

CHAPTER

5

The TEMPLATE Procedure

<i>Overview</i>	131
<i>Concepts</i>	132
<i>What Is a Template Store?</i>	132
<i>What Is the Default Style Definition Like?</i>	133
<i>How Are Values in Table Columns Justified?</i>	138
<i>How Do Style-Definition Inheritance and Style-Element Inheritance Work?</i>	139
<i>Style Element Inheritance in a Style Definition with No Parent</i>	140
<i>Style-Definition Inheritance</i>	146
<i>Style-Element Inheritance in a Style Definition with a Parent</i>	147
<i>Procedure Syntax</i>	156
<i>PROC TEMPLATE Statement</i>	157
<i>DEFINE Statement</i>	157
<i>DEFINE COLUMN Statement</i>	159
<i>DEFINE FOOTER Statement</i>	176
<i>DEFINE HEADER Statement</i>	177
<i>DEFINE STYLE Statement</i>	188
<i>DEFINE TABLE Statement</i>	209
<i>DELETE Statement</i>	227
<i>EDIT Statement</i>	227
<i>LINK Statement</i>	229
<i>LIST Statement</i>	229
<i>PATH Statement</i>	231
<i>SOURCE Statement</i>	232
<i>TEST Statement</i>	233
<i>Examples</i>	233
<i>Example 1: Customizing a Table Definition that a SAS Procedure Uses</i>	233
<i>Example 2: Creating a New Table Definition</i>	238
<i>Example 3: Setting the Style Element for Cells Based on Their Values</i>	247
<i>Example 4: Creating a Stand-alone Style Definition</i>	252
<i>Example 5: Creating and Modifying a Style Definition with User-Defined Attributes</i>	258
<i>Example 6: Modifying the Default Style Definition for the HTML Destination</i>	265

Overview

By default, ODS output is formatted according to instructions in various definitions that the procedure or DATA step points to. However, ODS provides ways for you to customize the output. You can customize the output for an entire SAS job, or you can customize the output for a single output object.

To customize the presentation aspects of the output at the level of the entire SAS job, use a style definition. To customize the output for a single output object, use a table

definition. The `TEMPLATE` procedure creates and modifies these definitions, which the Output Delivery System then uses to create formatted output.

A style definition

describes how to render the presentation aspects (color, font face, font size, and so forth) of an entire SAS job. A style definition determines the overall look of the documents that use it. Each style definition is composed of style elements.

A style element

is a collection of style attributes that apply to a particular part of the output. For instance, a style element may contain instructions for the presentation of column headers or for the presentation of the data inside cells. Style elements may also specify default colors and fonts for output that uses the style definition. Each style attribute specifies a value for one aspect of the presentation. For instance, the `BACKGROUND=` attribute specifies the color for the background of an HTML table, and the `FONT_STYLE=` attribute specifies whether to use a Roman, a slant, or an italic font.

A table definition

describes how to render the output for a tabular output object. (Almost all ODS output is tabular.) A table definition determines the order of table headers and footers, the order of columns, and the overall look of the output object that uses it. Each table definition contains or references table elements.

A 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 instance `FORMAT=` specifies the SAS format to use in a column. However, some attributes describe presentation aspects of the data.

Note: You can also define columns, headers, and footers outside of a table definition. Any table definition can then reference these table elements. Δ

Concepts

What Is a Template Store?

A template store is a specific kind of item store (a kind of SAS file) that stores the definitions that `PROC TEMPLATE` creates. Definitions that SAS Institute provides are in the template store `SASHELP.TMPLMST`. You can store your definitions in any template store that you can write to (see “`PATH` Statement” on page 231).

A template store can contain multiple levels (also referred to 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 data library that the libref references.

To view the contents of a template store in the SAS windowing environment, complete the following steps:

1 Select

►

from the Explorer.

- 2 In the Results window, select the Results folder. With your cursor on this folder, use your right mouse button to open the Templates window.
- 3 Click on the plus sign that is next to an icon to view the contents of a template store or a directory in a template store. If there is no plus sign next to the icon, double click on the icon to view the contents of that directory.

You can also use the LIST statement to view the contents of one or more template stores (see “LIST Statement” on page 229).

What Is the Default Style Definition Like?

The default style definition for the HTML destination is stored in `STYLES.DEFAULT` in the template store `SASHELP.TMPLMST`. You can view the style definition from the `TEMPLATE` window, or you can submit this `PROC TEMPLATE` step to write the style definition to the SAS log:

```
proc template;
  source styles.default;
run;
```

Note: The default style definition for the Printer destination is stored in `STYLES.PRINTER` in the template store `SASHELP.TMPLMST`. This style definition is similar to `STYLES.DEFAULT`. In fact, it inherits most of its attributes from `STYLES.DEFAULT`. Δ

You may want to alter parts of the default style definition, and Example 6 on page 265 shows you how to do so. When you try to modify the style definition for use at your site, it is helpful to know what each style element in the style definition is for. The following table lists all the style elements in the default style definition. The indentation in the table indicates the relationship between the style elements. An element that is indented inherits from the element above it and to its left. The table also provides a brief description of the purpose of each style element. An abstract style element is one that is not used to render any style element but provides a basis for one or more style elements to inherit.

Table 5.1 Style Elements that Are Available in the Default Style definition

Style Element	Description
fonts	Establishes a list of fonts.
color_list	Establishes a list of color names and their RGB values.
colors	Associates parts of SAS output with colors from <code>color_list</code> .
html	Provides HTML for specific parts of the output.
text	Provides text for specific parts of the output.
container	Abstract: provides a basis for all containers.

Table 5.1 (continued)

Style Element	Description
index	Abstract: provides a basis for the contents and page files.
indexprocname	Renders the procedure name in the body file.
contentprocname	Renders the procedure name in the contents file.
contentproclabel	Renders the procedure label in the contents file.
pagesprocname	Renders the procedure name in the page file.
pagesproclabel	Renders the procedure label in the page file.
indexaction	Abstract: determines what happens when the mouse is positioned over a folder or item.
folderaction	Determines what happens when the mouse is positioned over a folder.
itemaction	Determines what happens when the mouse is positioned over an item.
procnameaction	Determines what happens when the mouse is positioned over the procedure name in the table of contents.
titleaction	Determines what happens when the mouse is positioned over a SAS title.
indextitle	Abstract: controls the title of the contents and page files.
contenttitle	Renders the title in the contents file.
pagestitle	Renders the title in the page file.
document	Abstract: controls the various output files.
body	Renders the body file.
frame	Renders the frame file.
contents	Renders the contents file.
pages	Renders the page file.
date	Abstract: controls the contents of date fields.

Table 5.1 (continued)

Style Element	Description
bodydate	Renders the date field in the body file.
contentsdate	Renders the date field in the contents file.
pagesdate	Renders the date field in the page file.
indexitem	Abstract: controls the items in the contents and page files.
contentfolder	Controls the generic folder definition in the contents file.
bycontentfolder	Controls the byline folder definition in the contents file.
contentitem	Renders the lowest level of the hierarchy in a contents file.
pagestitem	Renders the lowest level of the hierarchy in a page file.
systitleandfootercontainer	Renders the container for system headers and footers (provided by TITLE and FOOTNOTE statements or by GTITLE and GFOOTNOTE statements in combination with the NOGTITLE and NOGFOOTNOTE options in the ODS HTML statement).
titleandnotecontainer	Renders the container for titles and notes that the procedure provides.
titlesandfooters	Abstract: controls the text of the system titles and footers.
systemtitle	Renders the text of system titles.
systemfooter	Renders the text of system footers.
pageno	Renders the text of the page number.
byline	Renders the text of the byline.
proctitle	Renders the text of titles that the procedure provides.
proctitlefixed	Renders the text of titles that the procedure provides with a fixed font.
bylinecontainer	Renders the container for the byline.
output	Abstract: controls basic presentation of the output.

Table 5.1 (continued)

Style Element	Description
table	Renders output that is a table.
batch	Renders output (like lineprinter plots and calendars) that requires a fixed font.
note	Abstract: controls the container for the text that precedes notes, warning, and errors from SAS.
notebanner	Renders the text that precedes the contents of a note.
notecontent	Renders the contents of a note.
notecontentfixed	Renders the contents of a note with a fixed font.
warnbanner	Renders the text that precedes the contents of a warning.
warncontent	Renders the contents of a warning.
warncontentfixed	Renders the contents of a warning with a fixed font.
errorbanner	Renders the text that precedes the contents of an error.
errorcontent	Renders the contents of an error.
errorcontentfixed	Renders the contents of an error with a fixed font.
fatalbanner	Renders the text that precedes the contents of a fatal error.
fatalcontent	Renders the contents of a fatal error.
fatalcontentfixed	Renders the contents of a fatal error with a fixed font.
cell	Abstract: controls table cells.
data	Renders the data in table cells.
datafixed	Renders the data in table cells with a fixed font.
dataempty	Renders empty table cells.
dataemphasis	Renders data in table cells with emphasis.
dataemphasisfixed	Renders data in table cells with emphasis and a fixed font.
datastrong	Renders data in cells with more emphasis than dataemphasis .

Table 5.1 (continued)

Style Element	Description
<code>datastrongfixed</code>	Renders data in table cells with more emphasis than dataemphasis and with a fixed font.
<code>headersandfooters</code>	Abstract: controls table headers and footers.
<code>header</code>	Renders the headers of a table.
<code>headerfixed</code>	Renders the headers of a table with a fixed font.
<code>headerempty</code>	Renders empty table headers.
<code>headeremphasis</code>	Renders the headers of a table with emphasis.
<code>headeremphasisfixed</code>	Renders the headers of a table with emphasis and with a fixed font.
<code>headerstrong</code>	Renders the headers of a table with more emphasis than headeremphasis .
<code>headerstrongfixed</code>	Renders the headers of a table with more emphasis than headeremphasis and with a fixed font.
<code>rowheader</code>	Renders row headers.
<code>rowheaderfixed</code>	Renders row headers with a fixed font.
<code>rowheaderempty</code>	Renders empty row headers.
<code>rowheaderemphasis</code>	Renders row headers with emphasis.
<code>rowheaderemphasisfixed</code>	Renders row headers with emphasis and with a fixed font.
<code>rowheaderstrong</code>	Renders row headers with more emphasis than rowheaderemphasis .
<code>rowheaderstrongfixed</code>	Renders row headers with more emphasis than rowheaderemphasis and with a fixed font.
<code>footer</code>	Renders the footers of a table.
<code>footerfixed</code>	Renders the footers of a table with a fixed font.
<code>footerempty</code>	Renders empty table footers.
<code>footeremphasis</code>	Renders the footers of a table with emphasis.

Table 5.1 (continued)

Style Element	Description
footeremphasisfixed	Renders the footers of a table with emphasis and with a fixed font.
footerstrong	Renders the footers of a table with more emphasis than footeremphasis .
footerstrongfixed	Renders the footers of a table with more emphasis than footeremphasis and with a fixed font.
rowfooter	Renders row footers.
rowfooterfixed	Renders row footers with a fixed font.
rowfooterempty	Renders empty row footers.
rowfooteremphasis	Renders row footers with emphasis.
rowfooteremphasisfixed	Renders row footers with emphasis and with a fixed font.
rowfooterstrong	Renders row footers with more emphasis than rowfooteremphasis .
rowfooterstrongfixed	Renders row footers with more emphasis than rowfooteremphasis and with a fixed font.
caption	Abstract: controls the caption field in PROC TABULATE.
beforecaption	Renders captions that precede the table.
aftercaption	Renders captions that follow the table.

How Are Values in Table Columns Justified?

The process of justifying the values in columns in Listing output involves the variable's format 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, JUST= has no apparent affect 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.

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 look like this 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, the value of JUST= does affect the placement of the values because the format field has room to move within the column.

just=left	just=center	just=right
*****	*****	*****
123.45	123.45	123.45
234.50	234.50	234.50
.	.	.
987.65	987.65	987.65

Notice that the decimal points remain aligned but that the numbers shift in relation to the column width. Now, if you add JUSTIFY=ON, the values are formatted within the column without regard to the format width. The results are as follows:

justify=on just=left	justify=on just=center	justify=on just=right
*****	*****	*****
123.45	123.45	123.45
234.50	234.50	234.50
.	.	.
987.65	987.65	987.65

The HTML and Printer destinations always justify the values in columns as if JUSTIFY=ON.

How Do Style-Definition Inheritance and Style-Element Inheritance Work?

When you use PROC TEMPLATE to create style definitions, it is important to understand inheritance. There are two types of inheritance: style-definition inheritance

and style-element inheritance. Recall that a style definition is composed of style elements (see “Overview” on page 131). A style definition is created with the DEFINE STYLE statement and its substatements and attributes. Style elements are created with the STYLE and REPLACE statements and their attributes.

The PARENT= attribute, used with the DEFINE STYLE statement, controls style-definition inheritance. When you specify a parent for a style definition, all the style elements and attributes and statements that are specified in the parent’s definition are used in the new definition unless the new definition overrides them.

The STYLE and REPLACE statements, used with the DEFINE STYLE statement, control style-element inheritance. They augment or override the attributes of a particular style element. You can use the STYLE statement in either a style definition that has no parent or a style definition that has a parent. However, you can use the REPLACE statement only in a style definition that has a parent.

