBOTTOMSTYLE=line-style"
Specify the width of the bottom border of the table	"BORDERBOTTOMWIDTH=dimension "
Specify the color of the border in the table if the border is just one color	"BORDERCOLOR=color"
Specify the color of the left border of the table	"BORDERLEFTCOLOR=color"
Specify the line style of the left border of the table	"BORDERLEFTSTYLE=line-style"
Specify the width of the left border of the table	"BORDERLEFTWIDTH=dimension"
Specify the color of the right border of the table	"BORDERRIGHTCOLOR=color "
Specify the line style of the right border of the table	"BORDERRIGHTSTYLE=line-style"
Specify the width of the right border of the table	"BORDERRIGHTWIDTH=dimension "

Task	Attribute
Specify the thickness of the spacing between cells in a table	"BORDERSPACING=dimension"
Specify the border style of one or more borders in the table	"BORDERSTYLE=line-style-1 <, ..., line-style-4 >"
Specify the color of the top border of a table	"BORDERTOPCOLOR=color"
Specify the line style of the top border of the table	"BORDERTOPSTYLE=line-style"
Specify the width of the top border of the table	"BORDERTOPWIDTH=dimension"
Specify the width of the border of the table	"BORDERWIDTH=dimension"
Specify the amount of white space on each of the four sides of the content in a table	"CELLPADDING=dimension | dimension%"
Specify the type of frame to use on the table	"FRAME=frame-type"
Specify the bottom margin for the tablet	"MARGINBOTTOM=dimension"
Specify the top margin for the table	"MARGINTOP=dimension"
Specify an image to place after the table	"POSTIMAGE="external-file" | fileref | POSTIMAGE="external-file?DESC=alternative-text""
Specify text to place after the table	"POSTTEXT="string""
Specify an image to place before the table	"PREIMAGE="external-file" | fileref | PREIMAGE="external-file?DESC=alternative-text""
Specify text to place before the table	"PRETEXT="string""
Specify the types of rules to use in the table	"RULES=rule-type"
Specify aligning the values by the decimal point	TEXTALIGN=DEC
Specify the number of spaces that the first line of output is indented	"TEXTINDENT=n"

Style Attributes Detailed Information

To quickly locate style attributes that are valid in your destination, see the Style Attributes Tables on page 808.

ABSTRACT=ON | OFF
specifies whether styles used in a document are used in CSS style files.

ON
specifies that styles are not used in CSS style files.

OFF
specifies that styles are used in CSS style files.

Valid in Markup family destinations, ODS HTML5 destination, and the ODS EPUB destination

ACTIVELINKCOLOR=*color*
specifies the color that a link changes to after you click it, but before the browser opens that file.

Valid in Markup family destinations

See color style attribute value on page 874

ASIS=ON | OFF
specifies how to handle leading spaces and line breaks in an HTML document.

ON
prints text with leading spaces and line breaks, in the same manner as the LISTING output.

OFF
trims leading spaces and ignores line breaks.

Valid in Markup family, PowerPoint, and Excel

Default OFF

Tip Specify the ASIS= style attribute within the DATA style element to prevent the compressing of blank characters in table cells.

BACKGROUNDCOLOR=*color*
specifies the color of the background of the tables, table cells, or graphs.

Valid in Markup family, PDF, Excel, PowerPoint, RTF, and Word destinations

Alias BACKGROUND=

Restrictions Microsoft (MS) Word documents do not support the HLS (hue lightness saturation) and CYMK (cyan magenta yellow black) colors.

In a Microsoft (MS) Word document, specify *color* as a hexadecimal value, a theme color, or as a color that SAS recognizes.

Interaction The CBACK= option in the SAS/GRAPH GOPTIONS statement overrides the BACKGROUNDCOLOR= attribute.

Notes By default, Microsoft (MS) Word does not print background colors. To enable printing of background colors from MS Word, select **File** ⇨ **Options** ⇨ **Display** ⇨ **Printing options**, and check the box for **Print background colors and images**.

Microsoft (MS) Word supports a transparent background color (#RRGGBBAA).

Microsoft (MS) Word supports the LIGHT1, LIGHT2, DARK1, DARK2, ACCENT1, ACCENT2, ACCENT3, ACCENT4, ACCENT5, and ACCENT6 theme colors. See more information on the Microsoft support website.

Tip Generally, the background color of the table cell overrides the background color of the table. You see the background color for the table only as the space between table cells (see "BORDERSPACING=*dimension*" on page 846).

See color style attribute value on page 874

Examples "Example 1: Creating a Stand-Alone Style" on page 466

"Example 3: Modifying the Default Style with the CLASS Statement" on page 482

CAUTION **Regarding printed documents** Specifying a background color for a document that is to be printed results in excessive use of printer ink.

BACKGROUNDIMAGE="*string*" | BACKGROUNDIMAGE=*'gradient'* | WATERMARK-IMAGE <?WASHOUT=0 | 1>

specifies an image in a table, table cell, or graph to use as the background. Viewers can tile or stretch the image as the background for the HTML table or graph that the procedure creates. For graphs, the specified image is stretched.

string

is the name of a GIF, JPEG, or PNG file, or a URL. Specifying a URL path is allowed only when using non-Microsoft Office destinations. Use a simple filename, a fileref, or a complete path. However, placing all image files in a local directory and then specifying a filename might be the simplest approach.

gradient

Specify as a linear-gradient (LINEAR), radial-gradient (RADIAL), or none (NONE). NONE turns off the background image and returns to the default.

```
ods powerpoint options(backgroundimage=
"linear-gradient(90deg ,#93ACFF, #82FF8E)");
```

Restriction The ODS destination of Word does not support repeating gradients; either linear or radial.

See "LINEAR-GRADIENT (side-or-corner|angle, color-stop [, color-stop]+)" in *SAS Output Delivery System: User's Guide*, and "RADIAL-GRADIENT (< shape > size position, color-stop [, color-stop]+)" in *SAS Output Delivery System: User's Guide*

?WASHOUT
specifies whether the watermark is washed out.

0 the watermark is not washed out

1 the watermark is washed out

Default WASHOUT=1

Valid in Markup family, Excel, PDF, TAGSETS.RTF, Word, and PowerPoint destinations

Restrictions Do not specify a URL for a background image when using Microsoft (MS) Office destinations such as Excel, PowerPoint, and Word.

For the ODS destination of Word, the image must be specified in a raster format. Vector formats are not supported.

MS Excel and MS Word do not support background images in data cells.

Interactions The BACKGROUNDIMAGE= attribute is overridden by the IBACK= and IMAGESTYLE=FIT options in the SAS/GRAPH GOPTIONS statement.

When you apply BACKGROUNDIMAGE= to a cell using the Report Writing Interface, the image adjusts to the size of the cell.

Note By default, Microsoft (MS) Word does not print background images. To enable printing of background images from MS Word, select **File** ⇨ **Options** ⇨ **Display** ⇨ **Printing options**, and check the box for **Print background colors and images**.

Tip You can also specify the image width or height, or both. For example:

```
width=200px; height=200px;
```

By default, the width and height are extracted from the image.

See string attribute value on page 879

BACKGROUNDPOSITION=*position*
specifies the position of the background of the tables, table cells, or graphs.

position can be one of the following:

- BOTTOM
- BOTTOM_CENTER
- BOTTOM_LEFT
- BOTTOM_RIGHT
- CENTER

- CENTER_BOTTOM
- CENTER_CENTER
- CENTER_LEFT
- CENTER_RIGHT
- CENTER_TOP
- LEFT
- LEFT_BOTTOM
- LEFT_CENTER
- LEFT_TOP
- RIGHT
- RIGHT_BOTTOM
- RIGHT_CENTER
- RIGHT_TOP
- TOP
- TOP_CENTER
- TOP_LEFT
- TOP_RIGHT

Valid in Markup family, PDF, and RTF destinations

Default TOP_LEFT

BACKGROUNDREPEAT=*option*
specifies whether an image is repeated horizontally, vertically, both, or not repeated. *option* can be one of the following:

NO_REPEAT
specifies that the image is not repeated.

REPEAT
specifies that the image is repeated both horizontally and vertically.

REPEAT_X
specifies that the image is repeated horizontally.

REPEAT_Y
specifies that the image is repeated vertically.

Valid in Markup family and PowerPoint destinations

Restriction The BACKGROUNDREPEAT= attribute is valid in most markup family destinations.

BODYSCROLLBAR=YES | NO | AUTO
specifies whether to put a scroll bar in the frame that references the body file.

YES
places a scroll bar in the frame that references the body file.

NO
specifies not to put a scroll bar in the frame that references the body file.

AUTO
places a scroll bar in the frame that references the body file only if needed.

Valid in Markup family destinations

Tip Typically, BODYSCROLLBAR= is set to AUTO.

BODYSIZE=*dimension* | *dimension*% | *
specifies the width of the frame that displays the body file in the HTML frame file.

dimension
is a nonnegative number or the width of the frame specified as a percentage of the entire display.

*
specifies to use whatever space is left after displaying the content and page files as specified by the CONTENTSIZE= attribute.

Valid in Markup family destinations

Tip If *dimension* is a nonnegative number, then the unit of measure is pixels.

See dimension attribute value on page 877

For information about the HTML files that ODS creates, see "HTML Links and References Produced by the HTML Destination " in *SAS Output Delivery System: User's Guide*.

BORDERBOTTOMCOLOR=*color*
specifies the color of the bottom border of a table or table cell.

Valid in Excel, Markup family, PDF, PowerPoint, RTF, Measured RTF, and Word destinations

Restriction In a Microsoft (MS) Word document, specify *color* as a hexadecimal value or as a color that SAS recognizes.

Tip You might also need to specify a BORDERBOTTOMWIDTH= attribute to override the style in the ODS destination.

See color style attribute value on page 874

BORDERBOTTOMSTYLE=*line-style*
specifies the line style of the bottom border of the specified table cell.

line-style
can be one of the following:

- DASHED
- DOTTED
- DOUBLE
- GROOVE
- HIDDEN
- INSET
- NONE
- OUTSET

- RIDGE
- SOLID

Valid in Markup family, Excel, PowerPoint, PDF, RTF, Measured RTF, and Word destinations

Tip You might also need to specify the BORDERBOTTOMWIDTH= attribute to override the style in the ODS destination.

BORDERBOTTOMWIDTH=*dimension*
specifies the width of the bottom border of a table or table cell.

For the ODS destination for Excel, when you specify the following dimensions, the thickness of the border appears as follows:

Less than 3pt	Border width is thin.
3pt, but less than 5pt	Border width is medium.
5pt or more	Border width is thick.

Valid in Markup family, Excel, PowerPoint, PDF, RTF, Measured RTF, and Word destinations

See dimension attribute value on page 877

Example "Using BORDERBOTTOMCOLOR with Excel Output" in *SAS Output Delivery System: User's Guide*

BORDERCOLLAPSE=COLLAPSE | SEPARATE
specifies whether the border is collapsed or separated.

Valid in HTML5 and EPUB destinations

Default SEPARATE

BORDERCOLOR=*color*
specifies the border color of a table or table cell. The color is applied to all four borders.

Valid in Markup family, Excel, PowerPoint, PDF, RTF, Measured RTF, and Word destinations

Restriction In a Microsoft (MS) Word document, specify *color* as a hexadecimal value or as a color that SAS recognizes.

See color style attribute value on page 874

BORDERCOLORLIGHT=*color*
in a table or table cell, specifies the lighter color to use in a border that uses two colors to create a three-dimensional effect.

Valid in Markup family and PDF destinations

Interaction The BORDERCOLORLIGHT style attribute is ignored in the creation of HTML4 output because it is not part of the HTML4 standard. To create a color border in HTML4 output, use the BORDERCOLOR= style attribute.

See color style attribute value on page 874

BORDERLEFTCOLOR=*color*
specifies the color of the left border of the table.

Valid in Markup family, Excel, PowerPoint, PDF, RTF, Measured RTF, and Word destinations

Restriction In a Microsoft (MS) Word document, specify *color* as a hexadecimal value or as a color that SAS recognizes.

Interaction Specifying a left border color overrides any color specified with the BORDERCOLOR= option.

Tip You might also need to specify the BORDERLEFTWIDTH= attribute to override the style in the ODS destination.

See color style attribute value on page 874

BORDERLEFTSTYLE=*line-style*
specifies the line style of the left border of the specified table cell.

line-style
can be one of the following:

- DASHED
- DOTTED
- DOUBLE
- GROOVE
- HIDDEN
- INSET
- NONE (use only with the Word destination)
- OUTSET
- RIDGE
- SOLID

Valid in Markup family, Excel, PowerPoint, RTF, Measured RTF, and Word destinations

Restriction In a Microsoft (MS) Word document, specify *color* as a hexadecimal value or as a color that SAS recognizes.