This section explains style-definition inheritance and style-element inheritance, beginning with the simpler case of style-element inheritance in a style definition that has no parent and progressing to more complicated cases. The focus here is on PROC TEMPLATE and the DEFINE STYLE statement, so only the PROC TEMPLATE code that creates the style definitions appears in the text. However, in order to produce the HTML output that is shown here, it is necessary to create a customized table (with the DEFINE TABLE statement in PROC TEMPLATE) and to bind that table to a data set (with the TEST statement in PROC TEMPLATE). The complete code that produces each piece of output is in “Programs that Illustrate Inheritance” on page 283.

Style Element Inheritance in a Style Definition with No Parent

When you create a style definition, you use a STYLE statement to create each style element in the definition. For instance, the following PROC TEMPLATE step creates a style definition, `concepts.style1`, that contains one style element, `celldatasimple`. This style element uses the Arial font face, a light blue background, and a white foreground:

Example Code 5.1 Creating a Style Definition with One Style Element

```
proc template;
  define style concepts.style1;
    style celldatasimple /
      font_face=arial
      background=very light vivid blue
      foreground=white;
  end;
run;
```

The style element `celldatasimple` does not inherit any attributes from any other element. It is simply created with these three attributes. All other attributes are set by the browser when a table is rendered in HTML with this style definition. The following HTML output uses this style definition:

Display 5.1 Using a Style Definition with One Style Element

The style definition that this HTML output uses contains only one style element: **celldatasimple**. All three columns use this style element. **celldatasimple** uses these attributes:

FONT_FACE=arial

BACKGROUND=very light vivid blue

FOREGROUND=white.

Country	Grain	Kilotons
Brazil	Rice	10035
China	Rice	190100
India	Rice	120012
Indonesia	Rice	51165
United States	Rice	7771

Now, suppose that you want an additional style element for cells. This style element emphasizes the data by using an italic font. It uses the same font face and background color as **celldatasimple**, but it uses blue instead of white for the foreground color. You can create the new style element completely on its own:

Example Code 5.2 Creating a Second Style Element Independently

```
proc template;
  define style concepts.style1;
    style celldatasimple /
      font_face=arial
      background=very light vivid blue;
      foreground=white
    style celldataemphasis /
      font_face=arial
      background=very light vivid blue
      foreground=blue
      font_style=italic;
  end;
run;
```

Alternatively, you can create **celldataemphasis** from **celldatasimple**:

Example Code 5.3 Creating a Second Style Element from the Existing Style Element

```
proc template;
  define style concepts.style1;
    style celldatasimple /
```

```

        font_face=arial
        background=very light vivid blue
        foreground=white;
    style celldataemphasis from celldatasimple /
        foreground=blue
        font_style=italic;
end;
run;

```

The PROC TEMPLATE steps in Example Code 5.2 on page 141 and Example Code 5.3 on page 141 produce identical results. In both cases, `celldatasimple` contains these attributes:

- FONT_FACE=arial
- BACKGROUND=very light vivid blue
- FOREGROUND=white.

`celldataemphasis` has these attributes:

- FONT_FACE=arial (inherited from `celldatasimple`)
- BACKGROUND=very light vivid blue (inherited from `celldatasimple`)
- FOREGROUND=blue (modified in `celldataemphasis`)
- FONT_STYLE=italic (added in `celldataemphasis`).

The following HTML output uses this style definition:

Display 5.2 A Style Definition with Two Style Elements

The style definition that this HTML output uses contains two style elements: `celldatasimple` and `celldataemphasis`. The columns for `Country` and `Kilotons` use `celldatasimple`. The column for `grain` uses `celldataemphasis`.

Country	Grain	Kilotons
Brazil	<i>Rice</i>	10035
China	<i>Rice</i>	190100
India	<i>Rice</i>	120012
Indonesia	<i>Rice</i>	51165
United States	<i>Rice</i>	7771

Although the PROC TEMPLATE steps in Example Code 5.2 on page 141 and Example Code 5.3 on page 141 produce identical HTML output, there is an important difference between them. In the Example 5.8, the program does not use style-element inheritance.

celldataemphasis is created independently of **celldatasimple**, so a change to **celldatasimple** does not affect **celldataemphasis**. For example, if you change the **STYLE** statement that creates **celldatasimple** so that the font face is Times, Example Code 5.2 on page 141 creates **celldataemphasis** with Arial as the font face.

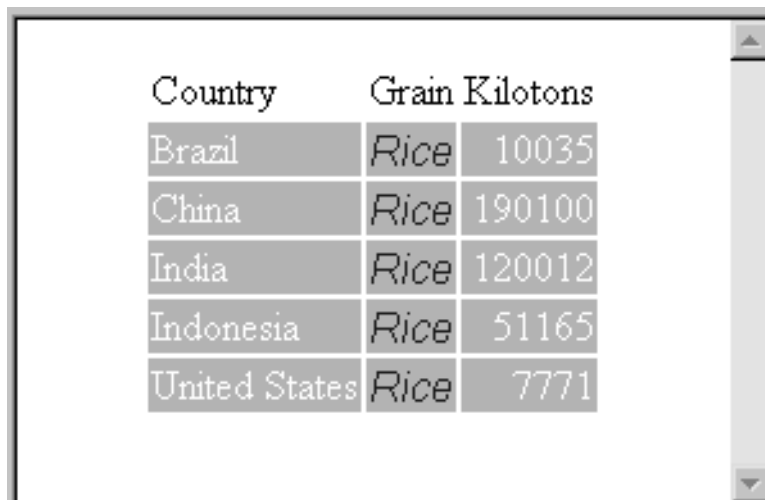
Example Code 5.4 Changing the Font Face in Only One Style Element

```
proc template;
  define style concepts.style1;
    style celldatasimple
      'The change to FONT_FACE= does not
      affect celldataemphasis.' /
      font_face=times
      background=very light vivid blue
      foreground=white;
    style celldataemphasis /
      font_face=arial
      background=very light vivid blue
      foreground=blue
      font_style=italic;
  end;
```

The following HTML output uses this style definition:

Display 5.3 Changing the Font Face in Only One Style Element

Here, the font face in the style element **celldatasimple**, which is used for the first and third columns in the HTML output, has changed from Arial to Times. However, **celldataemphasis**, which is used for the second column, still uses the Arial font face because it does not inherit any attributes from **celldatasimple**.



Country	Grain	Kilotons
Brazil	<i>Rice</i>	10035
China	<i>Rice</i>	190100
India	<i>Rice</i>	120012
Indonesia	<i>Rice</i>	51165
United States	<i>Rice</i>	7771

On the other hand, if you change the font face for **celldatasimple** from Arial to Times in Example Code 5.3 on page 141, **celldataemphasis** does use the Times font face because in this PROC TEMPLATE step, **celldataemphasis** inherits all the attributes from **celldatasimple**.

Example Code 5.5 Changing the Font Face in the Parent and Child Style Elements

```

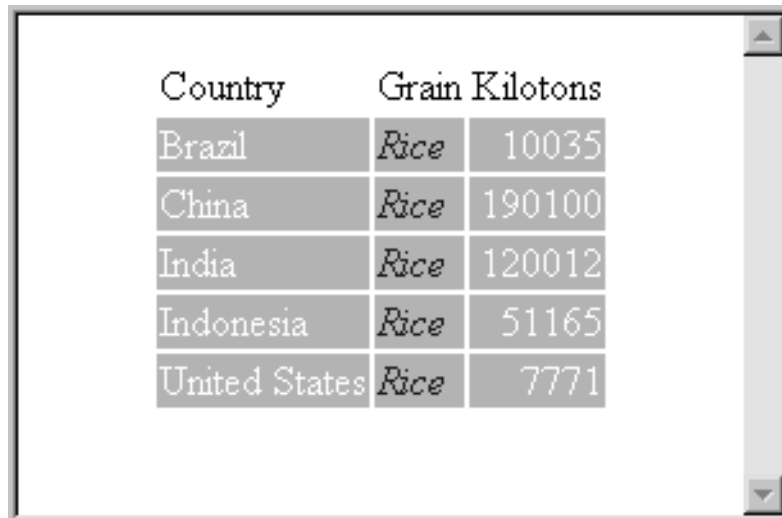
proc template;
  define style concepts.style1;
    style celldatasimple
      'The change to FONT_FACE= is passed to
      celldataemphasis, which inherits all the
      attributes of celldatasimple.' /
      font_face=times
      background=very light vivid blue
      foreground=white;
    style celldataemphasis from celldatasimple /
      foreground=blue
      font_style=italic;
  end;
run;

```

The following HTML output uses this style definition:

Display 5.4 Inheriting a Change to a Style Element

In this case, the change to the Times font face in **celldatasimple** is inherited by **celldataemphasis**. Both style elements use the Times font face. The only attributes that differ between the two style elements are ones that were explicitly redefined in the definition of **celldataemphasis** (the FOREGROUND= attribute, which was changed, and the FONT_STYLE= attribute, which was added). The columns for **Country** and **Kilotons** use **celldatasimple**. The column for **Grain** uses **celldataemphasis**.



Country	Grain	Kilotons
Brazil	<i>Rice</i>	10035
China	<i>Rice</i>	190100
India	<i>Rice</i>	120012
Indonesia	<i>Rice</i>	51165
United States	<i>Rice</i>	7771

Let's add a third style element to the style definition. This style element further emphasizes the data by using a large, bold, italic font. Again, you can create the new style element from scratch, or you can derive it from either of the other style elements. The following program creates **celldatalarge** from **celldataemphasis**:

Example Code 5.6 Creating the Style Element *celldatalarge*

```
proc template;
  define style concepts.style1;
    style celldatasimple /
      font_face=arial
      background=very light vivid blue
      foreground=white;
    style celldataemphasis from celldatasimple /
      foreground=blue
      font_style=italic;
    style celldatalarge from celldataemphasis /
      font_weight=bold
      font_size=5;
  end;
run;
```

HTML output that uses the new style definition appears in Display 5.5 on page 146. The style elements **celldatasimple** and **celldataemphasis** have not changed. **celldatasimple** has these attributes:

- **FONT_FACE**=arial
- **BACKGROUND**=very light vivid blue
- **FOREGROUND**=white.

celldataemphasis has these attributes:

- **FONT_FACE**=arial (inherited from **celldatasimple**)
- **BACKGROUND**=very light vivid blue (inherited from **celldatasimple**)
- **FOREGROUND**=blue (modified in **celldataemphasis**)
- **FONT_STYLE**=italic (added in **celldataemphasis**).

The new style element, **celldatalarge**, has these attributes:

- **FONT_FACE**=arial (inherited from **celldataemphasis**, which inherited it from **celldatasimple**)
- **BACKGROUND**=very light vivid blue (inherited from **celldataemphasis**, which inherited it from **celldatasimple**)
- **FOREGROUND**=blue (inherited from **celldataemphasis**)
- **FONT_STYLE**=italic (inherited from **celldataemphasis**)
- **FONT_WEIGHT**=bold (added in **celldatalarge**)
- **FONT_SIZE**=5 (added in **celldatalarge**).

Display 5.5 Adding the Style Element *celldatalarge*

The style definition that this HTML output uses contains three style elements: **celldatasimple**, **celldataemphasis**, and **celldatalarge**. The column for **Country** uses **celldatasimple**. The column for **Grain** uses **celldataemphasis**. The column for **Kilotons** uses **celldatalarge**.

Country	Grain	Kilotons
Brazil	<i>Rice</i>	10035
China	<i>Rice</i>	190100
India	<i>Rice</i>	120012
Indonesia	<i>Rice</i>	51165
United States	<i>Rice</i>	7771

In this case, **celldatalarge** inherits style attributes from **celldataemphasis**, and **celldataemphasis** inherits from **celldatasimple**. If you change the font face in **celldatasimple**, the font face in both other style elements changes. If you change the font style or foreground color in **celldataemphasis**, the font style or foreground color in **celldatalarge** also changes. Changes to **celldatalarge** affect only **celldatalarge** because no style element inherits from it.

The following points summarize style-element inheritance in a style definition that does not have a parent:

- You can create a new style element from any existing style element.
- The new style element inherits all the attributes from its parent.
- You can specify additional attributes in the new style definition. The attributes are added to the attributes that the element inherits.
- You can change the value of an inherited attribute by respecifying it in the definition of the new style element.

Style-Definition Inheritance

If you use the **PARENT=** attribute in a style definition, the new style definition inherits the entire style definition. You can then add new style elements or modify style elements that are inherited from the parent.

Let's continue with **concepts.style1**, which was created in "Style Element Inheritance in a Style Definition with No Parent" on page 140. The following program creates a new style definition, **concepts.style2**, which inherits the entire style definition from its parent, **concepts.style1**. At this point, the two style definitions are identical:

Example Code 5.7 Using Style-Definition Inheritance to Create a New Style Definition

```
proc template;
  define style concepts.style1;
    style celldatasimple /
      font_face=arial
      background=very light vivid blue
      foreground=white;
    style celldataemphasis from celldatasimple /
      foreground=blue
      font_style=italic;
    style celldatalarge from celldataemphasis /
      font_weight=bold
      font_size=5;
  end;
run;

proc template;
  define style concepts.style2;
    parent=concepts.style1;
  end;
run;
```

For this new style definition to be useful, it should differ from its parent. There are several ways to change the new style definition. The following section explains style-element inheritance in a style definition that has a parent.

Style-Element Inheritance in a Style Definition with a Parent

Let's start by adding a new style element to `concepts.style2`. The following program adds `celldatasmall`, a style element that does not exist in the parent style definition. Its definition is not based on any other style element.

Example Code 5.8 Creating a Style Element Independently in a Style Definition with a Parent

```
proc template;
  define style concepts.style1;
    style celldatasimple /
      font_face=arial
      background=very light vivid blue
      foreground=white;
    style celldataemphasis from celldatasimple /
      foreground=blue
      font_style=italic;
    style celldatalarge from celldataemphasis /
      font_weight=bold
      font_size=5;
  end;
run;

proc template;
  define style concepts.style2;
    parent=concepts.style1;
    style celldatasmall /
      font_face=arial
      background=very light vivid blue
```

```

        foreground=blue
        font_style=italic
        font_weight=bold
        font_size=2;
    end;
run;

```

If you look at the attributes for `celldatasmall`, you can see that they match the attributes for `celldatalarge` in the parent style definition except for `FONT_SIZE=`. Another way to create this new style element, then, is to create it from `celldatalarge`. You do this just as you did when you created a style element in a style definition that didn't have a parent:

Example Code 5.9 Creating a New Style Element from a Style Element in the Parent Style Definition

```

proc template;
  define style concepts.style1;
    style celldatasimple /
      font_face=arial
      background=very light vivid blue
      foreground=white;
    style celldataemphasis from celldatasimple /
      foreground=blue
      font_style=italic;
    style celldatalarge from celldataemphasis /
      font_weight=bold
      font_size=5;
  end;
run;

proc template;
  define style concepts.style2;
    parent=concepts.style1;
    style celldatasmall from celldatalarge /
      font_size=2;
  end;
run;

```

When you specify `FROM` in the `STYLE` statement in a style definition with a parent, `PROC TEMPLATE` first looks for the style element that you specify in the child style definition. If no such style element exists, it looks in the parent (and continues looking up through parents as far as is necessary or possible). In this case, because no style element called `celldatalarge` exists in `concepts.style2`, `PROC TEMPLATE` uses the style element from the parent style definition.

The style definition `concepts.style2` that is produced in Example Code 5.8 on page 147 is identical to the one that is produced in Example Code 5.9 on page 148. In both cases, the style element `celldatasmall` has these attributes:

- `FONT_FACE=arial` (inherited from `celldatalarge` through `celldataemphasis` and `celldatasimple`)
- `BACKGROUND=very light vivid blue` (inherited from `celldatalarge` through `celldataemphasis` and `celldatasimple`)
- `FOREGROUND=blue` (inherited from `celldatalarge` through `celldataemphasis`)
- `FONT_STYLE=italic` (inherited from `celldatalarge` through `celldataemphasis`)

- `FONT_WEIGHT=bold` (inherited from `celldatalarge`)
- `FONT_SIZE=2` (modified in `celldatasmall`).

The following HTML output uses `concepts.style2`:

Display 5.6 Creating a New Style Element from a Style Element in the Parent Style Definition

The style definition `concepts.style2` contains four style elements. The style definition inherits `celldatasimple`, `celldataemphasis`, and `celldatalarge` from the parent style definition, `concepts.style1`. The column for `Country` uses `celldatasimple`. The column for `grain` uses `celldataemphasis`. The first column for `Kilotons` uses `celldatalarge`. The fourth style element in the new style definition is `celldatasmall`. This style element is created in `concepts.style2`. It inherits from `celldatalarge` in `concepts.style1`. The fourth column, which repeats the values for `Kilotons`, uses `celldatasmall`.

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

Although Example Code 5.8 on page 147 and Example Code 5.9 on page 148 produce the same style definition for `concepts.style2`, they will produce different style definitions if you change the definition of `celldatalarge` in the parent (or the definition of any of the style elements that `celldatalarge` inherits from). In Example Code 5.8 on page 147, changes to `celldatalarge` do not affect `celldatasmall` because `celldatasmall` is created independently in the new style definition. It does not inherit from any style element in the parent style definition.

On the other hand, in Example Code 5.9 on page 148, changes to `celldatalarge` in the parent style definition do affect `celldatasmall` because `celldatasmall` inherits (and adds to) the attributes of `celldatalarge`. Similarly, changes to other style elements in the parent style definition do not affect `celldatasmall` in the Example Code 5.8 on page 147, but they do in the Example Code 5.9 on page 148.

For example, the following program is based on Example Code 5.9 on page 148. It changes the font face in `celldatasimple` from Arial to Times. All the other style elements, in both the parent and the child style definitions, inherit this change. The program also changes the foreground color of `celldataemphasis` to black. The style elements `celldatalarge` (in the parent style definition) and `celldatasmall` (in the new style definition) both inherit this foreground color.

Example Code 5.10 Inheriting Changes from Style Elements in the Parent Style Definition

```

proc template;
  define style concepts.style1;
    style celldatasimple /
      font_face=times
      background=very light vivid blue
      foreground=white;
    style celldataemphasis from celldatasimple /
      foreground=black
      font_style=italic;
    style celldatalarge from celldataemphasis /
      font_weight=bold
      font_size=5;
  end;
run;
proc template;
  define style concepts.style2;
    parent=concepts.style1;
    style celldatasmall from celldatalarge /
      font_size=2;
  end;
run;

```

The following HTML output uses this new style definition:

Display 5.7 Inheriting Changes to the Parent Style Definition

Changes to the style elements in the parent style definition are passed to style elements that inherit from these elements in both the parent and the child style definitions.

Country	Grain	Kilotons	Kilotons
Brazil	<i>Rice</i>	10035	10035
China	<i>Rice</i>	190100	190100
India	<i>Rice</i>	120012	120012
Indonesia	<i>Rice</i>	51165	51165
United States	<i>Rice</i>	7771	7771

Creating a new style element in a style definition that has a parent is not very different from creating a new style element in a style definition that does not have a parent. The only difference is that the style element that you specify with FROM in the STYLE statement can be in either the parent or the child style definition.

When you create a new style definition from a parent definition you can, in addition to adding new style elements, modify existing style elements. There are two ways to do this. One way uses the `STYLE` statement to make the change only to the style element that you specify. The other way uses the `REPLACE` statement to make the change to the style element that you specify and to all the style elements that inherit from that element.

The following program uses the `STYLE` statement to redefine the style element `celldataemphasis` in `concepts.style2`. It changes the background color to white:

Example Code 5.11 Redefining a Style Element with the `STYLE` Statement

```
proc template;
  define style concepts.style1;
    style celldatasimple /
      font_face=arial
      background=very light vivid blue
      foreground=white;
    style celldataemphasis from celldatasimple /
      foreground=blue
      font_style=italic;
    style celldatalarge from celldataemphasis /
      font_weight=bold
      font_size=5;
  end;
run;
proc template;
  define style concepts.style2;
    parent=concepts.style1;
    style celldataemphasis from celldataemphasis /
      background=white;
    style celldatasmall from celldatalarge /
      font_size=2;
  end;
run;
```

In this case, `celldataemphasis` in `concepts.style2` initially inherits all the attributes of `celldataemphasis` in `concepts.style1` because it is created from this style element. The inherited attributes are

- the Arial font face (which `celldataemphasis` inherits from `celldatasimple`)
- the very light vivid blue background (which `celldataemphasis` inherits from `celldatasimple`)
- the blue foreground
- the italic font style.

The `STYLE` statement that creates `celldataemphasis` in the new style definition changes the background color to white. The background color is the only difference between the `celldataemphasis` style elements in the parent and child style definitions.

But, what about `celldatalarge`? `celldatalarge` is not redefined in `concepts.style2`. It is defined only in the parent style definition, where it inherits all the attributes of `celldataemphasis`. So the question is, which `celldataemphasis` does it inherit from—the one in the parent style definition or the one in the child style definition? Does it get the white background or not?

The answer is that it does not. You can envision the process this way. The `STYLE` statement that creates `celldataemphasis` in the new style definition, affects only

those style elements that inherit from `celldataemphasis` and that are defined in the new style definition. Because `celldatalarge` is defined only in the parent style definition, it does not inherit the changes that are specified in the child. Similarly, `celldatasmall` does not inherit the white background because it inherits from `celldatalarge`. The following HTML output uses this version of `concepts.style2`:

Display 5.8 Using the STYLE Statement to Alter an Existing Style Element in the Child Style Definition

A style element that is defined with the STYLE statement in the child style definition does not pass its attributes to style elements that inherit from the like-named style element in the parent style definition. In this case, the change of the background color for `celldataemphasis` is made in the child style definition. The new background color is not inherited by `celldatalarge` because although it inherits from `celldataemphasis`, it is defined in the parent style definition, not the child definition. Nor is the change inherited by `celldatasmall`, which inherits its attributes from `celldatalarge` and from the parents of `celldatalarge`, which include `celldataemphasis` (as defined in the parent style definition) and `celldatasimple`.

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

But, suppose that you want to pass the white background from `celldataemphasis` on to `celldatalarge` even though it is defined only in the parent style definition? There are two ways to make this sort of change. The first is simply to redefine `celldatalarge` in the new style definition with a STYLE statement:

Example Code 5.12 Redefining a Style Element without Inheritance

```
proc template;
  define style concepts.style1;
    style celldatasimple /
      font_face=arial
      background=very light vivid blue
      foreground=white;
    style celldataemphasis from celldatasimple /
```

```

        foreground=blue
        font_style=italic;
    style celldatalarge from celldataemphasis /
        font_weight=bold
        font_size=5;
end;
run;

proc template;
    define style concepts.style2;
        parent=concepts.style1;
        style celldataemphasis from celldataemphasis /
            background=white;
        style celldatalarge from celldataemphasis /
            font_weight=bold
            font_size=5;
        style celldatasmall from celldatalarge /
            font_size=2;
    end;
run;

```

In this case, when PROC TEMPLATE processes the STYLE statement that creates **celldatalarge**, it looks for a style element named **celldataemphasis** to inherit from. Because there is such a style element in the child style definition, PROC TEMPLATE uses that style element. (If there were no such element in the new style definition, PROC TEMPLATE would look for one in the parent and use that one.) Therefore, **celldatalarge** inherits the new definition of **celldataemphasis**, which includes the white background. Similarly **celldatasmall**, which now inherits from **celldatalarge** in the child style definition, inherits the white background.

This method works fine for a few style elements. However, if a large number of style elements are inherited from **celldataemphasis**, it would be time-consuming to redefine all of them in the child style element.

Fortunately, there is a way to redefine **celldataemphasis** so that the changes are passed on to style elements that inherit from it. PROC TEMPLATE's flexibility allows you to choose whether you want to pass the new style attributes on to descendants or not.

If you want to make a change that “ripples” through to the style elements that are defined in the parent and that inherit from the style element that you redefine in the child style definition, use the REPLACE statement. You can only use the REPLACE statement if you have specified a parent style definition. The following program changes the background color of **celldataemphasis** with a REPLACE statement. You can think of this REPLACE statement as replacing the statement that defines the like-named style element in the parent style definition. The REPLACE statement doesn't actually change the parent style definition, but PROC TEMPLATE builds the child style definition as if it had changed the parent.

Example Code 5.13 Redefining a Style Element with the REPLACE Statement

```

proc template;
    define style concepts.style1;
        style celldatasimple /
            font_face=arial
            background=very light vivid blue
            foreground=white;
        style celldataemphasis from celldatasimple /

```

```

        foreground=blue
        font_style=italic;
    style celldatalarge from celldataemphasis /
        font_weight=bold
        font_size=5;
    end;
run;

proc template;
    define style concepts.style2;
        parent=concepts.style1;
        replace celldataemphasis from celldatasimple /
            foreground=blue
            font_style=italic
            background=white;
        style celldatasmall from celldatalarge /
            font_size=2;
    end;
run;

```

This is how PROC TEMPLATE constructs **concepts.style2**:

- 1 The PARENT= option makes **concepts.style1** the basis of the new style definition. The new style definition contains all the style elements that the parent contains: **celldatasimple**, **celldataemphasis**, and **celldatalarge**.
- 2 The new style definition does nothing to **celldatasimple**. Therefore, in **concepts.style2**, **celldatasimple** is the same as it is in **concepts.style1**.
- 3 The REPLACE statement essentially replaces the definition of **celldataemphasis** in **concepts.style1** while **concepts.style2** is being created. (It does not really alter **concepts.style1**, but **concepts.style2** is created as if it had.) Thus, not only does **celldataemphasis** now exist in **concepts.style2**, but also every style element that **concepts.style2** inherits from **concepts.style1** is based on the replaced definition.

A description of each style element in **concepts.style2** follows:

celldatasimple

is not redefined in **concepts.style2**. Nor does it inherit from any other style element. Therefore, it has the same attributes as **celldatasimple** in **concepts.style1** does:

- FONT_FACE=arial
- BACKGROUND=very light vivid blue
- FOREGROUND=white.

celldataemphasis

is defined in **concepts.style2**. It inherits from **celldatasimple**, so initially it has these attributes:

- FONT_FACE=arial
- BACKGROUND=very light vivid blue
- FOREGROUND=white.