Interaction Specifying a left border style overrides any style specified with the BORDERSTYLE= option.

Tip You might also need to specify the BORDERLEFTWIDTH= attribute to override the style in the ODS destination.

BORDERLEFTWIDTH=*dimension*
specifies the width of the left border of a table or table cell.

Valid in Markup family, Excel, PowerPoint, PDF, RTF, Measured RTF and Word destinations

Interaction Specifying a left border width overrides any width specified with the BORDERWIDTH= option.

See dimension attribute value on page 877

BORDERRIGHTCOLOR=*color*
specifies the color of the right border of a table or table cell.

Valid in Markup family, Excel, PowerPoint, PDF, RTF, Measured RTF, and Word destinations

Restriction In a Microsoft (MS) Word document, specify *color* as a hexadecimal value or as a color that SAS recognizes.

Interaction Specifying a right border color overrides any color specified with the BORDERCOLOR= option.

Tip You might also need to specify the BORDERRIGHTWIDTH= attribute to override the style in the ODS destination.

See color style attribute value on page 874

BORDERRIGHTSTYLE=*line-style*
specifies the line style of the right border of the selected cell.

line-style
can be one of the following:

- DASHED
- DOTTED
- DOUBLE
- GROOVE
- HIDDEN
- INSET
- NONE (use only with the Word destination)
- OUTSET
- RIDGE
- SOLID

Valid in Markup family, Excel, PowerPoint, RTF, Measured RTF, and Word destinations

Restriction In a Microsoft (MS) Word document, specify *color* as a hexadecimal value or as a color that SAS recognizes.

Interaction Specifying a right border style overrides any style specified with the BORDERSTYLE= option.

Tip You might also need to specify the BORDERRIGHTWIDTH= attribute to override the style in the ODS destination.

BORDERRIGHTWIDTH=*dimension*
specifies the width of the right border of the table.

Valid in Markup family, Excel, PowerPoint, PDF, RTF, Measured RTF and Word destinations

Interaction Specifying a right border width overrides any width specified with the BORDERWIDTH= option.

See dimension attribute value on page 877

BORDERSPACING=*dimension*
specifies the vertical and horizontal thickness of the spacing between cells in a table.

Valid in Markup family, RTF, PDF, and Word destinations

Alias CELLSPACING=

Default 0

Interaction If BORDERWIDTH= is nonzero, and if the background color of the table cells contrasts with the background color of the table, then the color of the table cell spacing is determined by the table's background.

See dimension attribute value on page 877

Examples “Example 1: Creating a Stand-Alone Style” on page 466

“Example 3: Modifying the Default Style with the CLASS Statement” on page 482

BORDERSTYLE=*line-style-1 <, ..., line-style-4 >*
specifies the border style of one or more borders.

line-style
is one of the following:

- DASHED
- DOTTED
- DOUBLE
- GROOVE
- HIDDEN
- INSET
- NONE
- OUTSET
- RIDGE
- SOLID

The order in which you specify the style determines the style applied to each side.

Table 21.24 BORDERSTYLE= Option Precedence

Example	Order	Result
borderstyle= double;	All four sides	Double
borderstyle= solid double;	Top and bottom Right and left	Solid Double
borderstyle= none double solid;	Top Right and left Bottom	None Double Solid
borderstyle= none double solid none;	Top Right Bottom Left	None Double Solid None

Valid in Markup family, Excel, RTF, Measured RTF, and Word destinations

BORDERTOPCOLOR=*color*
specifies the color of the top border of a table or table cell.

Valid in Markup family, PDF, RTF, PowerPoint, Excel, Measured RTF and Word destinations

Restriction In a Microsoft (MS) Word document, specify *color* as a hexadecimal value or as a color that SAS recognizes.

Tip Specify the BORDERTOPWIDTH= attribute to override the style in the ODS destination.

See color style attribute value on page 874

BORDERTOPSTYLE=*line-style*
specifies the line style of the top border of the specified table cell.

line-style
can be one of the following:

- DASHED
- DOTTED
- DOUBLE
- GROOVE
- HIDDEN
- INSET
- NONE
- OUTSET
- RIDGE

- SOLID

Valid in Markup family, Excel, PowerPoint, RTF, Measured RTF and Word destinations

Restriction For the RTF destination, specify the BORDERTOPSTYLE= attribute in conjunction with the BORDERTOPWIDTH= attribute to ensure that the style of the top border is the style that you specified.

Tip You might also need to specify the BORDERTOPWIDTH= attribute to override the style in the ODS destination.

BORDERTOPWIDTH=*dimension*
specifies the width of the top border of the table or table cell.

Valid in Markup family, Excel, PowerPoint, PDF, RTF, Measured RTF and Word destinations

See dimension attribute value on page 877

BORDERWIDTH=*dimension*
specifies the width of the table borders. The value of BORDERWIDTH= is applied to all four borders.

Valid in Markup family, Excel, PowerPoint, RTF, PDF and Word destinations

Tip Typically, when BORDERWIDTH=0, the ODS destination sets RULES=NONE (see the discussion about "RULES=*rule-type*" on page 869) and FRAME=VOID (see the discussion about "FRAME=*frame-type*" on page 857).

See dimension attribute value on page 877

Examples "Example 1: Creating a Stand-Alone Style" on page 466

"Example 3: Modifying the Default Style with the CLASS Statement" on page 482

CAPSTYLE=*line-shape*
specifies the shape of the line at the end of a box whisker. *line-shape* can be one of the following:

- "BRACKET"
- "LINE"
- "NONE"
- "SERIF"

Requirement You must enclose *line-shape* in quotation marks.

CELLPADDING=*dimension* | *dimension*%
specifies the amount of white space on each of the four sides of the content in a table cell.

dimension
is a nonnegative number or the amount of white space on each of the four sides of the text in a table cell specified as a percentage of the table.

Valid in Markup family other than HTML5, PowerPoint, PDF, RTF, and Word destinations

Restrictions CELLPADDING= is not valid in the HTML5 destination. All padding is done on the table cells.

In the ODS Destination for PowerPoint, the cellpadding dimension cannot go below 5 pt.

See dimension attribute value on page 877

Example "Example 3: Modifying the Default Style with the CLASS Statement" on page 482

CLASS="*string*"
specifies the name of the style sheet class to use in an HTML document for the table or table cell.

Valid in Markup family

Alias HTMLCLASS=

See string attribute value on page 879

COLOR=*color*
specifies the color of the foreground in tables, table cells, or graphs, which is primarily the color of text.

Valid in Markup family, PDF, PowerPoint, RTF, and Word destinations

Alias FOREGROUND=

Restriction In a Microsoft (MS) Word document, specify *color* as a hexadecimal value, a theme color, or as a color that SAS recognizes.

Interaction The COLOR= attribute is overridden by the CBACK= option in the SAS/GRAPH GOPTIONS statement.

See color style attribute value on page 874

Examples "Example 3: Modifying the Default Style with the CLASS Statement" on page 482

"Example 1: Creating a Stand-Alone Style" on page 466

CONNECT=*connect-line-type*
specifies the characteristics of a box plot connect line. *connect-line-type* can be one of the following:

- "MAX"
- "MEAN"
- "MEDIAN"
- "MIN"
- "Q1"
- "Q3"

Requirement You must enclose *connect-line-type* in quotation marks.

CONTENTSCROLLBAR=YES | NO | AUTO
specifies whether to put a scroll bar in the frames in the frame file that display the contents and the page files. (For information about the HTML files that ODS creates, see "HTML Links and References Produced by the HTML Destination " in *SAS Output Delivery System: User's Guide*.)

YES
places a scroll bar in the frames in the frame file that display the contents and the page files.

NO
specifies not to put a scroll bar in the frames in the frame file that display the contents and the page files.

AUTO
specifies that the browser put a scroll bar on the table of contents frame only if the content in that panel is big enough to require scrolling.

Valid in Markup family destinations

Tip Typically, CONTENTSCROLLBAR= is set to AUTO.

See For information about the HTML files that ODS creates, see "HTML Links and References Produced by the HTML Destination " in *SAS Output Delivery System: User's Guide*.

CONTENT=*html-code*
specifies the HTML code for page breaks.

html-code
is a string of html code that defines page breaks.

Valid in HTML5 and EPUB destinations

Example The following example is from Styles.Daisy:

```
content = '<div class="pagebreak" style="text-align: center;
    page-break-before: avoid;
    page-break-after: always"><span> </span></div>'
```

CONTENTSIZE=*dimension* | *dimension* % | *
specifies the width of the frames in the frame file that display the contents and the page files.

dimension
is a nonnegative number or the width of the frames specified as a percentage of the entire display.

*
specifies to use whatever space is left after displaying the body file as specified by the BODYSIZE= attribute.

Valid in Markup family destinations

Requirement *dimension* % must be a positive number between 0 and 100.

Tip If *dimension* is a nonnegative number, then the unit of measure is pixels.

See dimension attribute value on page 877

"BODYSIZE=*dimension* | *dimension*% | * " on page 842

For information about the HTML files that ODS creates, see "HTML Links and References Produced by the HTML Destination " in *SAS Output Delivery System: User's Guide*.

CONTENTTYPE="*string*"
specifies the value of the content type for pages in an HTML document that is sent directly to a web server rather than to a file.

string
is the content type for the pages.

Requirement *string* must be enclosed in quotation marks.

Tip The value of *string* is usually "text/html".

See string attribute value on page 879

Valid in Markup family destinations

Alias HTMLCONTENTTYPE=

CONTRASTCOLOR=*color*
specifies the alternate colors for maps. The alternate colors are applied to the blocks on region areas in block maps.

Valid in Markup family, RTF, and PDF destinations

See color style attribute value on page 874

DATASKIN=CRISP | GLOSS | MATTE | NONE | PRESSED | SHEEN
specifies the type of skin to apply to plots and charts (other than KPIs) to give them a raised, 3-D appearance.

The DATASKIN= style attribute is valid for the following plots and charts in the Graph Template Language:

- bar charts
- box plots
- bubble plots
- drop lines
- high-low charts
- histograms
- line charts
- needle plots
- pie charts
- polygon plots
- reference lines
- scatter plots
- series plots

- step plots
- vector plots
- waterfall charts

The DATASKIN= style attribute is valid for the following plots and charts in the SG procedures:

- bar charts
- scatter plots
- waterfall charts

Table 21.25 *DATASKIN Values*

NONE 	CRISP 	GLOSS 
MATTE 	PRESSED 	SHEEN 

Restriction The maximum number of skinned graphical elements is limited to 200 per plot in an overlay or prototype layout. When this limit is exceeded for a plot, the specified data skin is not applied to that plot. In that case, use the DATASKINMAX= option in your ODS GRAPHICS statement to increase the maximum limit.

DISPLAYOPTS=*"display-feature"*
specifies one or more display features for ODS graphs. To specify multiple features, enclose the list of features in quotation marks (for example: `displayopts="fill caps mean"`). *"display-feature"* can be one of the following:

CAPS
displays caps at the ends of the whiskers.

Restriction CAPS can be used only for box plots.

CONNECT
displays the line connecting multiple boxes.

Restriction CONNECT can be used only for box plots.

FILL
displays filled boxes, bars, ellipses, and bands.

Restriction FILL can be used only for box plots, histograms, ellipses, and confidence bands.

MEAN
displays the mean symbol within a box.

Restriction MEAN can be used only for box plots.

MEDIAN
displays the median line within the box.

NOTCHES
displays notched boxes.

Restriction NOTCHES can be used only for box plots.

OUTLIERS
displays markers for the outliers.

Restriction OUTLIERS can be used only for box plots.

OUTLINE
displays outlined ellipses and bars.

Restriction OUTLINE can be used only for ellipses, bands, and histograms.

Requirement You must enclose *"display-feature"* in quotation marks.

DOCTYPE="*string*"
specifies the entire doctype declaration for the HTML document, including the opening "<!DOCTYPE" and the closing ">".

string
is the doctype declaration.

Requirement *string* must be enclosed in quotation marks.