However, the REPLACE statement that creates **celldataemphasis** specifies a foreground color, a background color, and a font style. The foreground and

background color specifications override the inherited attributes. Therefore, the final list of attributes for `celldataemphasis` is

- `FONT_FACE=arial`
- `BACKGROUND=white`
- `FOREGROUND=blue`
- `FONT_STYLE=italic.`

`celldatalarge`

is not redefined in `concepts.style2`. Therefore, `concepts.style2` uses the same definition as `concepts.style1` uses. The definition of `celldatalarge` is from `celldataemphasis`. Because `celldataemphasis` was created in `concepts.style2` with a `REPLACE` statement, `celldatalarge` inherits the following attributes from the replaced definition of `celldataemphasis`:

- `FONT_FACE=arial`
- `BACKGROUND=white`
- `FOREGROUND=blue`
- `FONT_STYLE=italic.`

The definition of `celldatalarge` adds these attributes:

- `FONT_WEIGHT=bold`
- `FONT_SIZE=5.`

`celldatasmall`

exists only in `concepts.style2`. It is created from `celldatalarge`. `PROC TEMPLATE` first looks for `celldatalarge` in `concepts.style2`, but because it doesn't exist, it uses the definition in the parent style definition. `celldatasmall` is, therefore, just like `celldatalarge` except that the font size of 2 replaces the font size of 5.

The following HTML output uses this new style definition:

Display 5.9 Using the `REPLACE` Statement to Alter a Style Element and Its Children

Country	Grain	Kilotons	Kilotons
Brazil	Rice	<i>10035</i>	10035
China	Rice	<i>190100</i>	190100
India	Rice	<i>120012</i>	120012
Indonesia	Rice	<i>51165</i>	51165
United States	Rice	<i>7771</i>	7771

The following points summarize style-element inheritance in a style definition that has a parent:

- You can create a new style element from any style element in the parent or the child style definition.
- If you create a style element from another style element, PROC TEMPLATE first looks in the current style definition for that element. If the style definition does not contain such an element, PROC TEMPLATE looks in the parent (and in parent's parent, and so forth).
- A new style element inherits all the attributes from its parent.
- You can specify additional attributes in the new style definition. The attributes are added to the attributes that the element inherits.
- You can change the value of an inherited attribute by respecifying it in the definition of the new style element.
- If you use the STYLE statement to create a style element in the new style definition, only style elements that explicitly inherit from that style element in the new style definition inherit the change. Style elements that are not explicitly redefined in the new style definition inherit from the style element definition that is in the parent style definition.
- If you use the REPLACE statement to create a style element in the new style definition, all style elements that inherit from that element inherit the definition that is in the new style definition. If you want to keep any attributes that are specified in the definition that is in the parent, you must respecify them in the definition that you create in the child style definition.

Procedure Syntax

Availability: The LIST statement and the STORE= option are available in Version 8 of the SAS System

PROC TEMPLATE;

```
DEFINE definition-type definition-path < / STORE=libname.template-store>;
      statements-and-attributes
END;
```

```
DELETE definition-path < / STORE=libname.template-store >;
```

```
EDIT definition-path-1 <AS definition-path-2> < / STORE=libname.template-store > ;
      statements-and-attributes
END;
```

```
LINK definition-path-1 definition-path-2 </ <NOTES ='text'
      <STORE=libname.template-store >>>;
```

```
LIST <starting-path></ option(s)>;
```

```
PATH location(s);
```

```
SOURCE definition-path </ FILE='file-specification'><STORE=libref.template-store>;
```

```
TEST DATA=SAS-data-set< / STORE=libname.template-store>;
```

To do this ...	Use this statement
Create a definition for a table, column, header, footer, or style	DEFINE
Delete the specified definitions	DELETE
Edit an existing definition	EDIT
Create a link to an existing definition	LINK
List items in one or more template stores	LIST
Specify which locations to search for definitions, as well as the order in which to search for them	PATH
View the source code for the specified definition	SOURCE
Test the most recently created definition by binding it to the specified data set	TEST

PROC TEMPLATE Statement

PROC TEMPLATE;

DEFINE Statement

Creates a definition for a table, column, header, footer, or style.

Requirement: An END statement must be the last statement in the definition.

Interaction: In some cases, you can use a DEFINE statement inside a definition.

A table definition can contain one or more column, header, or footer definitions.

A column definition can include one or more header definitions.

See also: “DEFINE COLUMN Statement” on page 159, “DEFINE HEADER Statement” on page 177, “DEFINE STYLE Statement” on page 188, and “DEFINE TABLE Statement” on page 209

```
DEFINE definition-type definition-path< / STORE=libname.template-store>;
    statements-and-attributes
END;
```

Required Arguments

definition-type

specifies the type of definition to create, where *definition-type* is one of the following:

COLUMN

FOOTER

HEADER

STYLE

TABLE

The *definition-type* determines what other statements and what attributes can go in the definition. For details, see the documentation for the corresponding DEFINE statement.

definition-path

specifies where to store the definition. A definition-path consists of one or more names, separated by periods. Each name represents a directory, or level, in a template store. (A template store is a type of SAS file.) PROC TEMPLATE writes the definition to the first template store that you can write to in the current template-store path.

Restriction: If the definition is nested inside another definition, *definition-path* must be a single-level name.

Restriction: If you want to reference the definition that you are creating from another definition, do not nest the definition inside another one. For example, if you want to reference a header definition from multiple columns, do not define the header inside a column definition.

See also: For information on setting the current path, see “PATH Statement” on page 231.

Options**STORE=***libname.template-store*

specifies the template store in which to store the definition. If the template store does not exist, it is created.

Restriction: If the definition is nested inside another definition, you cannot use the STORE= option.

Availability: Version 8 of the SAS System

END Statement

Ends the definition.

END;

DEFINE COLUMN Statement

Creates a definition for a column.

Requirement: An END statement must be the last statement in the definition.

Interaction: A column definition can include one or more header definitions.

See also: “DEFINE HEADER Statement” on page 177

Featured in: Example 2 on page 238, Example 3 on page 247, and Example 4 on page 252

```

DEFINE COLUMN column-path < / STORE=libname.template-store>;
  < column-attribute-1; <... column-attribute-n; >>
  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 definition-path;
    statements-and-attributes
  END;
  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';
  TRANSLATE expression-1 INTO expression-2 <..., expression-n INTO
    expression-m>;
  END;

```

To do this ...	Use this statement
Set one or more column attributes.	<i>column-attributes</i>
Set the style element of the cells in the column according to the values of the variables.	CELLSTYLE-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.	COMPUTE AS
Create a definition for a column header.	DEFINE HEADER
Define a symbol that references a value that the data component supplies from the procedure or DATA step.	DYNAMIC
Define a symbol that references a macro variable. ODS will use the variable as a string. References to the macro variable are resolved when ODS binds the definition and the data component to produce an output object.	MVAR
Define a symbol that references a macro variable. ODS will convert 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 definition and the data component to produce an output object.	NMVAR

To do this ...	Use this statement
Provide information about the column.	NOTES
Translate the specified values to other values.	TRANSLATE-INTO
End the definition.	END

Required Arguments

column-path

specifies where to store the column definition. A *column-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 definition to the first template store that you can write to in the current path.

Restriction: If the definition is nested inside another definition, *definition-path* must be a single-level name.

Restriction: If you want to reference the definition that you are creating from another definition, do not nest the definition inside another one. For example, if you want to reference a column definition from multiple tables, do not define the column inside a table definition.

Options

STORE=*libname.template-store*

specifies the template store in which to store the definition. If the template store does not exist, it is created.

Restriction: If the definition is nested inside another definition, you cannot use the STORE= option.

Availability: Version 8 of the SAS System

Column Attributes

This section lists all the attributes that you can use in a column definition. For all attributes that support a value of ON, the following forms are equivalent:

```
ATTRIBUTE-NAME
ATTRIBUTE-NAME=ON
```

For all attributes that support a value of *variable*, *variable* can be any variable that you declare in the column definition with the DYNAMIC, MVAR, or NMVAR statement. If the attribute is a boolean, the value of *variable* should resolve to one of the following:

ON	YES	0
ON	_YES_	FALSE
1	OFF	NO
TRUE	_OFF_	_NO_

To do this ... *	Use this attribute
Influence the appearance of the contents of the cells	
Specify whether or not to suppress the value of a variable from one row to the next if the value does not change.	BLANK_DUPS=
Specify whether or not to wrap the text in the current column if it is too long to fit in the space that is provided.	FLOW=
Specify the format for the column.	FORMAT=
Specify the format width for the column if it isn't specified with FORMAT=.	FORMAT_WIDTH=
Specify the number of decimals for the column if it isn't specified with FORMAT=.	FORMAT_NDEC=
Supply a numeric value against which values in the column are compared to eliminate trivial values (absolute values less than the FUZZ= value) from computation and printing.	FUZZ=
Specify the horizontal justification of the format field within the column (and for the header if the definition for the header does not include JUST=).	JUST=
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=
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 column uses more than one line.	MAXIMIZE=
Specify whether or not 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).	OVERLINE=
Specify whether or not to treat the text as preformatted text.	PREFORMATTED=
Specify whether or not to print the column.	PRINT=
Specify a separator character to append to each value in the column.	SEPARATOR=
Specify the style element and style attributes to use for the column.	STYLE=
Specify the split character for the data in the column.	TEXT_SPLIT=
Specify whether or not 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).	UNDERLINE=
Specify the vertical justification for the column.	VJUST=

To do this ... *	Use this attribute
Specify the width of the column in characters.	WIDTH=
Specify the maximum width allowed for this column.	WIDTH_MAX=
Customize column headers	
Specify the text for the column header.	HEADER=
Specify whether or not to print the column header.	PRINT_HEADERS=
Influence the relationship to other columns	
Specify whether or not the column definition is generic — that is, whether or not it can be used by more than one variable.	GENERIC=
Specify whether or not the column is an ID column.	ID=
Specify whether or not to merge the current column with the column immediately to its right.	MERGE=
Specify whether or not to merge the current column with the column immediately to its left.	PRE_MERGE=
Specify the number of blank characters to leave between the current column and the column immediately to its left.	PRE_SPACE=
Specify the number of blank characters to leave between the current column and the column immediately to its right.	SPACE=
Influence the presentation of data panels	
Influence the place at which ODS splits a table when it creates multiple data panels.	GLUE=
Specify whether or not 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=
Specify the name of the column in the data component to associate with the current column.	DATANAME=
Specify a label for the column.	LABEL=
Specify the column definition that the current definition inherits from.	PARENT=
Specify the name to use for the corresponding variable in an output data set.	VARNAME=
Specify whether or not to include the column in an output data set.	DROP=
Specify which format to use if both a column definition and a data component specify one.	DATA_FORMAT_OVERRIDE=

* Different attributes affect different ODS destinations. For details, consult the documentation for a specific attribute.

BLANK_DUPS \leq ON | OFF | *variable*

specifies whether or not to suppress the value of a variable from one row to the next if the value does not change.

Default: OFF

Interaction: If the CLASSLEVELS= table attribute on page 213 is in effect, ODS ignores BLANK_DUPS=ON when any value changes in a preceding column that is also marked with BLANK_DUPS=ON.

ODS Destinations: All but Output. Note that when the Printer destination suppresses the value of a variable, it also suppresses the horizontal rule above the blank cell.

Featured in: Example 3 on page 247 and Example 4 on page 252

DATA_FORMAT_OVERRIDE \leq ON | OFF | *variable*

specifies which format to use if both a column definition and a data component specify one.

ON

Uses the format in the data component.

OFF

Uses the format in the column definition.

Default: OFF

ODS Destinations: All

Availability: Version 8 of the SAS System

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.

ODS Destinations: All

DROP \leq ON | OFF | *variable*

specifies whether or not to include the column in an output data set.

Default: OFF

ODS Destinations: Output

Availability: Version 8 of the SAS System

FLOW \leq ON | OFF | *variable*

specifies whether or not 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.

See also: MAXIMIZE= on page 166

ODS Destinations: Listing

Note: The HTML and Printer destinations always wrap the text if it is too long to fit in the space that is provided. Δ

FORMAT=*format-name* \langle *format-width* \langle *decimal-width* \rangle \rangle | *variable*

specifies the format for the column.

Default: If you don't specify FORMAT=, PROC TEMPLATE uses the format that the data component provides. If the data component does not provide a format, PROC TEMPLATE uses

- best8. for integers

- 12.3 for doubles
- the length of the variable for character variables.

If the format is provided by the data component and if the format includes a format width or a decimal width, PROC TEMPLATE uses the specified format. However, if `FORMAT_WIDTH=`, `FORMAT_NDEC=`, or both are used in the column definition, PROC TEMPLATE uses those values instead of the ones that the data component provided with its format name.

Restriction: If you specify a format width for a numeric column, its value cannot exceed 32.

Interaction: If you specify a format name but do not specify a format width or a decimal width with the format name, PROC TEMPLATE uses the values that are specified by `FORMAT_WIDTH=` and `FORMAT_NDEC=`. If these options aren't specified, PROC TEMPLATE uses the values for format width and decimal width that the data component provides.

ODS Destinations: All

FORMAT_WIDTH=*positive-integer* | *variable*

specifies the format width for the column.