See string attribute value on page 879

Valid in Markup family destinations

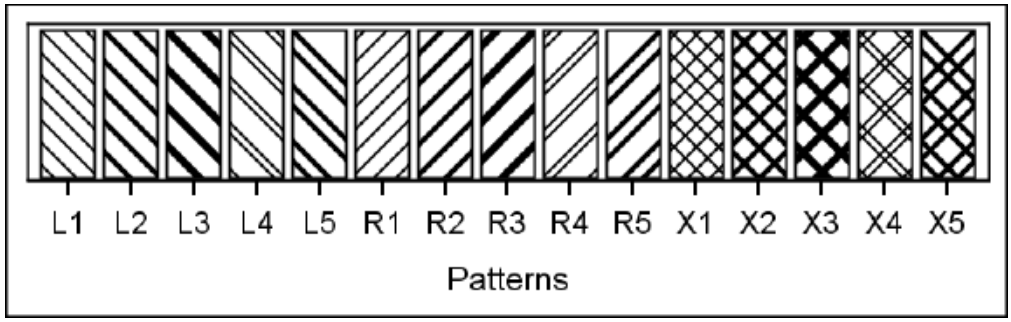
Alias HTMLDOCTYPE=

DROPSHADOW=ON | OFF
specifies whether the drop shadow color for text is displayed.

ENDCOLOR=*color*
specifies the final color used with a two- or three-color ramp.

See color style attribute value on page 874

FILLPATTERN=*fillpattern-value*
specifies the fill pattern to be displayed on the chart. The valid values are: S, E, L1, L2, L3, L4, L5, R1, R2, R3, R4, R5, X1, X2, X3, X4, and X5.



Restriction The FILLPATTERN= attribute is valid for bar charts only.

Tip To display these fill patterns on the bar chart through the style, you must also specify FILLPATTERN as one of the DISPLAYOPTS in the GRAPHBAR style element.

See For a table of style elements and the style attributes that are valid in each one, see "Style Elements Affecting Template-Based Graphics" on page 788 and "Style Elements Affecting Device-Based Graphics" on page 797.

FILLRULEWIDTH=*dimension*
places a rule of the specified width into the space around the text (or entire cell if there is no text) in a table where white space would otherwise appear.

Valid in PDF destinations

Tip If no text is specified, then FILLRULEWIDTH= fills the space around the text with hyphen marks. For example: --this-- or this ------.

See dimension attribute value on page 877

FLYOVER="*string*"
specifies the text to show in a data tip for various reporting elements, including table cells.

string
is the text of the data tip.

Restriction When using the ODS PDF destination, the FLYOVER= style attribute text should not exceed 1024 chars. If the length exceeds 1024, the text is ignored and a warning message is recorded in the SAS log.

Requirement *string* must be enclosed in quotation marks.

Note For destination-specific information for the FLYOVER attribute, see "Style Attributes Tables" on page 808.

See string attribute value on page 879

Valid in Markup family, EPUB, Excel, PowerPoint, Word, and PDF destinations

FONT=*font-definition* | FONT=*font-definition* | HEADING_FONT | BODY_FONT
specifies a font definition to use in tables, table cells, and graphs.

HEADING_FONT
For the ODS Word destination, HEADING_FONT specifies the heading font defined in the scheme. This font is to be used in table cells in the Word document.

BODY_FONT
For the ODS Word destination, BODY_FONT specifies the body font defined in the scheme. This font is to be used in table cells in the Word document.

Valid in Markup family, RTF, PDF, and Word destinations

Tips For a table, the FONT= attribute affects only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the font for the text that appears in the table, set the attribute for a table cell.

If the system does not recognize the font specified, then it refers to the system's default font. This attribute does not accept concatenated fonts. SAS Graph Styles can specify only one font.

See font-definition attribute value on page 877

Example "Example 3: Modifying the Default Style with the CLASS Statement" on page 482

FONTFAMILY="*string-1*<…, *string-n*>"
specifies the font to use in table cells and graphs. If you supply multiple fonts, then the destination device uses the first one that is installed on the system.

string
is the name of the font.

Requirement *string* must be enclosed in quotation marks.

See string attribute value on page 879

Valid in Markup family, Excel, PowerPoint, RTF, PDF, and Word destinations

Alias FONT_FACE=

Tips For a table, the FONTFAMILY= attribute affects only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the font for the text that appears in the table, set the attribute for a table cell.

You cannot be sure what fonts are available to someone who is viewing the output in a browser or printing it on a high-resolution printer. Most devices support the following fonts: Times, Courier, Arial, Helvetica.

Example "Example 1: Creating a Stand-Alone Style" on page 466

FONTSIZE=*dimension* | *size*
specifies the size of the font for tables, table cells, and graphs.

dimension
is a nonnegative number.

Alias FONT_SIZE=

Restriction If you specify a dimension, then specify a unit of measure. Without a unit of measure, the number becomes a relative size.

See dimension attribute value on page 877

size
The value of *size* is relative to all other font sizes in the HTML document.

Range 1 to 7

Valid in Markup family, Excel, PowerPoint, RTF, and PDF destinations

Tip For a table, the FONTSIZE= attribute affects only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and

POSTHTML= attributes. To alter the font for the text that appears in the table, set the attribute for a table cell.

Example "Example 1: Creating a Stand-Alone Style" on page 466

FONTSTYLE=ITALIC | ROMAN | SLANT
specifies the style of the font for tables, table cells, and graphs. In many cases, italic and slant map to the same font.

Valid in Excel, Markup family, PowerPoint, RTF, PDF, and Word destinations

Alias FONT_STYLE=

Tip For a table, the FONTSTYLE= attribute affects only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the font for the text that appears in the table, set the attribute for a table cell.

Examples "Example 1: Creating a Stand-Alone Style" on page 466

"Example 3: Modifying the Default Style with the CLASS Statement" on page 482

FONTWEIGHT=*weight*
specifies the font weight of tables, table cells, and graphs. *weight* is any of the following:

- MEDIUM
- BOLD
- DEMI_BOLD
- EXTRA_BOLD
- LIGHT
- DEMI_LIGHT
- EXTRA_LIGHT.

Valid in Markup family, Excel, PowerPoint, RTF, PDF, and Word destinations

Alias FONT_WEIGHT=

Restriction You cannot be sure what font weights are available to someone who is viewing the output in a browser or printing it on a high-resolution printer. Most devices support only MEDIUM and BOLD, and possibly LIGHT.

Tip For a table, the FONTWEIGHT= attribute affects only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the font for the text that appears in the table, set the attribute for a table cell.

Examples "Example 1: Creating a Stand-Alone Style" on page 466

"Example 1: Creating a Stand-Alone Style" on page 466

FONTWIDTH=*relative-width*

specifies the font width of tables, table cells, and graphs compared to the width of the usual design of the table, table cell, or graph. *relative-width* is any of the following:

- NORMAL
- COMPRESSED
- EXTRA_COMPRESSED
- NARROW
- WIDE
- EXPANDED

Valid in Markup family, RTF, and PDF destinations

Alias FONT_WIDTH=

Restriction Few fonts honor these values.

Tip For a table, the FONTWIDTH= attribute affects only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the font for the text that appears in the table, set the attribute for a table cell.

Example "Example 1: Creating a Stand-Alone Style" on page 466

FRAME=*frame-type*

specifies the type of frame to use on a table. This table shows the possible values for *frame-type* and their meanings:

Table 21.26 *Frame-type Values*

Value for *frame-type*	Frame Type
ABOVE	A border at the top
BELOW	A border at the bottom
BOX	Borders at the top, bottom, and both sides
HSIDES	Borders at the top and bottom
LHS	A border at the left side
RHS	A border at the right side
VOID	No borders
VSIDES	Borders at the left and right sides

Valid in Markup family, PowerPoint, RTF, PDF , and Word destinations

Example "Example 3: Modifying the Default Style with the CLASS Statement" on page 482

FRAMEBORDER=ON | OFF
specifies whether to put a border around the frame for an HTML file that uses frames.

ON
places a border around the frame for an HTML file that uses frames.

OFF
specifies not to put a border around the frame for an HTML file that uses frames.

Valid in Markup family destinations

GRADIENT_DIRECTION="YAXIS"" | XAXIS
specifies the direction of the gradient.

"YAXIS"
specifies a vertical gradient.

"XAXIS"
specifies a horizontal gradient.

HEIGHT=*dimension*
specifies the height of a table cell, graph, or graphics in an HTML document.

dimension
is a nonnegative number.

See dimension attribute value on page 877

Valid in Markup family, RTF, and PDF destinations

Aliases CELLHEIGHT=

OUTPUTHEIGHT=

Restriction The HEIGHT= option does not apply to output generated as a result of GRSEG (graph segment) output.

Interaction The YPIXELS= option in the SAS/GRAPH GOPTIONS statement overrides the HEIGHT= attribute.

Tip HTML automatically sets cell height appropriately. You will seldom need to specify this attribute in the HTML destination.

HREFTARGET="*target*"
specifies the window or frame in which to open the target of the link. *target* is one of these values:

_blank
opens the target in a new, blank window. The window has no name.

Restriction Use lowercase letters to specify values for HREFTARGET.

_parent
opens the target in the window from which the current window was opened.

Restriction Use lowercase letters to specify values for HREFTARGET.

_search
opens the target in the browser's search pane.

Restriction Use lowercase letters to specify values for HREFTARGET.

_self
opens the target in the current window.

Restriction Use lowercase letters to specify values for HREFTARGET.

_top
opens the target in the topmost window.

Restriction Use lowercase letters to specify values for HREFTARGET.

"*name*"
opens the target in the specified window or the frame.

Valid in Markup family destinations

Default _self

Restriction Use lowercase letters to specify values for HREFTARGET.

Requirement *target* must be enclosed in quotation marks.

HTMLID="*string*"
specifies an ID for the table or table cell. The ID is for use by Java Script.

string
is the ID text.

Requirement *string* must be enclosed in quotation marks.

See string attribute value on page 879

Valid in Markup family destinations

HTMLSTYLE="*string*"
specifies individual attributes and values for a table or table cell in an HTML document.

string
is the name of an attribute or value.

Requirement *string* must be enclosed in quotation marks.

See string attribute value on page 879

Valid in Markup family destinations

IMAGE="*external-file*"
IMAGE="*external-file*?DESC=*alternative-text*"
specifies the image to appear in a graph. This image is positioned or tiled.

external-file
names a GIF or JPEG file. Use a simple filename, a complete path, or a URL.

Requirement *external-file* must be enclosed in quotation marks.

?DESC=*alternative-text*
specifies alternative image text for the image. To change the alternative text for an image, append `?desc=mydesc` to the image filename. This option helps to make reports accessible for people with a wide range of abilities. For accessibility, the text should convey the meaning of the image.

If an empty text string is specified, then the image is ignored by screen readers.

Valid in PDF, EPUB, and HTML5 destinations

Alias DESC=

Accessibility note You can use the ACCESSIBLECHECK system option to write a note to the SAS log if no description is provided for an image.

Example
```
image="questions.gif?desc=questionmark montage"
```

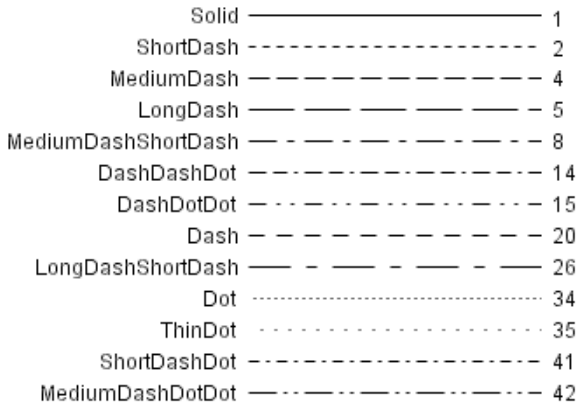
Valid in Markup family, PDF, and RTF destinations

Interaction The BACK= and IMAGESTYLE=TILE options in the SAS/GRAPH GOPTIONS statement override the IMAGE= attribute.

KPISKIN=BASIC | MODERN | NONE | ONYX | SATIN
specifies the type of skin to apply to KPI charts to give them a raised, 3-D appearance.

LINESTYLE=*pattern-number*
specifies the pattern of a line. Valid pattern numbers range from 1 to 46. Not all pattern numbers have names. You must specify the line pattern by its number. *pattern-number* can be one of the following:

Figure 21.1 *Table of Line Patterns*



LINETHICKNESS=*dimension*
specifies the thickness of a line.

See dimension attribute value on page 877

LINKCOLOR=*color*
specifies the color for the links in an HTML document that have not yet been visited.

Valid in Markup family, PDF, and RTF destinations

See color style attribute value on page 874

LISTENTRYANCHOR=ON | OFF
in an HTML document, the LISTENTRYANCHOR= attribute specifies whether to make the entry in the table of contents a link to the body file.

ON
specifies to make this entry in the table of contents a link to the body file.

OFF
specifies not to make this entry in the table of contents a link to the body file.

Valid in Markup family destinations

LISTSTYLEIMAGE=*string*
specifies a path or URL to an image file to be used as the bullet for list items.

Valid in PDF destinations

LISTSTYLETYPE=*bullet-type*
specifies the type of bullet to use for lists and the contents file. ODS uses bullets in the contents file.

bullet-type
The *bullet-type* value differs for PowerPoint, Markup, Excel, and Word destinations.

The following values are valid in the Markup destinations:

ARMENIAN	LOWER-GREEK
CIRCLE	LOWER-LATIN
DECIMAL	LOWER-ROMAN
DECIMAL-LEADING-ZERO	NONE
DISC	UPPER-ALPHA
GEORGIAN	UPPER-LATIN
INHERIT	UPPER-ROMAN
LOWER-ALPHA	SQUARE

The following values are valid in the PowerPoint destination:

ASTERISKS	HYPHEN
BOX	LOWER_ROMAN
CHECK	PARENTHESISED_LOWER_LATIN
CIRCLE	PARENTHESISED_DECIMAL
DECIMAL	SQUARE
DIAMOND	UPPER_ROMAN
DISC	

The following values are valid in the Excel destination:

CIRCLE	SQUARE
DISC	NONE

The following values are valid in the Word destination:

CIRCLE	LOWER_ROMAN
DECIMAL	NONE

DECIMAL_LEADING_ZERO	SQUARE
DISC	UPPER_ALPHA
LOWER_ALPHA	UPPER_ROMAN

Valid in PowerPoint, Markup family, and Excel destinations

Default DISC

See string attribute value on page 879

MARGINBOTTOM=*dimension*
specifies the bottom margin for the document. It is the vertical space after a table, or at the bottom of a cell.

Valid in Markup family, PDF, RTF, and Word destinations

Alias BOTTOMMARGIN=

Tip If the orientation of a PDF document is changed after the PDF destination is opened and before the PDF destination is closed, any setting for margins is taken from the OPTIONS statement in place before the ODS PDF FILE= statement. If no OPTIONS statement is used to explicitly set the margins, the margin settings are retrieved from the SAS registry, using PROC REGISTRY. For more info, see "REGISTRY Procedure" in *Base SAS Procedures Guide*.

See dimension attribute value on page 877

MARGINLEFT=*dimension*
specifies the left margin for the document.

Valid in Markup family, PowerPoint, PDF, RTF, and Word destinations

Alias LEFTMARGIN=

Interaction Specifying a left margin overrides any margin specified with the MARGIN= option.

Tip If the orientation of a PDF document is changed after the PDF destination is opened and before the PDF destination is closed, any setting for margins is taken from the OPTIONS statement in place before the ODS PDF FILE= statement. If no OPTIONS statement is used to explicitly set the margins, the margin settings are retrieved from the SAS registry, using PROC REGISTRY. For more info, see "REGISTRY Procedure" in *Base SAS Procedures Guide*.

See dimension attribute value on page 877

MARGINRIGHT=*dimension*
specifies the right margin for the document.

Valid in Markup family, PowerPoint, PDF, RTF, and Word destinations

Alias RIGHTMARGIN=

Interaction Specifying a right margin overrides any margin specified with the MARGIN= option.

Tip If the orientation of a PDF document is changed after the PDF destination is opened and before the PDF destination is closed, any setting for margins is taken from the OPTIONS statement in place before the ODS PDF FILE= statement. If no OPTIONS statement is used to explicitly set the margins, the margin settings are retrieved from the SAS registry, using PROC REGISTRY. For more info, see "REGISTRY Procedure" in *Base SAS Procedures Guide*.

See dimension attribute value on page 877

MARGINTOP=*dimension*
specifies the top margin for the document. It is the vertical space before the table, or at the top of a cell..

Valid in Markup family, PDF, RTF, and Word destinations

Alias TOPMARGIN=

Tip If the orientation of a PDF document is changed after the PDF destination is opened and before the PDF destination is closed, any setting for margins is taken from the OPTIONS statement in place before the ODS PDF FILE= statement. If no OPTIONS statement is used to explicitly set the margins, the margin settings are retrieved from the SAS registry, using PROC REGISTRY. For more info, see "REGISTRY Procedure" in *Base SAS Procedures Guide*.

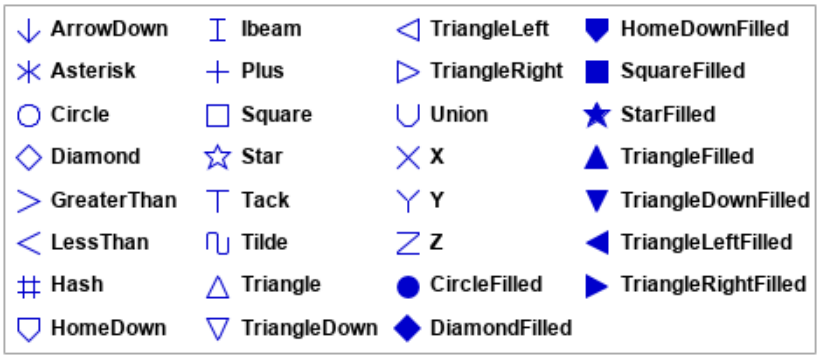
See dimension attribute value on page 877

MARKERSIZE=*dimension*
specifies the marker size (both width and height).

See dimension attribute value on page 877

MARKERSYMBOL=*marker-symbol*
specifies a marker symbol. *marker-symbol* can be one of the following:

Figure 21.2 *Table of Marker Symbols*



NEUTRALCOLOR=*color*
specifies the middle color in a three-color ramp.

See color style attribute value on page 874

NOBREAKSPACE=ON | OFF
specifies how to handle space characters in table cells.

ON
does not let SAS break a line at a space character.

OFF
lets SAS break a line at a space character if appropriate.

Valid in Markup family, Excel, PDF, and RTF destinations

ORPHANS=*integer*
specifies the minimum number of lines of text that must appear in a paragraph before it is forced to move to another page.

Valid in EPUB destination

Default 2

OUTLINECOLOR=*color*
specifies the color of the link-focus indicator outline.

Valid in HTML5 destination

Tip The OUTLINECOLOR= attribute is useful for making your HTML5 documents accessible. See Enhancing the Appearance of the Link-Focus Indicator in *Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics*.

See color style attribute value on page 874

OUTLINESTYLE=*line-style*
specifies the line style of the link-focus indicator outline.

line-style
can be one of the following:

- DASHED
- DOTTED
- DOUBLE
- GROOVE
- HIDDEN
- INSET
- OUTSET
- RIDGE
- SOLID

Valid in HTML5 destination

Tip The OUTLINESTYLE= attribute is useful for making your HTML5 documents accessible. See Enhancing the Appearance of the Link-Focus Indicator in *Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics*.

OUTLINEWIDTH=*dimension*
specifies the line width of the link-focus indicator outline.

Valid in HTML5 destination

Tip The OUTLINEWIDTH= attribute is useful for making your HTML5 documents accessible. See Enhancing the Appearance of the Link-Focus Indicator in *Creating Accessible SAS Viya Platform Output Using ODS and ODS Graphics*.

See dimension attribute value on page 877

OUTPUTHEIGHT=*dimension*
specifies the height of a graph.

See dimension attribute value on page 877

OUTPUTWIDTH=*dimension*
specifies the width of a graph.

See dimension attribute value on page 877

PADDING=*dimension* | *dimension%*
specifies the amount of white space between the content of the table cell and the border. The value of PADDING= applies to all four sides.

To change the padding of each side, use one or more of the following attributes:

- PADDINGBOTTOM= on page 865
- PADDINGLEFT= on page 865
- PADDINGRIGHT= on page 865
- PADDINGTOP= on page 866

Valid in Markup family, RTF, and PDF destinations

See dimension attribute value on page 877

PADDINGBOTTOM=*dimension* | *dimension*%
specifies the amount of white space on the bottom of the content of the table cell.

Valid in Markup family, PowerPoint, RTF, and PDF destinations

Default 0

See dimension attribute value on page 877

PADDINGLEFT=*dimension* | *dimension*%
specifies the amount of white space on the left side of the content of the table cell.

Valid in Markup family, PowerPoint, RTF, and PDF destinations

Default 0

See dimension attribute value on page 877

PADDINGRIGHT=*dimension* | *dimension*%
specifies the amount of white space on the right side of the content of the table cell.

Valid in Markup family, PowerPoint, RTF, and PDF destinations

Default 0

See dimension attribute value on page 877

PADDINGTOP=*dimension* | *dimension*%
specifies the amount of white space on the top of the content of the table cell.

Valid in Markup family, PowerPoint, RTF, and PDF destinations

Default 0

See dimension attribute value on page 877

PAGEBREAKHTML="*string*"
specifies HTML to place at page breaks in an HTML document.

string
is the HTML code used to place at page breaks.

Requirement *string* must be enclosed in quotation marks.

See string attribute value on page 879

Valid in Markup family destinations

POSTHTML="*string*"
specifies the HTML code to place after the table or table cell.

string
is the HTML code to place after a table or table cell.

Requirement *string* must be enclosed in quotation marks.

See string attribute value on page 879

Valid in Markup family destinations

Example "Example 3: Modifying the Default Style with the CLASS Statement" on page 482

POSTIMAGE="*external-file*" | *fileref* | POSTIMAGE="*external-file*? DESC=*alternative-text*"
specifies an image to place after the table or table cell.

external-file
names a GIF or JPEG file. Use a simple filename, a complete path, or a URL.

Requirements *external-file* must be enclosed in quotation marks.

For the ODS destination for Word, the image must be specified with a filename or a complete path and not with a URL.

fileref
is a reference that has been assigned to an external file. Use the FILENAME statement to assign a fileref.

See "Statements" in *SAS DATA Step Statements: Reference* for information about the FILENAME statement.

?DESC=*alternative-text*
specifies alternative image text for the image. To change the alternative text for an image, append `?desc=mydesc` to the image filename. This option helps to make reports accessible for people with a wide range of abilities. For accessibility, the text should convey the meaning of the image.

If an empty text string is specified, then the image is ignored by screen readers.

Valid in PDF, EPUB, and HTML5 destinations

Alias DESC=

Accessibility note You can use the ACCESSIBLECHECK system option to write a note to the SAS log if no description is provided for an image.

Example
```
postimage="questions.gif?desc=questionmark montage"
```

Valid in Markup family, PDF, POWERPOINT, RTF, and WORD destinations

Restrictions The ODS EXCEL destination does not support the POSTIMAGE= option.

For the ODS destination for PowerPoint, the POSTIMAGE= attribute can be used to place an image below a table, but it cannot be used to put an image after a cell.

Tip The ODS destination for Excel can display an image when using the TEMPLATE procedure and when specifying the BACKGROUNDIMAGE= option on the DEFINE STYLE statement. You can also use the Report Writing Interface (RWI) IMAGE method. For more information see Overview: TEMPLATE Procedure on page 434.

POSTTEXT="*string*"
specifies text to place after the table cell or table.

Requirement *string* must be enclosed in quotation marks.

See string attribute value on page 879

PREHTML="*string*"
specifies the HTML code to place before the table or table cell.

Restriction The PREHTML= attribute is valid only for markup family destinations.

See string attribute value on page 879

PREIMAGE="*external-file*" | *fileref* | PREIMAGE="*external-file*?DESC=*alternative-text*"
specifies an image to place before the table or table cell.

external-file
names a GIF or JPEG file. Use a simple filename, a complete path, or a URL.

Requirements Enclose *external-file* in quotation marks.

For the ODS destination for Word, the image must be specified with a filename or a complete path and not with a URL.

fileref
is a reference that has been assigned to an external file. Use the FILENAME statement to assign a fileref. (For information about the FILENAME statement, see "Statements" in *SAS DATA Step Statements: Reference*.)

?DESC=*alternative-text*
specifies alternative image text for the image. To change the alternative text for an image, append `?desc=mydesc` to the image filename. This option helps to make reports accessible for people with a wide range of abilities. For accessibility, the text should convey the meaning of the image.

If an empty text string is specified, then the image is ignored by screen readers.

Valid in PDF, EPUB, and HTML5 destinations

Alias DESC=

Accessibility note You can use the ACCESSIBLECHECK system option to write a note to the SAS log if no description is provided for an image.

Example
```
preimage="questions.gif?desc=questionmark montage"
```

Valid in Markup family, PDF, POWERPOINT, RTF, and WORD destinations

Restrictions For the ODS destination for PowerPoint, the PREIMAGE= attribute can be used to place an image above a table, but it cannot be used to put an image before a cell.

When using the PREIMAGE= style attribute with the PDF destination, you must specify STARTPAGE=NO on the PDF statement to display page numbers, times, dates, and titles. Without the STARTPAGE=NO option, preimages are treated like graphs and have no page numbers, times, dates, or titles displayed.