Default: the format width that is specified with `FORMAT=`

Range: 1 to 32 for numeric variables; operating system limit for character variables

Interaction: If you specify a format width with `FORMAT=` and with `FORMAT_WIDTH=`, PROC TEMPLATE uses the one that you specify with `FORMAT_WIDTH=`.

ODS Destinations: All

FORMAT_NDEC=*positive-integer* | *variable*

specifies the number of decimals for the column.

Default: the decimal width that is specified with `FORMAT=`

Interaction: If you specify a decimal width with `FORMAT=` and with `FORMAT_NDEC=`, PROC TEMPLATE uses the one that you specify with `FORMAT_NDEC=`.

ODS Destinations: All

FUZZ=*number* | *variable*

supplies a numeric value against which values in the column are compared to eliminate trivial values (absolute values less than the `FUZZ=` value) from computation and printing. A number whose absolute value is less than the `FUZZ=` value is treated as zero in computations and printing.

Default: the smallest representable floating-point number on the computer that you are using

ODS Destinations: All but Output

GENERIC`<=ON | OFF | variable>`

specifies whether or not the column definition is generic – that is, whether or not it can be used by more than one column. Generic columns are useful in tables with many similar columns. For instance, the table definitions for both PROC SQL and the DATA step define only two columns: one for character variables and one for numeric variables. When the program runs, it determines which column definition the data component should use for each column.

Default: OFF

ODS Destinations: All but Output

Featured in: Example 2 on page 238, Example 3 on page 247, and Example 4 on page 252

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

Tip: A value of -1 forces the table to split between the current column and the column to its right.

ODS Destinations: Listing and Printer

HEADER=*header-specification*

specifies the text for the column header. *header-specification* can be one of the following:

text

Provides the actual text of the header.

header-name

specifies the name of a header definition to use. You create a header definition with the DEFINE HEADER statement (see “DEFINE HEADER Statement” on page 177). If *header-name* is a single-level name, the header definition must occur within the current column definition.

variable

specifies the name of a variable that you declare 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

Tip: The HEADER= option provides a simple way for you to specify the text of a column header. If you want to customize the header further, use the DEFINE HEADER statement with the appropriate header attributes. (See “DEFINE HEADER Statement” on page 177.)

Tip: You can use the split character in the text of the header to force the text to a new line.

See also: LABEL= on page 166 and TEXT_SPLIT= on page 170

ODS Destinations: All

Note: If you are using the Output destination, the column header becomes the label of the corresponding variable in the output data set if you do not specify one with the LABEL= attribute and if the data component does not supply one. Δ

Featured in: Example 2 on page 238 and Example 4 on page 252

ID<=ON | OFF | *variable*>

specifies whether or not 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

Tip: ODS treats all columns up to and including a column that is marked with ID=ON as ID columns.

ODS Destinations: Listing and Printer

Featured in: Example 2 on page 238

JUST=*justification* | *variable*

specifies the horizontal justification of the format field within the column (and of the header if the definition for the header does not include JUST=). For a discussion of how the Listing destination justifies data, see “How Are Values in Table Columns Justified?” on page 138. *justification* can be one of the following:

Left

specifies left justification.

Alias: L

Right

specifies right justification.

Alias: R

Center

specifies center justification.

Alias: C

Default: LEFT for columns that contain character values; RIGHT for columns that contain numeric values.

Interaction: For the Listing destination, ODS justifies the format field within the column width. At times, you may need to specify the JUSTIFY= attribute to get the results that you want. See the discussion of JUSTIFY= on page 166.

See also: FORMAT= on page 163 and WIDTH= on page 170

ODS Destinations: All but Output

Featured in: Example 1 on page 233

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. For a discussion of how ODS destinations justify data, see “How Are Values in Table Columns Justified?” on page 138.

Default: OFF

Interaction: JUSTIFY=ON can interfere with decimal alignment.

Tip: If you translate numeric data to character data, you may need to use JUSTIFY= to align the data as you wish.

Featured in: Example 3 on page 247

ODS Destinations: Listing (The HTML and Printer destinations always behave as if JUSTIFY=ON.)

LABEL=*'text'* | *variable*

specifies a label for the column in the output data set.

Default: If you do not specify 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.

ODS Destinations: Output

Tip: If the Output destination is open, LABEL= 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 might 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 would result in 33% of the text on each line. MAXIMIZE=ON may write lines of text that vary greatly in length. MAXIMIZE=OFF may 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 FLOW= on page 163).

ODS Destinations: Listing

MERGE<=ON | OFF | *variable*>

specifies whether or not 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 definition. 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.

See also: PRE_MERGE= on page 168

ODS Destinations: All but Output

OPTIONAL<=ON | OFF | *variable*>

specifies whether or not 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 definitions contain OPTIONAL=ON, PROC TEMPLATE includes either all or none of these columns in the output object.

ODS Destinations: Listing

OVERLINE<=ON | OFF | *variable*>

specifies whether or not 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). PROC TEMPLATE uses the second formatting character to draw the line. (See the discussion of FORMCHAR= on page 215.)

Default: OFF

ODS Destinations: Listing

PARENT=*column-path*

specifies the column definition that the current definition inherits from. A *column-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 definition inherits from the the specified column in the first template store that you can read from in the current path.

When you specify a parent, all the attributes and statements that are specified in the parent's definition are used in the current definition unless the current definition specifically overrides them.

ODS Destinations: All

PREFORMATTED<=ON | OFF | *variable*>

specifies whether or not 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 renders 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.

ODS Destinations: HTML and Printer

PRE_MERGE<=ON | OFF | *variable*>

specifies whether or not 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 definition. 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.

See also: MERGE= on page 167

ODS Destinations: All but Output

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=.

See also: SPACE= on page 169, COL_SPACE_MIN= on page 213, and COL_SPACE_MAX= on page 213

ODS Destinations: Listing

PRINT<=ON | OFF | *variable*>

specifies whether or not to print the column.

Default: ON

See also: OPTIONAL= on page 167 and DROP= on page 163

ODS Destinations: All but Output

PRINT_HEADERS<=ON | OFF | *variable*>

specifies whether or not to print the column header and any underlining and overlining.

Default: ON

See also: UNDERLINE= on page 170 and OVERLINE= on page 167

ODS Destinations: All but Output

SEPARATOR='character' | *variable*

specifies a separator character to append to each value in the column.

Default: None

Tip: To specify a hexadecimal character as the separator character, put an x after the closing quote. For instance, the following option assigns the hexadecimal character 2D as the separator character:

```
separator='2D'x
```

ODS Destinations: Listing and Printer

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=.

See also: PRE_SPACE= on page 168, COL_SPACE_MIN= on page 213, and COL_SPACE_MAX= on page 213

ODS Destinations: Listing

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]). Δ

style-element-name

is the name of the style element to use to render the data in the column. The style element must be part of a style definition that is registered with the Output Delivery System. SAS Institute provides some style definitions. Users can create their own style definitions with PROC TEMPLATE (see “DEFINE STYLE Statement” on page 188). By default, ODS renders different parts of ODS output with different style elements. For instance, by default, the data in a column is rendered with the style element `data`. The style elements that you would be most likely to use with the STYLE= column attribute are

- data
- datafixed
- dataempty
- dataemphasis
- dataemphasisfixed
- datastrong
- datastrongfixed.

The style element provides the basis for rendering the column. Additional style attributes that you provide can modify the rendering.

For information on finding an up-to-date list of the style definitions and for viewing a style definition so that you can see the style elements that are available, see “Customizing Presentation Aspects of ODS Output” on page 42. For information about the default style definition that ODS uses, see “What Is the Default Style Definition Like?” on page 133.

style-element-name can be either the name of a style element or a variable whose value is a style element.

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 on the style attributes that you can specify, see “Style Attributes” on page 192.

ODS Destinations: HTML and Printer

Featured in: Example 2 on page 238 and Example 4 on page 252

TEXT_SPLIT=*character* | *variable*

specifies the split character for the data in the column. PROC TEMPLATE breaks a value in the column 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

ODS Destinations: All but Output

UNDERLINE<=ON | OFF | *variable*>

specifies whether or not 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). PROC TEMPLATE uses the second formatting character to draw the line. (See the discussion of FORMCHAR= on page 215.)

Default: OFF

ODS Destinations: Listing

VARNAME=*variable-name* | *variable*

specifies the name to use for the corresponding variable in an output data set.

Default: If you do not specify VARNAME=, PROC TEMPLATE uses the the value of the DATANAME= attribute. If you do not specify DATANAME=, PROC TEMPLATE uses the name of the column.

Tip: 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.

ODS Destinations: Output

VJUST=*justification* | *variable*

Specifies the vertical justification for the column. *justification* can be 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 for the Printer destination; CENTER for the HTML destination

ODS Destinations: HTML and Printer

Featured in: Example 2 on page 238

WIDTH=*positive-integer* | *variable*

specifies the width of the column in characters.

Default: If you do not specify a width, PROC TEMPLATE uses the format width. If the column has no format associated with it, PROC TEMPLATE uses a width of

- 8 for integers
- 12 for doubles
- data length for character variables.

Interaction: The length of the column header can influence the width of the column.

See also: WIDTH_MAX on page 171 and WIDTH= header attribute on page 185

ODS Destinations: Listing

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

ODS Destinations: Listing

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, flyover) of individual cells.

Featured in: Example 3 on page 247

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 cell in the column. It can be any expression that is valid in the WHERE statement (or the WHERE= data set option). For information on expressions that you can use in the WHERE statement, see “Statements” in *SAS Language Reference: Dictionary*. Use *_VAL_* to represent the value of the current column. You may also reference symbols that you declared in a DYNAMIC, MVAR, or NMVAR statement in the definition.

If *expression* resolves to TRUE (a non-zero 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.

Restriction: You may not reference the values of other columns in *expression*.

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.

Options

Note: Neither *style-attribute-specification* nor *style-element-name* is required. However, you must use at least one of them. Δ

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 on the style attributes that you can set in a column definition, see “Style Attributes” on page 192.

Default: If you don’t specify any style attributes to modify, ODS uses the unmodified *style-element-name*.

style-element-name

is the name of the style element to use to render the data in the column. The style element must be the name of a style element that is part of a style definition that is registered with the Output Delivery System. SAS Institute provides some style definitions. Users can create their own style definitions with PROC TEMPLATE (see “DEFINE STYLE Statement” on page 188). By default, ODS renders different parts of ODS output with different style elements. For instance, by default, the data in a column is rendered with the style element `data`. The style elements that you would be most likely to use with the CELLSTYLE-AS statement in a column definition are

- `data`
- `datafixed`
- `dataempty`
- `dataemphasis`
- `dataemphasisfixed`
- `datastrong`
- `datastrongfixed`.

The style element provides the basis for rendering the column. Additional style attributes that you provide can modify the rendering.

For information on finding an up-to-date list of the style definitions and for viewing a style definition so that you can see the style elements that are available, see “Customizing Presentation Aspects of ODS Output” on page 42. For information about the default style definition that ODS uses, see “What Is the Default Style Definition Like?” on page 133.

Default: `data`

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.

COMPUTE AS *expression*;

Required Arguments

expression

is an expression that assigns a value to each table cell in the column. It can be any expression that is valid in the WHERE statement (or the WHERE= data set option). For information on expressions that you can use in the WHERE statement, see “Statements” in *SAS Language Reference: Dictionary*.

To reference another column in a COMPUTE-AS statement, use the name of the column. You may also reference symbols that you declared in a DYNAMIC, MVAR, or

NVAR statement in the current definition. In addition, if the column has values in the data component, you may reference the column itself in the expression. However, if you are creating a column that does not exist in the data component, you cannot reference the column in the expression because there is no underlying value to use.