The ODS EXCEL destination does not support the PREIMAGE= option.

Tip The ODS destination for Excel can display an image when using the TEMPLATE procedure and when specifying the BACKGROUNDIMAGE= option on the DEFINE STYLE statement. You can also use the Report Writing Interface (RWI) IMAGE method. For more information see Overview: TEMPLATE Procedure on page 434.

PRETEXT="*string*"
specifies text to place before the table cell or table.

string
text that is placed before the table cell or table.

Valid in Markup family, Excel, PDF, RTF, and Word destinations

Requirement Enclose *string* in quotation marks.

See string attribute value on page 879

Example "Customizing the Table of Contents" in *SAS Output Delivery System: User's Guide*

PROTECTSPECIALCHARS=ON | OFF | AUTO
specifies how less-than signs (<), greater-than signs (>), and ampersands (&) are interpreted in table cells. In HTML and other markup languages, these characters indicate the beginning of a markup tag, the end of a markup tag, and the beginning of the name of a file or character entity.

ON
interprets special characters as the characters themselves. That is, when ON is in effect the characters are protected before they are passed to the HTML or other markup language destination so that the characters are not interpreted as part of the markup language. Using ON enables you to show markup language tags in the HTML document.

OFF
interprets special characters as markup language tags. That is, when OFF is in effect, the characters are passed to the HTML or other markup language destination without any protection so that the special characters are interpreted as part of the markup language.

AUTO
interprets any string that starts with a < and ends with a > as a markup language tag (ignoring spaces that immediately precede the <, spaces that immediately follow the >, and spaces at the beginning and end of the string). In any other string, AUTO protects the special characters from their markup language meaning.

Valid in Markup family, Excel, PDF, and RTF destinations

RULES=*rule-type*
specifies the types of rules to use in tables. This table shows the possible values for the RULES= attribute and their meanings:

Table 21.27 *RULES= Attribute Values*

Value of RULES= Attribute	Locations of Rules
ALL	Between all rows and columns
COLS	Between all columns
GROUPS	Between the table header and the table and between the table and the table footer, if there is one
NONE	No rules anywhere
ROWS	Between all rows

Valid in Markup family, PowerPoint, PDF, RTF, and Word destinations

STARTCOLOR=*color*
specifies the start fill color for a graph. It is used to create a gradient effect.

Note: You can have either a start and end gradient effect or no gradient effect. If you specify a TRANSPARENCY level and you only specify the STARTCOLOR, then the end color is completely transparent gradationally to the specified start color.

Valid in HTML4 destination

See color style attribute value on page 874

TAGATTR="*string*"
specifies text to insert into HTML.

string
is the text that is inserted into HTML tags.

Requirements *string* must be enclosed in quotation marks.

string must be valid HTML for the context in which the style element is created.

Interaction When using the Report Writing Interface with the ODS destination for Excel, the TAGATTR= Style Attribute is needed to pass Excel formats instead of the FORMAT parameter within the FORMAT_CELL Method .

Tip Many style elements are created between <TD> and </TD> tags. To determine how a style element is created, look at the source for the output.

See string attribute value on page 879

Valid in Markup family and Excel destinations

TEXTALIGN=*alignment*
specifies justification in tables, table cells, and graphs. In graphs, this option specifies the justification of the image specified with the IMAGE= statement. For example, this statement would produce a page number that is centered at the bottom of the page: `style PageNo from TitleAndFooters / textalign=c verticalalign=b;` This statement would produce a date in the body file that is left-justified at the top of the page: `style BodyDate from Date / textalign=l;` The value of *alignment* can be one of the following:

CENTER
specifies center justification.

Alias C

DEC
specifies aligning the values by the decimal point.

Alias D

Restriction Decimal alignment is supported for the PDF, RTF, and Word destinations only.

JUST
specifies a justified alignment of the cells in a document

Alias J

Restriction A justified alignment is supported for the Word destination only.

LEFT
specifies left justification.

Alias L

RIGHT
specifies right justification.

Alias R

Restriction Not all contexts support RIGHT. If RIGHT is not supported, it is interpreted as CENTER.

Valid in Markup family, PDF, PowerPoint, Excel, Word, and RTF destinations

Alias JUST=

Restriction For the HTML5 destination, you might be able to use MARGINRIGHT=0 instead.

Tips For the PDF destinations and the MARKUP destination, use the style attribute TEXTALIGN= with the style attribute VERTICALALIGN= in the style element PageNo to control the placement of page numbers.

For PDF destinations and the MARKUP destination, control the placement of dates by using the style attribute TEXTALIGN= with the style attribute VERTICALALIGN= in the BodyDate or Date. style element.

TEXTDECORATION=*presentation–options*
changes the visual presentation of the text. *presentation–options* can be one of the following:

BLINK
specifies that the text's visual presentation alternates rapidly between visible and invisible.

Valid in Excel, HTML, RTF, PowerPoint, and Excel destinations

LINE_THROUGH
specifies that a line is drawn through the text.

Valid in HTML, PDF, measured RTF, and RTF destinations

OVERLINE
specifies that a line is drawn above the text.

Valid in HTML and PDF destinaions

UNDERLINE
specifies that a line is drawn below the text.

Valid in HTML, PDF, measured RTF, and RTF destinations

Tip TEXTDECORATION= can be used with inline formatting and the ODS PDF statement to enhance PDF files.

Example "Formatting Cells with the Textdecoration Style Attribute " in *SAS Output Delivery System: Advanced Topics*

TEXTINDENT=*n*
specifies the number of spaces that the first line of output is indented.

n
specifies the number of spaces to indent the output.

Valid in Markup family, PDF, Excel, PowerPoint, RTF, and Word destinations

Alias INDENT=

Default The default value for XML is 2. For all other ODS destinations, the default value is 0. Tables in Microsoft Word documents are center-justified.

TICKDISPLAY="INSIDE"" | OUTSIDE"" | ACROSS"
specifies the placement of all major and minor axis tick marks.

TEXTJUSTIFY=INTER_WORD | INTER_CHARACTER
specifies how to evenly distribute text.

INTER_WORD
specifies that the words are evenly distributed across the page.

INTER_CHARACTER
specifies that all characters are evenly distributed across a page.

Valid in RTF, TAGSETS.RTF, and Word destinations

Restriction The ODS destination for Word supports only the INTER_WORD option.

Interaction The ODS destination for Word accepts the INTER_WORD option when the TEXTALIGN=J (alias JUST=) style attribute is set.

Tip Use the TEXTJUSTIFY= style attribute with the TEXTALIGN=J (alias JUST=) style attribute.

TRANSPARENCY=*dimension*
specifies a transparency level for graphs. The values are 0.0 (opaque) to 1.0 (transparent).

Valid in HTML destination

See dimension attribute value on page 877

URL="*uniform-resource-locator*"
specifies a URL to link to from various reporting elements, including table cells.

Valid in Markup family, Excel, PDF, and RTF destinations

Requirement *uniform-resource-locator* must be enclosed in quotation marks.

VERTICALALIGN=BOTTOM | MIDDLE | TOP
specifies vertical justification for graphs and cells. In graphs, this option specifies the vertical justification of the image specified with IMAGE=. For example, this statement produces a page number that is centered at the bottom of the page: `style PageNo from TitleAndFooters / textalign=c verticalalign=b;` This statement produces a date in the body file that is left-justified at the top of the page: `style BodyDate from Date / textalign=l verticalalign=t;`

BOTTOM
specifies bottom justification.

Alias B

MIDDLE
specifies center justification.

Alias M

TOP
specifies top justification.

Alias T

Valid in Markup family, Excel, PDF, RTF, and Word destinations

Alias VJUST=

Tips For PDF and markup family destinations, use the style attribute VERTICALALIGN= with the style attribute TEXTALIGN= in the style element PAGENO to control the placement of page numbers.

For PDF and markup family destinations, control the placement of dates by using the style attribute VERTICALALIGN= with the style attribute TEXTALIGN= in the BODYDATE or DATE style element.

VISITEDLINKCOLOR=*color*
specifies the color for links that have been visited in an HTML document.

Valid in Markup family destinations

See color style attribute value on page 874

WATERMARK=ON | OFF
specifies whether to make the image that is specified by BACKGROUNDIMAGE= into a watermark. A watermark appears in a fixed position as the window is scrolled.

ON
specifies to make the image that is specified by BACKGROUNDIMAGE= into a watermark.

OFF
specifies not to make the image that is specified by BACKGROUNDIMAGE= into a watermark.

Valid in Markup family and RTF destinations

Tip You can apply a watermark to output generated using the ODS TAGSETS.RTF destination by specifying a background image file. Note that the image is applied to the RTF document and not to a table or a table cell.

See "BACKGROUNDIMAGE="*string*"|BACKGROUNDIMAGE=*'gradient'* | WATERMARK-IMAGE <?WASHOUT=0 | 1>" on page 839

WIDTH=*dimension*
specifies the width of a table cell, table, line, or a graph.

When used with graphs, the WIDTH= option must be specified as a pixel or percentage value. If a unit of measure is not specified with the *dimension*, then the value will be in pixels. If a unit of measure other than pixels or percentage is specified with the *dimension*, then the HEIGHT=*dimension* is not applied to the graph.

dimension
is a nonnegative number.

See dimension attribute value on page 877

Valid in Markup family, Excel, PDF, and RTF destinations

Aliases CELLWIDTH=

OUTPUTWIDTH=

Restriction The WIDTH= option does not apply to output generated as a result of GRSEG (graph segment) output.

Interaction The XPIXELS= option in the SAS/GRAPH GOPTIONS statement overrides the WIDTH= attribute.

Tips A column of cells has the width of the widest cell in the column.

Use WIDTH=100% to make the table or graph as wide as the window that it is open in.

WINDOWS=*integer*
specifies the number of lines of text that must appear at the top of a page if a paragraph is separated by a page break.

Valid in EPUB destination

Default 2

Style Attributes Values

color
is a string that identifies a color. A color is defined in the following ways:

- most of the color names that are supported by SAS/GRAPH. These names include the following:
  - a predefined SAS color (for example, blue or VIYG)
  - a red/green/blue (RGB) value (for example, CX0023FF)
  - a hue/light/saturation (HLS) value (for example, H14E162D)
  - a gray-scale value (for example, GRAYBB).
  - a red/green/blue transparency (RGBA) value (for example, a98FB9880)
  - a cyan/magenta/yellow/black (CMYK) value (for example, FFFFFF00)

  Note: RGBA color mode is not supported by Java devices. RGBA color mode is supported by ActiveX devices when the output is used in Microsoft applications.

- an RGB value with a leading number sign (#) rather than CX (for example, #0023FF).
- one of the colors that exist in the SAS session when the style is used:
  - DMSBLUE
  - DMSRED
  - DMSPINK
  - DMSGREEN
  - DMSCYAN
  - DMSYELLOW
  - DMSWHITE
  - DMSORANGE
  - DMSBLACK
  - DMSMAGENTA
  - DMSGRAY
  - DMSBROWN
  - SYSBACK
  - SYSSECB
  - SYSFORE

  Note: Use these colors only when running SAS in the windowing environment.

- an English description of an HLS. Such descriptions use a combination of words to describe the lightness, the saturation, and the hue (in that order). Use the Color Naming System to form a color in the following ways:
  - combining a chromatic hue with a lightness, a saturation, or both
  - combining the achromatic hue gray with a lightness
  - combining the achromatic hue black or white without qualifiers

  Use the words in the following table:

Table 21.28 *Hue/Light/Saturation (HLS) Values*

Lightness	Saturation	Chromatic Hue	Achromatic Hue
		Blue	Black[1]
Very dark	Grayish	Purple	
Dark	Moderate	Red	
Medium	Strong	Orange | brown	Gray[2]
Light	Vivid	Yellow	
Very light		Green	
			White[1]

1 Black and white cannot be combined with a lightness value or a saturation value.
2 Gray cannot be combined with a saturation value.

Combine these words to form a wide variety of colors. Here are examples:

- light vivid green
- dark vivid orange
- light yellow

Note: The Output Delivery System first tries to match a color with a SAS/GRAPH color. Thus, although brown and orange are interchangeable in the table, if you use them as unmodified hues, then they are different. The reason for this is that ODS interprets them as SAS colors, which are mapped to different colors.

You can also specify hues that are intermediate between two neighboring colors. To do so, combine one of these adjectives with one of its neighboring colors:

- reddish
- orangish
- brownish
- yellowish
- greenish
- bluish
- purplish
- bluish purple
- reddish orange
- yellowish green

Tips For a list of some valid colors, see Link to Valid Colors to use with cascading style sheets.

To see how color names map to hexadecimal values, submit the following REGISTRY procedure code:

```
proc registry list startat="COLORNAMES";
run;
```

See RBG Color Codes, HLS Color Codes, and Gray-Scale Color codes in *SAS/GRAPH: Reference* for information about SAS/GRAPH colors.

dimension

is a whole number, a percentage, or a nonnegative number followed by one of these units of measure:

Table 21.29 *Units of Measure for Dimension*

cm	Centimeters
em	Standard typesetting measurement unit for width
ex	Standard typesetting measurement unit for height
in	Inches
mm	Millimeters
pt	A printer's point

Default For the PRINTER destination, units of 1/150 of an inch

font-definition

is the name of a font, the font size, and font keywords. A font definition has this general format.

("*font-face-1* <… , *font-face-n*>", *font-size*, *keyword-list*)

A font specification in an ODS style can specify multiple fonts so that a reasonable substitution can be made when a font cannot be located on the current computer. The fonts are normally listed in a most-specific to most-generic order. For example, the TitleFont attribute shown in the following example specifies a list of fonts in the following order: <sans-serif>, <MTsans-serif>, Helvetica, and then Helv.

```
'TitleFont'=("<sans-serif>,<MTsans-serif>,Helvetica,Helv",3,bold)
```

For table title text, SAS searches the computer for a font in the TitleFont font list in the order listed. The first font in the list that is found is used.

If a font is enclosed in angle brackets, that indicates that the resolved font is defined in the ODS\FONTS portion of the SAS registry. One way to view the SAS registry font definitions is with the Registry Procedure.

"*font-face*"

specifies the name of the font.

ODS styles can now use new TrueType fonts. All Universal Printers and many SAS/GRAPH devices use the FreeType library to render TrueType fonts for output in all of the operating environments that SAS software supports. In addition, by default, many SAS/GRAPH device drivers and all Universal Printers generate output using ODS styles, and these ODS styles use TrueType fonts. In addition to SAS Monospace and SAS Monospace Bold, 21 new TrueType fonts are made available when you install SAS:

- five Latin fonts compatible with Microsoft
- eight multilingual Unicode fonts
- eight monolingual Asian fonts

For more information about the TrueType fonts, see "TrueType Fonts Supplied by SAS" in *SAS Viya Platform Universal Printing*.

Restriction You must enclose multiple *font-face* in quotation marks. If you specify only one font and if its name does not include a space character, then omit the quotation marks.

Tip If you specify more than one font, then the destination device uses the first one that is installed on the system.

font-size
specifies the size of the font. *font-size* is a dimension or a number without units of measure. If you specify a dimension, then specify a unit of measure. Without a unit of measure the number becomes a size that is relative to all other font sizes in the HTML document. For more information, see dimension attribute value on page 877.

keyword-list
specifies the font weight, font style, and font width. Include one value for each, in any order. This table shows the keywords to use:

Table 21.30 *Font Keywords*

Keywords for Font Weight	Keywords for Font Style	Keywords for Font Width
MEDIUM	ITALIC	NORMAL[1]
BOLD	ROMAN	COMPRESSED[1]
DEMI_BOLD[1]	SLANT	EXTRA_COMPRESSED[1]
EXTRA_BOLD[1]	OBLIQUE[1]	NARROW[1]
LIGHT	NORMAL[1]	WIDE[1]
DEMI_LIGHT[1]		EXPANDED[1]
EXTRA_LIGHT[1]		
NORMAL[1]		

1 Few fonts honor these values.

Example “Example 2: Using User-Defined Attributes” on page 473

format
is a SAS format or a user-defined format.

integer | integer-list | integer-column-list
specifies a column variable that contains integer values, or a dynamic variable that refers to such a column variable.

integer
specifies a single integer.

integer-list
specifies a sequence of integer values, or a column variable that contains integer values, or a dynamic variable that refers to such a column variable or to a string.

integer-column-list
specifies a sequence of column variables, or a column variable that contains column variables, or a dynamic variable that refers to such a column variable, or a dynamic variable that refers to a string containing a list of column variables. Values within the columns must be integers.

style-reference
is a reference to an attribute that is defined in the current style or in the parent style (or beyond). The value used is the name of the style element followed by the name of an attribute, in parentheses, within that element. Style references have the following form.

style-attribute=target-style-element("target-style-attribute")

style-attribute
specifies the name of the style attribute.

target-style-element
specifies the name of the style element that contains the style attribute that you want to reference.

target-style-attribute
specifies the style attribute with the value that you want to use.

Requirement You must enclose *target-style-attribute* in quotation marks if it is a user-supplied style attribute.

See “Understanding Style References ” on page 448

Example “Example 2: Using User-Defined Attributes” on page 473

"*string*"
is a quoted character string.

user-defined-format
specifies a format created with the FORMAT procedure.

Restriction *user-defined-format* can be specified only for data cells.

PART 9

Appendices

Appendix 1
Output Object Table Names **883**

Appendix 2
Example Code **909**

Appendix 3
Lua License **911**

Appendix 1

Output Object Table Names

ODS Table Names Produced by Base SAS Procedures 883
ODS Table Names Produced by Base SAS High-Performance Procedures 897
ODS Table Names Produced by Base SAS Statistical Procedures 897
ODS Table Names Produced by SAS/STAT Procedures 898
ODS Table Names Produced by SAS/STAT High-Performance Procedures 902
ODS Table Names Produced by SAS Enterprise Miner High-Performance Procedures 903
ODS Table Names Produced by SAS/ETS Procedures 904
ODS Table Names Produced by SAS/ETS High-Performance Procedures 906
ODS Table Names Produced by SAS/QC Procedures 906
ODS Table Names Produced by SAS/OR Procedures 907
ODS Table Names Produced by Graph Algorithms and Network Analysis Procedure 908

ODS Table Names Produced by Base SAS Procedures

This table lists the output object table names that Base SAS procedures produce. The table provides the name of each table, a description of what the table contains, and the option, if any, that creates the output object table.

Table A13.1 *ODS Table Names Produced by the CATALOG Procedure*

Table Name	Description
Catalog_Random	Table generated when the catalog is in a random-access data library
Catalog_Sequential	Table generated when the catalog is in a sequential-access data library

Table A13.2 *ODS Table Names Produced by the COMPARE Procedure*

Table Name	Description	Option
CompareDatasets	Information about the data set or data sets	Omit NOSUMMARY or NOVALUE options
CompareDetails (Comparison results for observations)	List of observations that the base data set and the compare data set do not have in common	PRINTALL
CompareDifferences	Report of variable value differences	Omit NOVALUES option
CompareSummary	Summary report of observations, values, and variables of unequal values	
CompareVariables	List of differences in variable types or attributes between the base data set and the compare data set	Omit NOSUMMARY option unless the variables are identical
ODS Tables Created by the ID Statement		
CompareDetails	List of notes and warnings concerning duplicate ID variable values if duplicate ID variable values exist in either of the data sets	

Table A13.3 ODS Table Names Produced by the CORR Procedure

Table Name	Description	Option
Cov	Covariances	COV
CronbachAlpha	Coefficient alpha	ALPHA
CronbachAlphaDel	Coefficient alpha with deleted variable	ALPHA
Csscp	Corrected sums of squares and crossproducts	CSSCP
FisherPearsonCorr	Pearson correlation statistics using Fisher's *z* transformation	FISHER
FisherSpearmanCorr	Spearman correlation statistics using Fisher's *z* transformation	FISHER SPEARMAN
HoeffdingCorr	Hoeffding's D statistics	HOEFFDING
KendallCorr	Kendall's tau-b coefficients	KENDALL
PearsonCorr	Pearson correlations	PEARSON
PolychoricCorr	Polychoric correlations	POLYCHORIC
PolyserialCorr	Polyserial correlations	POLYSERIAL
SimpleStats	Simple descriptive statistics	
SpearmanCorr	Spearman correlations	SPEARMAN
Sscp	Sums of squares and crossproducts	SSCP
VarInformation	Variable information	
ODS Tables Created by the PARTIAL Statement		
FisherPearsonPartialCorr	Pearson partial correlation statistics using Fisher's *z* transformation	FISHER

Table Name	Description	Option
FisherSpearmanPartialCorr	Spearman partial correlation statistics using Fisher's *z* transformation	FISHER SPEARMAN
PartialCsscp	Partial corrected sums of squares and crossproducts	CSSCP
PartialCov	Partial covariances	COV
PartialKendallCorr	Partial Kendall tau-b coefficients	KENDALL
PartialPearsonCorr	Partial Pearson correlations	
PartialSpearmanCorr	Partial Spearman correlations	SPEARMAN

Table A13.4 *ODS Table Names Produced by the DATASETS and CONTENTS Procedures*

Table Name	Description	Option
Directory	General library information	Omit NOLIST option
Members	Library member information	Omit NOLIST option

Table A13.5 *ODS Table Names Produced by the CONTENTS Procedure or the DATASETS Procedure with the CONTENTS Statement*

Table Name	Description	Option
Attributes	Data set attributes	Omit SHORT option
Directory	General library information	DATA=<*libref* .>_ALL_ or the DIRECTORY option[1]
EngineHost	Engine and operating environment information	Omit SHORT option

Table Name	Description	Option
IntegrityConstraints	List of integrity constraints	Omit SHORT option and data has integrity constraints
IntegrityConstraintsShort	Concise listing of integrity constraints	SHORT option specified and data has integrity constraints
Indexes	List of indexes	Omit SHORT option and data set is indexed
IndexesShort	Concise list of indexes	SHORT option specified and data set is indexed
Members	Library member information	DATA=<*libref* .>_ALL_ or the DIRECTORY option[1]
Position	List of variables by logical position in the data set	Omit SHORT option and specify the VARNUM option
PositionShort	Concise list of variables by logical position in the data set	SHORT and VARNUM options
Sortedby	Sort information	Omit SHORT option and data set is sorted
SortedbyShort	Concise sort information	SHORT option and data set is sorted
Variables	List of variables in alphabetical order	Omit SHORT option
VariablesShort	Concise listing of variables in alphabetical order	SHORT

1 ForPROC DATASETS, if both the NOLIST option and either the DIRECTORY option or DATA=<*libref*.>_ALL_ are specified, then the NOLIST option is ignored.

Table A13.6 *ODS Table Names Produced by the FREQ Procedure*

Table Name	Description	Statement	Option
BarnardsTest	Barnard's exact test	EXACT	BARNARD
BinomialCLs	Binomial confidence limits	TABLES	BINOMIAL(AC | J | W)

Table Name	Description	Statement	Option
BinomialEquiv	Binomial equivalence analysis	TABLES	BINOMIAL(EQUIV)
BinomialEquivLimits	Binomial equivalence limits	TABLES	BINOMIAL(EQUIV)
BinomialEquivTest	Binomial equivalence test	TABLES	BINOMIAL(EQUIV)
BinomialNoninf	Binomial noninferiority test	TABLES	BINOMIAL(NONINF)
Binomial	Binomial proportion	TABLES	BINOMIAL
BinomialTest	Binomial proportion test	TABLES	BINOMIAL
BinomialSup	Binomial superiority test	TABLES	BINOMIAL(SUP)
BreslowDayTest	Breslow-Day test	TABLES	CMH ($h \times 2 \times 2$ table)
CMH	Cochran-Mantel-Haenszel statistics	TABLES	CMH
ChiSq	Chi-square tests	TABLES	CHISQ
CochransQ	Cochran's	TABLES	AGREE ($h \times 2 \times 2$ table)
ColScores	Column scores	TABLES	SCOROUT
CommonOddsRatioCl	Exact confidence limits for the common odds ratio	EXACT	COMOR ($h \times 2 \times 2$ table)
CommonOddsRatioTest	Common odds ratio exact test	EXACT	COMOR ($h \times 2 \times 2$ table)
CommonPdiff	Common proportion difference	TABLES	RISKDIFF(COMMON) ($h \times 2 \times 2$ table)

Table Name	Description	Statement	Option
CommonRelRisks	Common relative risks	TABLES	CMH ($h \times 2 \times 2$ table)
CrossList	Crosstabulation table in column format	TABLES	CROSSLIST (n -way table, $n > 1$)
CrossTabFreqs	Crosstabulation table	TABLES	(n -way table, $n > 1$)
EqualKappaTest	Test for equal simple kappas	TABLES	AGREE ($h \times 2 \times 2$ table)
EqualKappaTests	Tests for equal kappas	TABLES	AGREE ($h \times r \times r$, $r > 2$)
EqualOddsRatios	Tests for equal odds ratios	EXACT	EQOR ($h \times 2 \times 2$ table)
GailSimon	Gail-Simon test	TABLES	GAILSIMON ($h \times 2 \times 2$ table)
FishersExact	Fisher's exact test	EXACT or TABLES or TABLES	FISHER FISHER or EXACT CHISQ (2×2 table)
FishersExactMC	Monte Carlo estimates for Fisher's exact test	EXACT	FISHER / MC
Gamma	Gamma	TEST	GAMMA
GammaTest	Gamma test	TEST	GAMMA
JTTest	Jonckheere-Terpstra test	TABLES	JT
JTTestMC	Monte Carlo estimates for Jonckheere-Terpstra exact test	EXACT	JT / MC
Kappa	Simple kappa coefficient	TEST or EXACT	KAPPA KAPPA