For example, the following 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;
```

Tip: The COMPUTE AS statement can alter values in an output object. None of the definitions that SAS Institute provides modifies any values. If you want to determine if a definition was provided by SAS Institute, use the ODS VERIFY statement (see “ODS VERIFY Statement” on page 72). If the definition is not from SAS Institute, the ODS VERIFY statement returns a warning when it runs the SAS program that uses the definition. If you receive such a warning, you can use the SOURCE statement to look at the definition and determine if the COMPUTE AS statement is used to alter values. (See “SOURCE Statement” on page 232.)

Tip: Because you can use column names in *expression*, *_VAL_* is not recognized as an alias for the current column.

DEFINE HEADER Statement

Creates a definition for a header inside a column definition.

Main discussion: “DEFINE HEADER Statement” on page 177

```
DEFINE HEADER definition-name ;
  statements-and-attributes
END;
```

Required Arguments

definition-name

specifies the name of the new header.

Restriction: *definition-name* must be a single-level name.

Note: If you want to reference the header definition that you are creating from another definition, you must create it outside the column definition. Δ

DYNAMIC Statement

Defines a symbol that references a value that the data component supplies from the procedure or DATA step.

Scope: You can use the DYNAMIC statement in the definition of a table, column, header, or footer. A dynamic variable that is defined in a definition is available to that definition and to all the definitions that it contains.

Main discussion: “DYNAMIC Statement” on page 222

DYNAMIC *variable-1* <'text-1'> <... *variable-n* <'text-n'>>;

MVAR Statement

Defines a symbol that references a macro variable. ODS will use the value of the variable as a string. References to the macro variable are resolved when ODS binds the definition and the data component to produce an output object.

Scope: You can use the MVAR statement in the definition of a table, column, header, or footer. A macro variable that is defined in a definition is available to that definition and to all the definitions that it contains.

Main discussion: “MVAR Statement” on page 224

MVAR *variable-1* <'text-1'> <... *variable-n* <'text-n'>>;

NMVAR Statement

Defines a symbol that references a macro variable. ODS will convert 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 definition and the data component to produce an output object.

Scope: You can use the NMVAR statement in the definition of a table, column, header, or footer. A macro variable that is defined in a definition is available to that definition and to all the definitions that it contains.

Main discussion: “NMVAR Statement” on page 225

NMVAR *variable-1* <'text-1'> <... *variable-n* <'text-n'>>;

NOTES Statement

Provides information about the column.

Tip: The NOTES statement becomes part of the compiled column definition, which you can view with the SOURCE statement, whereas SAS comments do not.

NOTES *'text'*;

Required Arguments

text

provides information about the column.

TRANSLATE-INTO Statement

Translates the specified values to other values.

TRANSLATE *expression-1 INTO expression-2 <...>, expression-n INTO expression-m*;

Required Arguments

expression-1

is an expression that is evaluated for each table cell in the column. It can be any expression that is valid in the WHERE statement (or the WHERE= data set option). For information on expressions that you can use in the WHERE statement, see “Statements” in *SAS Language Reference: Dictionary*. Use `_VAL_` to represent the value of the current column. You may also reference symbols that you declared in a DYNAMIC, MVAR, or NVAR statement in the table definition.

If *expression-1* resolves to TRUE (a non-zero 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.

Restriction: You may not 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.

expression-2

is an expression that specifies the value to use in the cell in place of the variable's actual value. It can be any expression that is valid in the WHERE statement (or the WHERE= data set option). For information on expressions that you can use in the

WHERE statement, see “Statements” in *SAS Language Reference: Dictionary*. Use `_VAL_` to represent the value of the current column. You may also reference symbols that you declared in a DYNAMIC, MVAR, or NVAR statement in the table definition.

Restriction: *expression-2* must resolve to a character value, not a numeric value.

Restriction: You may not reference the values of other columns in *expression-2*.

Tip: When you translate a numeric value to a character value, the column definition does not try to apply the numeric format that is associated with the column.

Instead, it simply writes the character value into the format field, starting at the left. If you want the value to be right justified, use the JUSTIFY=ON attribute.

See also: JUSTIFY= on page 166

END Statement

Ends the definition.

END;

DEFINE FOOTER Statement

Creates a definition for a table footer.

Requirement: An END statement must be the last statement in the definition.

Featured in: Example 2 on page 238 and Example 4 on page 252

See: “DEFINE HEADER Statement” on page 177

DEFINE FOOTER *footer-path* < / STORE=*libname.template-store*>;

< *footer-attribute-1*; <... *footer-attribute-n*; >>

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*';

TEXT *footer-specification*;

TEXT2 *footer-specification*;

TEXT3 *footer-specification*;

END;

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 “DEFINE HEADER Statement” on page 177.

DEFINE HEADER Statement

Creates a definition for a header.

Requirement: An END statement must be the last statement in the definition.

Featured in: Example 2 on page 238

```
DEFINE HEADER header-path </ STORE=libname.template-store>;
  <header-attribute-1; <... header-attribute-n; >>
  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';
  TEXT header-specification;
  TEXT2 header-specification;
  TEXT3 header-specification;
END;
```

To do this ...	Use this statement
Set one or more header attributes.	<i>header-attribute(s)</i>
Define a symbol that references a value that the data component supplies from the procedure or DATA step.	DYNAMIC
Define a symbol that references a macro variable. ODS will use the value of the variable as a string. References to the macro variable are resolved when ODS binds the definition and the data component to produce an output object.	MVAR
Define a symbol that references a macro variable. ODS will convert 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 definition and the data component to produce an output object.	NMVAR
Provide information about the table.	NOTES
Specify the text of the header.	TEXT
Specify an alternative header to use in the Listing output if the header that is provided by the TEXT statement is too long.	TEXT2
Specify an alternative header to use in the Listing output if the header that is provided by the TEXT2 statement is too long.	TEXT3
End the header definition.	END

Required Arguments

header-path

specifies where to store the header definition. 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 definition to the first template store that you can write to in the current path.

Restriction: If the definition is nested inside another definition, *definition-path* must be a single-level name.

Restriction: If you want to reference the definition that you are creating from another definition, do not nest the definition inside another one. For example, if you want to reference a header definition from multiple columns, do not define the header inside a column definition.

Options

STORE=*libname.template-store*

specifies the template store in which to store the definition. If the template store does not exist, it is created.

Restriction: If the definition is nested inside another definition, you cannot use the STORE= option.

Availability: Version 8 of the SAS System

Header Attributes

This section lists all the attributes that you can use in a header definition. A column header spans a single column. A spanning header spans multiple columns. These two kinds 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, the following forms are equivalent:

```
ATTRIBUTE-NAME
ATTRIBUTE-NAME=ON
```

For all attributes that support a value of *variable*, *variable* can be any variable that you declare in the table definition with the DYNAMIC, MVAR, or NMVAR statement. If the attribute is a boolean, the value of *variable* should resolve to one of the following:

ON	YES	0
ON	_YES_	FALSE
1	OFF	NO
TRUE	_OFF_	_NO_

To do this ...*	Use this attribute
Influence the appearance of the contents of the header	
Specify whether or not to try to expand the column width to accommodate the longest word in the column header.	FORCE=
Specify the horizontal justification for the column header.	JUST=
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=
Specify whether or not to draw a continuous line above the header.	OVERLINE=
Specify whether or not to treat the text as preformatted text.	PREFORMATTED=
Specify whether or not to print the header.	PRINT=
Specify the number of blank lines to place between the current header and the next one or between the current footer and the previous one.	SPACE=
Specify the split character for the header.	SPLIT=
Specify the style element and any changes to its attributes to use for the header.	STYLE=
Specify whether or not to start a new line of the header in the middle of a word.	TRUNCATE=
Specify whether or not to draw a continuous line underneath the header.	UNDERLINE=
Specify vertical justification for the header.	VJUST=
Specify the width of the header in characters.	WIDTH=
Influence the content of the header	
Specify a character to use to expand the header to fill the space over the column or columns that the header spans.	EXPAND=
Specify whether or not to repeat the text of the header until the space that is allotted for the header is filled.	REPEAT=
Influence the placement of the header	
Specify the last column that a spanning header covers.	END=
Specify the first column that a spanning header covers.	START=
Specify whether or not to expand the header to reach the sides of the page.	EXPAND_PAGE=
Specify whether or not 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=

To do this ...*	Use this attribute
Specify whether or not 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
Specify whether or not to extend the text of the header into the header space of adjacent columns.	SPILL_ADJ=
Specify whether or not to extend the text of the header into the adjacent margin.	SPILL_MARGIN
Specify whether or not multiple columns can use the header.	GENERIC=
Specify the header definition that the current definition inherits from.	PARENT=

* Different attributes affect different ODS destinations. For details, consult the documentation for a specific attribute.

END=*column-name* | *variable*

specifies the last column that a spanning header covers.

Default: the last column

See also: START= on page 183

ODS Destinations: All but Output

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 REPEAT=ON and EXPAND=ON, PROC TEMPLATE honors EXPAND=.

See also: REPEAT= on page 182

Tip: If the string or the variable that you specify contains more than one character, PROC TEMPLATE uses only the first character.

See also: EXPAND_PAGE= on page 180

ODS Destinations: Listing

EXPAND_PAGE<= ON | OFF | *variable*>

specifies whether or not to expand the header to reach the sides of the page.

Default: OFF

See also: EXPAND= on page 180

ODS Destinations: Listing

FIRST_PANEL<= ON | OFF | *variable*>

specifies whether or not 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

See also: LAST_PANEL= on page 181

ODS Destinations: Listing and Printer

FORCE \leq ON | OFF | *variable*

specifies whether or not 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

See also: WIDTH= on page 185 and WIDTH_MAX= on page 171

ODS Destinations: Listing

GENERIC \leq ON | OFF | *variable*

specifies whether or not multiple columns can use the header.

Default: OFF

Restriction: This attribute is primarily for writers of SAS procedures and for DATA step programmers.

ODS Destinations: All but Output

JUST=*justification* | *variable*

specifies the horizontal justification for the column header, where *justification* can be 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

ODS Destinations: All but Output

Featured in: Example 1 on page 233

LAST_PANEL \leq ON | OFF | *variable*

specifies whether or not 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

See also: FIRST_PANEL on page 180

ODS Destinations: Listing and Printer

MAXIMIZE \leq 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 might 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 would may result in 33% of the text on each line.

MAXIMIZE=ON may write lines of text that vary greatly in length.

MAXIMIZE=OFF may result in using less than the full column width.

Default: OFF

ODS Destinations: Listing

OVERLINE \langle =ON | OFF | *variable* \rangle

specifies whether or not to draw a continuous line above the header. PROC TEMPLATE uses the second formatting character to draw the line. (See the discussion of FORMCHAR= on page 215.)

Default: OFF

ODS Destinations: Listing

PARENT=*header-path*

specifies the header definition that the current definition 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 definition inherits from the the specified header definition in the first template store that you can read from in the current path.

When you specify a parent, all the attributes and statements that are specified in the parent's definition are used in the current definition unless the current definition specifically overrides them.

ODS Destinations: All

PREFORMATTED \langle =ON | OFF | *variable* \rangle

specifies whether or not 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 renders the text in a monospace font.

Default: OFF

Interaction: 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.

ODS Destinations: HTML and Printer

PRINT \langle =ON | OFF | *variable* \rangle

specifies whether or not to print the header.

Default: ON

Tip: 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 definition nor the data component provides a label.

ODS Destinations: All

REPEAT \langle =ON | OFF | *variable* \rangle

specifies whether or not 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 REPEAT=ON and EXPAND=ON, PROC TEMPLATE honors EXPAND=.

See also: EXPAND= on page 180

ODS Destinations: Listing

SPACE=*positive-integer* | *variable*

specifies the number of blank lines to place between the current header and the next one or between the current footer and the previous one.

Default: 0

Tip: A row of underlining or overlining is considered a header or a footer.

ODS Destinations: Listing

Featured in: Example 2 on page 238

SPILL_ADJ<=ON | OFF | *variable*>

specifies whether or not 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, PROC TEMPLATE honors only one. FORCE= takes precedence over the other three attributes, followed by SPILL_MARGIN= and SPILL_ADJ=.

See also: FORCE= on page 181, SPILL_MARGIN= on page 183, and TRUNCATE= on page 185

ODS Destinations: Listing

SPILL_MARGIN<=ON | OFF | *variable*>

specifies whether or not to extend the text of the header into the adjacent margin.

Default: OFF

Restriction: SPILL_MARGIN= applies only to a spanning header that spans all the columns in a data panel.

Interaction: FORCE=, SPILL_MARGIN=, SPILL_ADJ=, and TRUNCATE= are mutually exclusive. If you specify more than one, PROC TEMPLATE honors only one. FORCE= takes precedence over the other three attributes, followed by SPILL_MARGIN= and SPILL_ADJ=.

See also: FORCE= on page 181, SPILL_ADJ on page 183, and TRUNCATE= on page 185

ODS Destinations: Listing

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.

Tip: The first character in a header is automatically treated as a split character if it is not one of the following:

- an alphanumeric character
- a blank
- an underscore (`_`)
- a hyphen (`-`)
- a period (`.`)
- a percent sign (`%`).

ODS Destinations: All but Output

START=*column-name* | *variable*

specifies the first column that a spanning header covers.

Default: the first column

See also: END= on page 180

ODS Destinations: All but Output

STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>

specifies the style element and any changes to its attributes to use for the header. 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]). Δ

style-element-name

is the name of the style element to use to render the header. The style element must be part of a style definition that is registered with the Output Delivery System. SAS Institute provides some style definitions. Users can create their own style definitions with PROC TEMPLATE (see “DEFINE STYLE Statement” on page 188). By default, ODS renders different parts of ODS output with different elements. For instance, by default, a table header is rendered 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

- 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

- 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

- rowheader
- rowheaderfixed
- rowheaderempty
- rowheaderemphasis
- rowheaderemphasisfixed
- rowheaderstrong
- rowheaderstrongfixed.

The style element provides the basis for rendering the header. Additional style attributes that you provide can modify the rendering.

For information on finding an up-to-date list of the style definitions and for viewing a style definition so that you can see the style elements that are available, see “Customizing Presentation Aspects of ODS Output” on page 42. For information about the default style definition that ODS uses, see “What Is the Default Style Definition Like?” on page 133.

style-element-name can be either the name of a style element or a variable whose value is a style element.

Default: header

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 on the style attributes that you can specify, see “Style Attributes” on page 192.

ODS destinations: HTML and Printer

Featured in: Example 1 on page 233 and Example 2 on page 238

TRUNCATE<=ON | OFF | *variable*>

specifies whether or not to start a new line of the header 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.

Note: TRUNCATE=OFF is the same as FORCE=ON. Δ

Default: OFF

Interaction: If you specify FORCE=, SPILL_MARGIN=, or SPILL_ADJ=, the TRUNCATE= attribute is ignored.

See also: FORCE= on page 181, SPILL_MARGIN= on page 183, and SPILL_ADJ= on page 183

ODS Destinations: Listing

UNDERLINE<=ON | OFF | *variable*>

specifies whether or not to draw a continuous line below the header. PROC TEMPLATE uses the second formatting character to draw the line. (See the discussion of FORMCHAR= on page 215.)

Default: OFF

ODS Destinations: Listing

VJUST=*justification* | *variable*

Specifies vertical justification for the header. *justification* can be 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

ODS Destinations: HTML and Printer

WIDTH=*positive-integer* | *variable*

specifies the width of the header in characters.

Default: If you do not specify a width, PROC TEMPLATE uses the column width.

Tip: If you want a vertical header, specify a width of 1.

ODS Destinations: Listing

DYNAMIC Statement

Defines a symbol that references a value that the data component supplies from the procedure or DATA step.

Scope: You can use the DYNAMIC statement in the definition of a table, column, header, or footer. A dynamic variable that is defined in a definition is available to that definition and to all the definitions that it contains.

Main discussion: “DYNAMIC Statement” on page 222

DYNAMIC *variable-1* <'text-1'> <... *variable-n* <'text-n'>>;

MVAR Statement

Defines a symbol that references a macro variable. ODS will use the value of the variable as a string. References to the macro variable are resolved when ODS binds the definition and the data component to produce an output object.

Scope: You can use the MVAR statement in the definition of a table, column, header, or footer. A macro variable that is defined in a definition is available to that definition and to all the definitions that it contains.

Main discussion: “MVAR Statement” on page 224

MVAR *variable-1* <'text-1'> <... *variable-n* <'text-n'>>;

NMVAR Statement

Defines a symbol that references a macro variable. ODS will convert 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 definition and the data component to produce an output object.

Scope: You can use the NMVAR statement in the definition of a table, column, header, or footer. A macro variable that is defined in a definition is available to that definition and to all the definitions that it contains.

Main discussion: “NMVAR Statement” on page 225

NMVAR *variable-1* <'text-1'> <... *variable-n* <'text-n'>>;

NOTES Statement

Provides information about the header.

Tip: The NOTES statement becomes part of the compiled header definition, which you can view with the SOURCE statement, whereas SAS comments do not.

NOTES *'text'*;

Required Arguments

text

provides information about the header.

TEXT Statement

Specifies the text of the header or the label of a variable in an output data set.

Featured in: Example 2 on page 238

TEXT *header-specification(s)*;

Required Arguments

header-specification(s)

specifies the text of the header. Each *header-specification* can be one of the following:

LABEL

uses the label of the object that the header applies to as the text of the header. For instance, if the header is for a column, _LABEL_ specifies the label for the variable that is associated with the column. If the header 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. Each *text-specification* can be

- a quoted string
- a variable, followed by an optional format. The variable can be 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 SPLIT= on page 183.) △

Default: If you don't use 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 specification, the header is a blank line.

Featured in: Example 2 on page 238

TEXT2 Statement

Provides an alternative header to use in the Listing output if the header that is provided by the TEXT statement is too long.

See: “TEXT Statement” on page 187

TEXT3 Statement

Provides an alternative header to use in the Listing output if the header that is provided by the TEXT2 statement is too long.

See: “TEXT Statement” on page 187

DEFINE STYLE Statement

Creates a style definition for any destination that supports the STYLE= option.

Requirement: An END statement must be the last statement in the definition.

Featured in: Example 4 on page 252