Table Name	Description	Statement	Option
KappaMC	Monte Carlo exact test for simple kappa coefficient	EXACT	KAPPA / MC
KappaStatistics	Kappa statistics	TABLES	AGREE, no TEST or EXACT ($r \times r$ table, > 2)
KappaTest	Simple kappa test	TEST or EXACT	KAPPA KAPPA
KappaWeights	Kappa weights	TABLES	AGREE(PRINTKWTS)
List	List format multiway table	TABLES	LIST
LRChiSq	Likelihood ratio chi-square exact test	EXACT	LRCHI
LRChiSqMC	Monte Carlo exact test for likelihood ratio chi-square	EXACT	LRCHI / MC
MantelFleiss	Mantel-Fleiss criterion	TABLES	CMH(MF) ($h \times 2 \times 2$ table)
McNemarsTest	McNemar's test	TABLES	AGREE (2×2 table)
Measures	Measures of association	TABLES	MEASURES
MHChiSq	Mantel-Haenszel chi-square exact test	EXACT	MHCHI
MHChiSqMC	Monte Carlo exact test for Mantel-Haenszel chi-square	EXACT	MHCHI / MC
NLevels	Number of variable levels	PROC	NLEVELS
OddsRatioCLs	Odds ratio confidence limits	TABLES	OR(CL=) (2×2 table)

Table Name	Description	Statement	Option
OddsRatioExactCL	Exact confidence limits for the odds ratio	EXACT	OR (2×2 table)
OneWayChiSq	One-way chi-square test	TABLES	CHISQ (one-way table)
OneWayChiSqMC	Monte Carlo exact test for one-way chi-square	EXACT	CHISQ / MC (one-way table)
OneWayFreqs	One-way frequencies	PROC or TABLES	(no TABLES stmt) (one-way table)
OneWayLRChiSq	One-way likelihood ratio chi-square test	TABLES	CHISQ(LRCHI) (one-way table)
OverallKappa	Overall simple kappa	TABLES	AGREE ($h \times 2 \times 2$ table)
OverallKappas	Overall kappa coefficients	TABLES	AGREE ($h \times r \times r$, > 2)
PdiffCLs	Proportion difference confidence limits	TABLES	RISKDIFF(CL=) (2×2 table)
PdiffEquiv	Equivalence analysis for the proportion difference	TABLES	RISKDIFF(EQUIV) (2×2 table)
PdiffEquivLimits	Equivalence limits for the proportion difference	TABLES	RISKDIFF(EQUIV) (2×2 table)
PdiffEquivTest	Equivalence test for the proportion difference	TABLES	RISKDIFF(EQUIV) (2×2 table)
PdiffNoninf	Noninferiority test for the proportion difference	TABLES	RISKDIFF(NONINF) (2×2 table)
PdiffSup	Superiority test for the proportion difference	TABLES	RISKDIFF(SUP) (2×2 table)

Table Name	Description	Statement	Option
PdiffTest	Proportion difference test	TABLES	RISKDIFF(EQUAL) (2×2 table)
PearsonChiSq	Pearson chi-square exact test	EXACT	PCHI
PearsonChiSqMC	Monte Carlo exact test for Pearson chi-square	EXACT	PCHI / MC
PearsonCorr	Pearson correlation	TEST or EXACT	PCORR PCORR
PearsonCorrMC	Monte Carlo exact test for Pearson correlation	EXACT	PCORR / MC
PearsonCorrTest	Pearson correlation test	TEST or EXACT	PCORR PCORR
PlCorr	Polychoric correlation	TEST	PLCORR
PlCorrTest	Polychoric correlation test	TEST	PLCORR
RelativeRisks	Relative risk estimates	TABLES	RELRISK or MEASURES (2×2 table)
RelRisk1ExactCL	Exact confidence limits for column 1 relative risk	EXACT	RELRISK (2×2 table)
RelRisk2ExactCL	Exact confidence limits for column 2 relative risk	EXACT	RELRISK (2×2 table)
RiskDiffCol1	Column 1 risk estimates	TABLES	RISKDIFF (2×2 table)
RiskDiffCol2	Column 2 risk estimates	TABLES	RISKDIFF (2×2 table)
RowScores	Row scores	TABLES	SCOROUT

Table Name	Description	Statement	Option
SomersDCR	Somers' *D(C|R)*	TEST or EXACT	SMDCR SMDCR
SomersDCRMC	Monte Carlo exact test for Somers' *D(C|R)*	EXACT	SMDCR / MC
SomersDCRTest	Somers' *D(C|R)* test	TEST or EXACT	SMDCR SMDCR
SomersDRC	Somers' *D(R|C)*	TEST or EXACT	SMDRC SMDRC
SomersDRCMC	Monte Carlo exact test for Somers' *D(R|C)*	EXACT	SMDRC / MC
SomersDRCTest	Somers' *D(R|C)* test	TEST or EXACT	SMDRC SMDRC
SpearmanCorr	Spearman correlation	TEST or EXACT	SCORR SCORR
SpearmanCorrMC	Monte Carlo exact test for Spearman correlation	EXACT	SCORR / MC
SpearmanCorrTest	Spearman correlation test	TEST or EXACT	SCORR SCORR
SymmetryTest	Test of symmetry	TABLES	AGREE
TauB	Kendall's tau-	TEST or EXACT	KENTB KENTB
TauBMC	Monte Carlo exact test for Kendall's tau-	EXACT	KENTB / MC
TauBTest	Kendall's tau- test	TEST or EXACT	KENTB KENTB
TauC	Stuart's tau-	TEST or EXACT	STUTC STUTC

Table Name	Description	Statement	Option
TauCMC	Monte Carlo exact test for Stuart's tau-	EXACT	STUTC / MC
TauCTest	Stuart's tau- test	TEST or EXACT	STUTC STUTC
TrendTest	Cochran-Armitage trend test	TABLES	TREND
TrendTestMC	Monte Carlo exact test for trend	EXACT	TREND / MC
WtKappa	Weighted kappa coefficient	TEST or EXACT	WTKAP WTKAP
WtKappaMC	Monte Carlo exact test for weighted kappa coefficient	EXACT	WTKAP / MC
WtKappaTest	Weighted kappa test	TEST or EXACT	WTKAP WTKAP

Table A13.7 *ODS Table Names Produced by the MEANS and SUMMARY Procedures*

Table Name	Description
Summary	Summary of descriptive statistics for variables across all observations and within groups of observations

Table A13.8 *ODS Table Names Produced by the REPORT Procedure*

Table Name	Description
Report	Detail report, summary report, or combination of both detail and summary information report

Table A13.9 ODS Table Names Produced by the SQL Procedure

For detailed information, see the *SAS SQL Procedure User's Guide* .	
Table Name	**Description**
SQL_Results	SAS view

Table A13.10 ODS Table Names Produced by the TABULATE Procedure

Table Name	**Description**
Table	Descriptive statistics in tabular format that use some or all of the variables in a data set

Table A13.11 ODS Table Names Produced by the UNIVARIATE Procedure

Table Name	**Description**	**Option**
ODS Tables Created by the PROC UNIVARIATE Statement		
BasicIntervals	Confidence intervals for mean, standard deviation, variance	CIBASIC
BasicMeasures	Measures of location and variability	Default
ExtremeObs	Extreme observations	Default
ExtremeValues	Extreme values	NEXTRAVAL=
Frequencies	Frequencies	FREQ
LocationCounts	Counts used for sign test and signed rank test	LOCCOUNT
Missing Values	Missing values	Default, if missing values exist
Modes	Modes	MODES

Table Name	Description	Option
Moments	Sample moments	Default
Plots	Line printer plots	PLOTS
Quantiles	Quantiles	Default
RobustScale	Robust measures of scale	ROBUSTSCALE
SSPlots	Line printer side-by-side box plot	PLOTS with BY statement
TestsForLocation	Tests for location	Default
TestsForNormality	Tests for normality	NORMALTEST
TrimmedMeans	Trimmed means	TRIMMED=
WinsorizedMeans	Winsorized means	WINSORIZED=
ODS Tables Created by the HISTOGRAM Statement		
Bins	Histogram bins	MIDPERCENTS secondary option
FitQuantiles	Quantiles of fitted distribution	Any distribution option
GoodnessOfFit	Goodness-of-fit tests for fitted distribution	Any distribution option
HistogramBins	Histogram bins	MIDPERCENTS option
ParameterEstimates	Parameter estimates for fitted distribution	Any distribution option

ODS Table Names Produced by Base SAS High-Performance Procedures

Table A13.12 ODS Table Names Produced by Base SAS High-Performance Procedures

For detailed information, see the procedures in *Base SAS Procedures Guide: High-Performance Procedures*	
Procedure	**ODS Table Names**
HPBIN	ODS Tables Produced by PROC HPBIN
HPCORR	ODS Tables Produced by PROC HPCORR
HPIMPUTE	ODS Tables Produced by PROC HPIMPUTE
HPSAMPLE	ODS Tables Produced by PROC HPSAMPLE

ODS Table Names Produced by Base SAS Statistical Procedures

Table A13.13 ODS Table Names Produced by Base SAS Statistical Procedures

For detailed information, see the procedures in *Base SAS Procedures Guide: Statistical Procedures*	
Procedure	**ODS Table Names**
CORR	ODS Tables Produced by PROC CORR
FREQ	ODS Tables Produced by PROC FREQ
UNIVARIATE	ODS Tables Produced by PROC UNIVARIATE

ODS Table Names Produced by SAS/STAT Procedures

Table A13.14 *ODS Table Names Produced by SAS/STAT Procedures*

For detailed information, see the procedures in *SAS/STAT User's Guide*	
Procedure	**ODS Table Names**
ACECLUS	ODS Tables Produced by PROC ACECLUS
ADAPTIVEREG	ODS Tables Produced by PROC ADAPTIVEREG
ANOVA	ODS Tables Produced by PROC ANOVA
BCHOICE	ODS Tables Produced by PROC BCHOICE
CALIS	ODS Tables Produced by PROC CALIS
CANCORR	ODS Tables Produced by PROC CANCORR
CANDISC	ODS Tables Produced by PROC CANDISC
CATMOD	ODS Tables Produced by PROC CATMOD
CAUSALGRAPH	ODS Tables Produced by PROC CAUSALGRAPH
CAUSALMED	ODS Tables Produced by PROC CAUSALMED
CAUSALTRT	ODS Tables Produced by PROC CAUSALTRT
CLUSTER	ODS Tables Produced by PROC CLUSTER
CORRESP	ODS Tables Produced by PROC CORRESP
DISCRIM	ODS Tables Produced by PROC DISCRIM
FACTOR	ODS Tables Produced by PROC FACTOR
FASTCLUS	ODS Tables Produced by PROC FASTCLUS
FMM	ODS Tables Produced by PROC FMM
FREQ	ODS Tables Produced by PROC FREQ

For detailed information, see the procedures in *SAS/STAT User's Guide*	
Procedure	**ODS Table Names**
GAM	ODS Tables Produced by PROC GAM
GAMPL	ODS Tables Produced by PROC GAMPL
GEE	ODS Tables Produced by PROC GEE
GENMOD	ODS Table Produced by PROC GENMOD
GLIMMIX	ODS Tables Produced by Proc GLIMMIX
GLM	ODS Tables Produced by PROC GLM
GLMMOD	ODS Tables Produced by PROC GLMMOD
GLMPOWER	ODS Tables Produced by PROC GLMPOWER
GLMSELECT	ODS Tables Produced by PROC GLMSELECT
HPCANDISC	ODS Tables Produced by PROC HPCANDISC
HPFMM	ODS Tables Produced by PROC HPFMM
HPGENSELECT	ODS Tables Produced by PROC HPGENSELECT
HPLMIXED	ODS Tables Produced by PROC HPLMIXED
HPLOGISTIC	ODS Tables Produced by PROC HPLOGISTIC
HPMIXED	ODS Tables Produced by PROC HPMIXED
HPNLMOD	ODS Tables Produced by PROC HPNLMOD
HPPLS	ODS Tables Produced by PROC HPPLS
HPPRINCOMP	ODS Tables Produced by PROC HPPRINCOMP
HPQUANTSELECT	ODS Tables Produced by PROC HPQUANTSELECT
HPREG	ODS Tables Produced by PROC HPREG
HPSPLIT	ODS Tables Produced by PROC HPSPLIT
ICLIFETEST	ODS Tables Produced by PROC ICLIFETEST
ICPHREG	ODS Tables Produced by PROC ICPHREG

For detailed information, see the procedures in *SAS/STAT User's Guide*	
Procedure	**ODS Table Names**
INBREED	ODS Tables Produced by PROC INBREED
IRT	ODS Tables Produced by PROC IRT
KDE	ODS Tables Produced by PROC KDE
KRIGE2D	ODS Tables Produced by PROC KRIGE2d
LATTICE	ODS Tables Produced by PROC LATTICE
LIFEREG	ODS Tables Produced by PROC LIFEREG
LIFETEST	ODS Test Produced by PROC LIFETEST
LOESS	ODS Tables Produced by PROC LOESS
LOGISTIC	ODS Tables Produced by PROC LOGISTIC
MCMC	ODS Tables Produced by PROC MCMC
MDS	ODS Tables Produced by PROC MDS
MI	ODS Table Produced by PROC MI
MIANALYZE	ODS Tables Produced by PROC MIANALYZE
MIXED	ODS Tables Produced by PROC MIXED
MODECLUS	ODS Tables Produced by PROC MODECLUS
MULTTEST	ODS Tables Produced by PROC MULTTEST
NESTED	ODS Tables Produced by PROC NESTED
NLIN	ODS Tables Produced by PROC NLIN
NLMIXED	ODS Tables Produced by PROC NLMIXED
NPAR1WAY	ODS Tables Produced by PROC NPAR1WAY
ORTHOREG	ODS Tables Produced by PROC ORTHOREG
PHREG	ODS Tables Produced by PROC PHREG
PLAN	ODS Tables Produced by PROC PLAN

For detailed information, see the procedures in *SAS/STAT User's Guide*	
Procedure	**ODS Table Names**
PLM	ODS Tables Produced by PROC PLM
PLS	ODS Tables Produced by PROC PLS
POWER	ODS Tables Produced by PROC POWER
PRINCOMP	ODS Tables Produced by PROC PRINCOMP
PRINQUAL	ODS Tables Produced by PROC PRINQUAL
PROBIT	ODS Tables Produced by PROC PROBIT
PSMATCH	ODS Tables Produced by PROC PSMATCH
QUANTLIFE	ODS Tables Produced by PROC QUANTLIFE
QUANTREG	ODS Tables Produced by PROC QUANTREG
QUANTSELECT	ODS Tables Produced by PROC QUANTSELECT
REG	ODS Tables Produced by PROC REG
RMSTREG	ODS Tables Produced by PROC RMSTREG
ROBUSTREG	ODS Tables Produced by PROC ROBUSTREG
RSREG	ODS Tables Produced by PROC RSREG
SEQDESIGN	ODS Tables Produced by PROC SEQDESIGN
SEQTEST	ODS Tables Produced by PROC
SIM2D	ODS Tables Produced by PROC SIM2D
SPP	ODS Tables Produced by PROC SPP
STDIZE	ODS Tables Produced by PROC STDIZE
STDRATE	ODS Tables Produced by PROC STDRATE
STEPDISC	ODS Tables Produced by PROC STEPDISC
SURVEYFREQ	ODS Tables Produced by PROC SURVEYFREQ
SURVEYIMPUTE	ODS Tables Produced by PROC SURVEYIMPUTE

For detailed information, see the procedures in *SAS/STAT User's Guide*	
Procedure	**ODS Table Names**
SURVEYLOGISTIC	ODS Tables Produced by PROC SURVEYLOGISTIC
SURVEYMEANS	ODS Tables Produced by PROC SURVEYMEANS
SURVEYPHREG	ODS Tables Produced by PROC SURVEYPHREG
SURVEYREG	ODS Tables Produced by PROC SURVEYREG
SURVEYSELECT	ODS Tables Produced by PROC SURVEYSELECT
TPSPLINE	ODS Tables Produced by PROC TPSLINE
TRANSREG	ODS Tables Produced by PROC TRANSREG
TREE	ODS Tables Produced by PROC TREE
TTEST	ODS Tables Produced by PROC TTEST
VARCLUS	ODS Tables Produced by PROC VARCLUS
VARCOMP	ODS Tables Produced by PROC VARCOMP
VARIOGRAM	ODS Tables Produced by PROC VARIOGRAM

ODS Table Names Produced by SAS/STAT High-Performance Procedures

Table A13.15 *ODS Table Names Produced by SAS/STAT High-Performance Procedures*

For detailed information, see the procedures in *SAS/STAT User's Guide*	
Procedure	**ODS Table Names**
GAMPL	ODS Tables Produced by PROC GAMPL
HPCANDISC	ODS Tables Produced by PROC HPCANDISC

For detailed information, see the procedures in *SAS/STAT User's Guide*	
Procedure	**ODS Table Names**
HPFMM	ODS Tables Produced by PROC HPFMM
HPGENSELECT	ODS Tables Produced by PROC HPGENSELECT
HPLMIXED	ODS Tables Produced by PROC HPLMIXED
HPLOGISTIC	ODS Tables Produced by PROC HPLOGISTIC
HPNLMOD	ODS Tables Produced by PROC HPNLMOD
HPPLS	ODS Tables Produced by PROC HPPLS
HPPRINCOMP	ODS Tables Produced by PROC HPPRINCOMP
HPQUANTSELECT	ODS Tables Produced by PROC HPQUANTSELECT
HPREG	ODS Tables Produced by PROC HPREG
HPSPLIT	ODS Tables Produced by PROC HPSPLIT

ODS Table Names Produced by SAS Enterprise Miner High-Performance Procedures

Table A13.16 *ODS Table Names Produced by SAS Enterprise Miner High-Performance Procedures*

For detailed information, see the procedures in *SAS Enterprise Miner: High-Performance Procedures*	
Procedure	**ODS Table Names**
HP4SCORE	ODS Tables Produced by PROC HP4SCORE
HPCLUS	ODS Tables Produced by PROC HPCLUS
HPDECIDE	ODS Tables Produced by PROC HPDECIDE

For detailed information, see the procedures in *SAS Enterprise Miner: High-Performance Procedures*	
Procedure	**ODS Table Names**
HPFOREST	ODS Tables Produced by PROC HPFOREST
HPNEURAL	ODS Tables Produced by PROC HPNEURAL
HPREDUCE	ODS Tables Produced by PROC HPREDUCE
HPSVM	ODS Tables Produced by PROC HPSVM

ODS Table Names Produced by SAS/ETS Procedures

This table lists the output object table names that SAS/ETS procedures produce. You must license SAS/ETS software in order to produce these output objects. The table provides the name of each table, a description of what the table contains, and the option, if any, that creates the output object table. For more information about SAS/ETS procedures, see *SAS/ETS User's Guide*.

Table A13.17 *ODS Table Names Produced by SAS/ETS Procedures*

For detailed information, see the procedures in *SAS/ETS User's Guide*	
Procedure	**ODS Table Names**
ARIMA	ODS Tables Produced by PROC ARIMA
AUTOREG	ODS Tables Produced by PROC AUTOREG
COPULA	ODS Tables Produced by PROC COPULA
COUNTREG	ODS Tables Produced by PROC COUNTREG
ENTROPY	ODS Tables Produced by PROC ENTROPY
ESM	ODS Tables Produced by PROC ESM
HPCOUNTREG	ODS Tables Produced by PROC HPCOUNTREG
HPPANEL	ODS Tables Produced by PROC HPPANEL

For detailed information, see the procedures in *SAS/ETS User's Guide*	
Procedure	**ODS Table Names**
HPQLIM	ODS Tables Produced by PROC HPQLIM
LOAN	ODS Tables Produced by PROC LOAN
MDC	ODS Tables Produced by PROC MDC
PANEL	ODS Tables Produced by PROC PANEL
QLIM	ODS Tables Produced by PROC QLIM
SIMILARITY	ODS Tables Produced by PROC SIMILARITY
SIMLIN	ODS Tables Produced by PROC SIMLIN
SPATIALREG	ODS Tables Produced by PROC SPATIALREG
SPECTRA	ODS Tables Produced by PROC SPECTRA
SSM	ODS Tables Produced by PROC SSM
STATESPACE	ODS Tables Produced by PROC STATESPACE
SYSLIN	ODS Tables Produced by PROC SYSLIN
TIMEDATA	ODS Tables Produced by PROC TIMEDATA
TIMESERIES	ODS Tables Produced by PROC TIMESERIES
TSCSREG	ODS Tables Produced by PROC TSCSREG
UCM	ODS Tables Produced by PROC UCM
VARMAX	ODS Tables Produced by PROC VARMAX
X11	ODS Tables Produced by PROC X11
X13	ODS Tables Produced by PROC X13

ODS Table Names Produced by SAS/ETS High-Performance Procedures

Table A13.18 *ODS Table Names Produced by SAS/ETS High-Performance Procedures*

For detailed information, see the procedures in *SAS/ETS User's Guide*	
Procedure	**ODS Table Names**
HPCOUNTREG	ODS Tables Produced by PROC HPCOUNTREG
HPPANEL	ODS Tables Produced by PROC HPPANEL
HPQLIM	ODS Tables Produced by PROC HPQLIM

ODS Table Names Produced by SAS/QC Procedures

Table A13.19 *ODS Table Names Produced by SAS/QC Procedures*

For detailed information, see the procedures in *SAS/QC User's Guide*	
Procedure	**ODS Table Names**
FACTEX	ODS Tables Produced by PROC FACTEX
MVPMODEL	ODS Tables Produced by PROC MVPMODEL
OPTEX	ODS Tables Produced by PROC OPTEX
RAREEVENTS	ODS Tables Produced by PROC RAREEVENTS
RELIABILITY	ODS Tables Produced by PROC RELIABILITY

ODS Table Names Produced by SAS/OR Procedures

Table A13.20 *ODS Table Names Produced by SAS/OR Procedures*

For detailed information, see the procedures in the following documents:

- *SAS/OR User's Guide: Project Management*
- *SAS/OR User's Guide: Mathematical Programming*
- *SAS/OR User's Guide: Local Search Optimization*
- *SAS/OR User's Guide: Network Optimization Algorithms*

Procedure	ODS Table Name
DTREE	ODS Tables Produced by PROC DTREE
OPTLP	ODS Tables Produced by PROC OPTLP
OPTLSO	ODS Tables Produced by PROC OPTLSO
OPTMILP	ODS Tables Produced by PROC OPTMILP
OPTMODEL	ODS Tables Produced by PROC OPTMODEL
OPTNET	ODS Tables Produced by PROC OPTNET
OPTQP	ODS Tables Produced by PROC OPTQP

ODS Table Names Produced by Graph Algorithms and Network Analysis Procedure

Table A13.21 *ODS Table Names Produced by Graph Algorithms and Network Analysis Procedure*

Procedure	ODS Table Name
OPTGRAPH	ODS Tables Produced by PROC OPTGRAPH

Appendix 2

Example Code

Creating the DistrData Data Set 909
Creating the Univ ODS Document 909

Creating the DistrData Data Set

```
data distrdata;
   drop n;
   label Normal_x='Normal Random Variable'
         Exponential_x='Exponential Random Variable';
   do n=1 to 100;
      Normal_x=10*rannor(53124)+50;
      Exponential_x=ranexp(18746363);
      output;
   end;
run;
```

Creating the Univ ODS Document

```
data distrdata;
   drop n;
   label Normal_x='Normal Random Variable'
         Exponential_x='Exponential Random Variable';
   do n=1 to 100;
      Normal_x=10*rannor(53124)+50;
      Exponential_x=ranexp(18746363);
      output;
   end;
 run;
```

```
ods document name=univ;

title '100 Obs Sampled from a Normal Distribution';
proc univariate data=distrdata noprint;
  var Normal_x;

  histogram Normal_x /normal(noprint) cbarline=grey name='normal';
run;

  title '100 Obs Sampled from an Exponential Distribution';

proc univariate data=distrdata noprint;
  var Exponential_x;

  histogram /exp(fill l=3) cfill=yellow midpoints=.05 to 5.55 by .25
             name='exp';
run;

  ods document close;
title;

quit;
```

Appendix 3

Lua License

Lua License 911

Lua License

Copyright © 1994–2019 Lua.org, PUC-Rio.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.