```
DEFINE STYLE style-path </ STORE=libname.template-store>;
  < PARENT=style-path>
  NOTES 'text';
  REPLACE new-style-element-name <FROM existing-style-element-name><'text'>
    < / style-attribute-specification(s)>;
  STYLE new-style-element-name <FROM existing-style-element-name><'text'>
    < / style-attribute-specification(s)>;
END;
```

Required Arguments

style-path

specifies where to store the style definition. A *style-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 definition to the first template store that you can write to in the current path.

Options

STORE=*libname.template-store*

specifies the template store in which to store the definition. If the template store does not exist, it is created.

Availability: Version 8 of the SAS System

Style-definition Attributes

PARENT=*style-path*

specifies the style definition that the current definition inherits from. A *style-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 definition inherits from the specified style definition in the first template store that you can read from in the current path.

When you specify a parent, all the style elements and attributes and statements that are specified in the parent's definition are used in the current definition unless the current definition overrides them.

SAS Institute provides some style definitions. You can specify one of these style definitions for *style-path*, or you can specify a user-defined style definition. Some of the style definitions that are currently shipped with the SAS System include:

- styles.default
- styles.beige
- styles.brick
- styles.brown
- styles.d3d
- styles.minimal
- styles.printer
- styles.statdoc.

For information on finding an up-to-date list of the style definitions and for viewing a style definition, see “Customizing Presentation Aspects of ODS Output” on page 42.

NOTES Statement

Provides information about the style definition.

Tip: The NOTES statement becomes part of the compiled style definition, which you can view with the SOURCE statement, whereas SAS comments do not.

NOTES *'text'*;

Required Arguments

text

provides information about the style definition. Text of this type becomes part of the compiled definition, which you can view with the SOURCE statement, whereas SAS comments do not.

REPLACE Statement

Adds to the child style definition a style element that also exists in the parent style definition. You can think of the REPLACE statement as replacing the statement that defines the like-named style element in the parent style definition. The REPLACE statement doesn't actually change the parent style definition, but PROC TEMPLATE builds the child style definition as if it had changed the parent. All style elements that inherit attributes from this style element inherit the ones that are specified in the REPLACE statement, not the ones that are used in the parent style definition.

Restriction: To use the REPLACE statement, you must specify a parent style definition with PARENT= in the DEFINE STYLE statement.

See also: "How Do Style-Definition Inheritance and Style-Element Inheritance Work?" on page 139

Featured in: Example 6 on page 265

```
REPLACE style-element-name-1 <FROM style-element-name-2><'text'>
  </ style-attribute-specification(s)>;
```

Required Arguments

style-element-name-1

names the style element to replace. A like-named style element must exist in the parent style definition. PROC TEMPLATE stores *style-element-name-1* in the current style definition and replaces all its attributes with the attributes that you specify in the REPLACE statement. If an attribute is defined in the like-named style element in the parent and you do not explicitly specify it in the REPLACE statement, the value of the attribute defaults to the value that was inherited from the parent of the like-named style element.

Options

style-element-name-2

names the style element that *style-element-name-1* inherits from. The style element must exist in the current style definition or in the parent of the current style definition. PROC TEMPLATE looks first in the current style definition for the style element. If it doesn't find it, it looks in the parent style definition.

style-attribute-specification(s)

specifies the style attributes for *style-element-name-1*. The new style element inherits from the parent style element all the attributes that the parent inherits. However, all the attributes that are explicitly specified in the definition of *style-element-name-2* must be respecified in the REPLACE statement if you want to keep them. You can override any attribute of the parent style element, whether it is inherited or explicitly defined, by specifying it in the REPLACE statement. Each *style-attribute-specification* has the following general form:

style-attribute-name=style-attribute-value

style-attribute-name

can be the name of an attribute that is listed in “Style Attributes” on page 192, or it can be the name of a user-defined attribute.

Restriction: If *style-attribute-name* refers to a user-defined attribute, you must enclose the name in quotation marks. If *style-attribute-name* refers to an attribute that is listed in “Style Attributes” on page 192, do not enclose the name in quotation marks. For more information on user-defined attributes, see “Style Attributes” on page 192.

style-attribute-value

assigns the value to the attribute. For information on style-attribute values, see “Style Attributes” on page 192.

'text'

provides information about the REPLACE statement. Text of this type becomes part of the compiled definition, which you can view with the SOURCE statement, whereas SAS comments do not.

STYLE Statement

Creates a new style-element.

Featured in: Example 4 on page 252

```
STYLE new-style-element-name <FROM existing-style-element-name><'text'>
  </ style-attribute-specification(s)>;
```

Required Arguments

new-style-element-name

names the style element to create. PROC TEMPLATE stores the style element in the current style definition.

Options

existing-style-element-name

names an existing style element to inherit from. The style element must exist in the current style definition or in the parent of the current style definition.

style-attribute-specification(s)

specify new style attributes or modifications to existing style attributes for the new style element. The new style element inherits all the style attributes of *existing-style-element-name*. You can override any of these attributes by specifying it in the STYLE statement. Each *style-attribute-specification* has the following general form:

style-attribute-name=style-attribute-value

style-attribute-name

can be the name of an attribute that is listed in “Style Attributes” on page 192, or it can be the name of a user-defined style attribute.

Restriction: If *style-attribute-name* refers to a user-defined attribute, you must enclose the name in quotation marks. If *style-attribute-name* refers to an attribute that is listed in “Style Attributes” on page 192, do not enclose the name in quotation marks.

style-attribute-value

assigns the value to the attribute. If you use an attribute from the list in “Style Attributes” on page 192, you must use the kind of value that the attribute expects.

For more information on style-attribute values, see “Style Attributes” on page 192.

'text'

provides information about the REPLACE statement. Text of this type becomes part of the compiled definition, which you can view with the SOURCE statement, whereas SAS comments do not.

Style Attributes

The default value that is used for an attribute depends on the style definition that is in use. For information on viewing the attributes in a style definition, see “Customizing Presentation Aspects of ODS Output” on page 42. The implementation of an attribute depends on the ODS destination that formats the output. In addition, if you are creating HTML output, the implementation of an attribute depends on the browser that you use.

Values for style attributes are often one of the following:

'string'

is a quoted character string.

dimension

is a nonnegative number, optionally followed by one of the following units of measure:

cm	centimeters
in	inches
mm	millimeters
pt	a printer's point
px	pixels (based on the size of a pixel on the target device)

Note: In Version 8 of the SAS System, only the Printer destination supports units of measure on dimensions. However, if you specify CSS in the ODS HTML statement, the HTML destination supports units of measure. The CSS option is experimental in Version 8. Δ

Default: For the HTML destination, pixels; for the Printer destination, units of 1/150 of an inch

color

is a string that identifies a color. A color can be

- any of the color names that are supported by SAS/GRAPH. These names include
 - 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).
- An RGB value with a leading pound sign (#) rather than CX (for example, #0023FF).
- One of the colors that exists in the SAS session when the style definition is used:
 - DMSBLUE
 - DMSRED
 - DMSPINK
 - DMSGREEN
 - DMSCYAN
 - DMSYELLOW
 - DMSWHITE
 - DMSORANGE
 - DMSBLACK
 - DMSMAGENTA
 - DMSGRAY
 - DMSBROWN
 - SYSBACK
 - SYSSECB
 - SYSFORE.

Note: Use these colors only if you are running SAS in the windowing environment. Δ

- An English description of an HLS value. Such descriptions use a combination of words to describe the lightness, the saturation, and the hue (in that order). The words that you can use are shown in the following table:

Lightness	Saturation	Hue
black	gray	blue
very dark	grayish	purple
dark	moderate	red
medium	strong	orange brown
light	vivid	yellow
very light		green
white		

You can combine these words to form a wide variety of colors. Some examples are

- 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, they are different. The reason for this is that ODS interprets them as SAS colors, which are mapped to different colors. Δ

You can also specify hues that are intermediate between two neighboring colors. To do so, combine one of the following adjectives with one of its neighboring colors:

- reddish
- orangish
- brownish
- yellowish
- greenish
- bluish
- purplish.

For example, you can use the following as hues:

- bluish purple (which is the same as purplish blue)
- reddish orange
- yellowish green.

See also: For information on SAS/GRAPH colors, see *SAS/GRAPH Software: Reference*.

format

is a SAS format or a user-defined format.

reference

is a reference to an attribute that is defined in the current style definition or in the parent (or beyond). In this case, the value that you use is the name of the style element followed, in parentheses, by the name of an attribute name within that element. For example, suppose that you create a style element called `DATACELL` that uses the `FOREGROUND=` and `BACKGROUND=` style elements this way:

```
style datacell / background=blue
                foreground=white;
```

Later, you can ensure that another style element, `NEWCELL`, uses the same background color by defining it this way:

```
style newcell / background=datacell(background);
```

Similarly, suppose that you create a style element called `HIGHLIGHTING` that defines three attributes this way:

```
style highlighting /
  "go"=green
  "caution"=yellow
  "stop"=red;
```

Later, you can define a style element called `MESSAGES` that uses the colors that are defined in `HIGHLIGHTING`:

```
style messages;
  "note"=highlighting("go")
  "warning"=highlighting("caution")
  "error"=highlighting("stop");
```

In this way, multiple style elements could use the colors that you define in `HIGHLIGHTING`. If you decide to change the value of `go` to blue, you simply change

its value in the definition of HIGHLIGHTING, and every style element that references highlighting (“go”) will use blue instead of green.

Note: In the first example, the style attribute BACKGROUND= is a predefined style attribute. Therefore, when you reference it, you do not put it in quotation marks. However, in the second example, go is a user-defined attribute. You define it with quotation marks, and when you reference it, you must use quotation marks. (This section describes all the predefined style attributes that are available.) Δ

You can use a special form of reference to get a value for a style attribute from the macro table at the time that the style element is used. For instance, the following STYLE statement uses the current value of the macro variable bkgr for the background color of the style element cell:

```
style cell / background=symget("bkgr");
```

Featured in: Example 5 on page 258

font-definition

A value can also be a font definition. A font definition has the following general format:

(“font-face-1 <... , font-face-n>”, font-size, keyword-list)

If you specify only one font face and if its name does not include a space character, you can omit the quotation marks. If you specify more than one font face, the browser uses the first one that is installed on your system.

font-size specifies the size of the font. *font-size* can be a dimension or a number without units of measure. If you specify a dimension, you must 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 document. See the discussion of dimensions on page 192.

keyword-list specifies the font weight, font style, and font width. You can include one value for each, in any order. The following table shows the keywords that you can use:

Keywords for Font Weight	Keywords for Font Style	Keywords for Font Width
MEDIUM	ITALIC	NORMAL*
BOLD	ROMAN	COMPRESSED*
DEMI_BOLD*	SLANT	EXTRA_COMPRESSED*
EXTRA_BOLD*		NARROW*
LIGHT		WIDE*
DEMI_LIGHT*		EXPANDED*
EXTRA_LIGHT*		

* Few fonts honor these values.

Featured in: Example 5 on page 258

To do this ...*	Use this attribute
Influence the characteristics of individual cells	
Specify how to handle leading spaces, trailing spaces, and line breaks.	ASIS=
Specify the height of the cell.	CELLHEIGHT=

To do this ...*	Use this attribute
Specify the width of the cell.	CELLWIDTH=
Specify the text to show in a tool tip for the cell.	FLYOVER=
Specify the window or frame in which to open the target of the link.	HREFTARGET=
Specify how to handle space characters.	NOBREAKSPACE=
Specify text to insert in the HTML	TAGATTR=
Specify a URL to link to.	URL=
Specify vertical justification.	VJUST=
Influence the characteristics of individual tables or cells	
Specify a font definition.	FONT=
Specify the font face to use.	FONT_FACE=
Specify the size of the font.	FONT_SIZE=
Specify the style of the font.	FONT_STYLE=
Specify the font weight.	FONT_WEIGHT=
Specify the font width compared to the width of the usual design.	FONT_WIDTH=
Specify the color of the foreground, which is primarily the color of the text.	FOREGROUND=
Specify the name of the stylesheet class to use for the table or cell.	HTMLCLASS=
Specify an ID for the table or cell.	HTMLID=
Specify individual attributes and values for the table or cell.	HTML_STYLE=
Specify justification.	JUST=
Specify the HTML code to place after the HTML table or cell.	POSTHTML=
Specify an image to place after the HTML table or cell.	POSTIMAGE=
Specify text to place after the cell or HTML table.	POSTTEXT=
Specify the HTML code to place before the HTML table or cell.	PREHTML=
Specify an image to place before the HTML table or cell.	PREIMAGE=
Specify text to place before the cell or HTML table.	PRETEXT=
Determine how less-than signs (<), greater-than signs (>), and ampersands (&) are interpreted.	PROTECTSPECIALCHARACTERS=

Influence the characteristics of tables

To do this ...*	Use this attribute
Specify the amount of white space on each of the four sides of the text in a cell.	CELLPADDING=
Specify the thickness of the spacing between cells.	CELLSPACING=
Specify the type of frame to use on an HTML table.	FRAME=
Specify the width of the HTML table.	OUTPUTWIDTH=
Specify the types of rules to use in an HTML table.	RULES=
Influence the characteristics of individual frames in HTML output	
Specify the string to use for bullets in the contents file.	BULLETS=
Specify the position, within the frame file, of the frames that display the contents and the page files.	CONTENTPOSITION=
Specify whether or not to put a scrollbar in the frames in the frame file that display the contents and the page files.	CONTENTSCROLLBAR=
Specify the width of the frames in the frame file that display the contents and the page files.	CONTENTSIZE=
Specify whether or not to put a border around the frame for an HTML file that uses frames.	FRAMEBORDER=
Specify the width of the border around the frames for an HTML file that uses frames.	FRAMEBORDERWIDTH=
Specify the width of the space between frames for HTML that uses frames.	FRAMESPACING=
Influence the characteristics of the document	
Specify the bottom margin for the document.	BOTTOMMARGIN=
Provide the value of the content type for pages that you send directly to a web server rather than to a file.	HTMLCONTENTTYPE=
Specify the entire doctype declaration for the HTML document, including the opening "<!DOCTYPE" and the closing ">".	HTMLDOCTYPE=
Specify the left margin for the document.	LEFTMARGIN=
Specify the color for links that have not yet been visited.	LINKCOLOR=
Specify whether or not to make this entry in the table of contents a link to the body file.	LISTENTRYANCHOR=
Specify whether or not to double space between entries in the table of contents.	LISTENTRYDBLSPACE=
Specify the height for graphics in the document.	OUTPUTHEIGHT=

To do this ...*	Use this attribute
Specify an upper limit for extending the width of the column.	OVERHANGFACTOR=
Specify HTML to place at page breaks.	PAGEBREAKHTML=
Specify the right margin for the document.	RIGHTMARGIN=
Specify the top margin for the document.	TOPMARGIN=
Specify whether or not to make the image that is specified by BACKGROUNDIMAGE= into a "watermark." A watermark appears in a fixed position as the window is scrolled.	WATERMARK=

* Different attributes affect different ODS destinations. For details, consult the documentation for a specific attribute.

Note: You can use the value `_UNDEF_` for any style attribute. ODS treats an attribute that is set to `_UNDEF_` as if its value had never been set, even in the parent or beyond. Δ

In the list of style attributes that follows, any attribute that is not documented as applying to a particular destination applies to all destinations that support the `STYLE=` option in the ODS statement that opens the destination. In Version 8 of the SAS System, the two destinations that support `STYLE=` are the HTML destination and the Printer destination.

ASIS=ON|OFF

specifies how to handle leading spaces, trailing spaces, and line breaks.

ON

prints text with leading spaces, trailing spaces, and line breaks as they are.

OFF

trims leading spaces and trailing spaces. OFF ignores line breaks.

Applies to: cells

BACKGROUND=color

specifies the color of the background.

Tip: Generally, the background color of the cell overrides the background color of the table. You see the background color for the table only as the space between cells (see `CELLSPACING=` on page 200).

Applies to: tables or cells

Featured in: Example 4 on page 252 and Example 6 on page 265

BACKGROUNDIMAGE='string'

specifies an image to use as the background. Viewers that can tile the image as the background for the HTML table that the procedure creates will do so. *string* is the name of a GIF or JPEG file. You can use a simple file name, a complete path, or a URL. However, the most versatile approach is to use a simple filename and to place all image files in the local directory.

Applies to: tables or cells

ODS Destinations: HTML

BODYSROLLBAR=YES | NO | AUTO

specifies whether or not to put a scrollbar in the frame that references the body file.

Tip: Typically, BODYSCROLLBAR is set to AUTO.

Applies to: frame

ODS Destinations: HTML

BODYSIZE=*dimension* | *number* % | *

specifies the width of the frame that displays the body file in the HTML frame file. (For information on the HTML files that ODS creates, see “Files Produced by the HTML Destination” on page 36.)

dimension

is a nonnegative number. The unit of measure is pixels.

number %

specifies the width of the frame 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.

Applies to: frame

ODS Destinations: HTML

BORDERCOLOR=*color*

specifies the color of the border if the border is just one color.

Applies to: tables or cells

BORDERCOLORDARK=*color*

specifies the darker color to use in a border that uses two colors to create a three-dimensional effect.

Applies to: tables or cells

ODS Destinations: HTML

Featured in: Example 4 on page 252 and Example 6 on page 265

BORDERCOLORLIGHT=*color*

specifies the lighter color to use in a border that uses two colors to create a three-dimensional effect.

Applies to: tables or cells

ODS Destinations: HTML

Featured in: Example 4 on page 252 and Example 6 on page 265

BORDERWIDTH=*dimension*

specifies the width of the border of the table.

Applies to: tables

Tip: Typically, when BORDERWIDTH=0, the ODS destination sets RULES=NONE (see the discussion of RULES= on page 207) and FRAME=VOID (see the discussion of FRAME= on page 203).

Featured in: Example 4 on page 252 and Example 6 on page 265

BOTTOMMARGIN=*dimension*

specifies the bottom margin for the document.

Applies to: document

BULLET='string'

specifies the string to use for bullets in the contents file. ODS uses bullets in the contents file. *string* can be one of the following:

- circle
- decimal
- disc
- lower-alpha
- lower-roman
- none
- square
- upper-alpha
- upper-roman.

Applies to: contents

ODS Destinations: HTML

CELLHEIGHT=*dimension* | *integer*%

specifies the height of the cell. If you specify a percent, it represents a percentage of the height of the table. A row of cells will have the height of the highest cell in the row.

Tip: HTML automatically sets cell height appropriately. You should seldom need to specify this attribute.

Applies to: cells

ODS Destinations: HTML and Printer

CELLPADDING=*dimension* | *integer*%

specifies the amount of white space on each of the four sides of the text in a cell.

Applies to: tables

Featured in: Example 6 on page 265

CELLSPACING=*dimension*

specifies the thickness of the spacing between cells.

Applies to: tables

Interaction: If BORDERWIDTH= is nonzero, and if the background color of the cells contrasts with the background color of the table, the color of the cell spacing is determined by the table's background.

Featured in: Example 4 on page 252 and Example 6 on page 265

CELLWIDTH=*dimension* | *integer*%

specifies the width of the cell. If you specify a percent, it represents a percentage of the width of the table. A column of cells will have the width of the widest cell in the column.

Applies to: cells

Tip: The ODS destination sets cell width appropriately. You should seldom need to specify this attribute.

ODS Destinations: HTML and Printer

CONTENTPOSITION=*position*

specifies the position, within the frame file, of the frames that display the contents and the page files. (For information on the HTML files that ODS creates, see "Files Produced by the HTML Destination" on page 36.) *position* can be

LEFT

places the frames on the left.

Alias: L

RIGHT

places the frames on the right.

Alias: R

TOP

places the frames at the top.

Alias: T

BOTTOM

places the frames at the bottom.

Alias: B

Applies to: frame

ODS Destinations: HTML

CONTENTSCROLLBAR=YES | NO | AUTO

specifies whether or not to put a scrollbar in the frames in the frame file that display the contents and the page files. (For information on the HTML files that ODS creates, see “Files Produced by the HTML Destination” on page 36.)

Tip: Typically, CONTENTSCROLLBAR is set to AUTO.

Applies to: frame

ODS Destinations: HTML

CONTENTSIZE=*dimension* | *number* % | *

specifies the width of the frames in the frame file that display the contents and the page files. (For information on the HTML files that ODS creates, see “Files Produced by the HTML Destination” on page 36.)

dimension

is a nonnegative number. The unit of measure is pixels.

number %

specifies the width of the frames 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.

See also: BODYSIZE= on page 199

Applies to: frame

ODS Destinations: HTML

FLYOVER=*string*

specifies the text to show in a tool tip for the cell.

Applies to: cells

ODS Destinations: HTML

FONT=*font-definition*

specifies a font definition to use. For more information, see the discussion of font definition on page 195.

Applies to: tables and cells

Tip: When you specify this attribute for a table, it 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, you must set the attribute for a cell.

Featured in: Example 6 on page 265

FONT_FACE='string-1<..., string-n>'

specifies the font face to use. If you supply multiple font faces, the browser uses the first one that is installed on your system.

You cannot be sure what fonts are available to someone who is viewing your output in a browser or printing it on a high-resolution printer. Most devices support

- times
- courier
- arial, helvetica.

Applies to: cells

Tip: When you specify this attribute for a table, it 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, you must set the attribute for a cell.

Featured in: Example 4 on page 252

FONT_SIZE=dimension | size

specifies the size of the font. The value of *size* is relative to all other font sizes in the document.

Applies to: table and cells

Range: 1 to 7, for *size*

Restriction: If you specify a dimension, you must specify a unit of measure.

Without a unit of measure, the number becomes a relative size. See the discussion of dimensions on page 192.

Tip: When you specify this attribute for a table, it 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, you must set the attribute for a cell.

Featured in: Example 4 on page 252

FONT_STYLE=ITALIC | ROMAN | SLANT

specifies the style of the font. In many cases, italic and slant map to the same font.

Applies to: tables and cells

Tip: When you specify this attribute for a table, it 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, you must set the attribute for a cell.

Featured in: Example 4 on page 252 and Example 6 on page 265

FONT_WEIGHT=weight

specifies the font weight. *weight* can be any of the following:

- MEDIUM
- BOLD
- DEMI_BOLD
- EXTRA_BOLD
- LIGHT
- DEMI_LIGHT
- EXTRA_LIGHT.

Applies to: tables and cells

Restriction: You cannot be sure what font weights are available to someone who is viewing your output in a browser or printing it on a high-resolution printer. Most devices support only MEDIUM and BOLD, and possibly LIGHT.

Tip: When you specify this attribute for a table, it 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, you must set the attribute for a cell.

Featured in: Example 4 on page 252

FONT_WIDTH=relative-width

specifies the font width compared to the width of the usual design. *relative-width* can be any of the following:

- NORMAL
- COMPRESSED
- EXTRA_COMPRESSED
- NARROW
- WIDE
- EXPANDED.

Applies to: tables and cells

Restriction: Few fonts honor these values.

Tip: When you specify this attribute for a table, it 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, you must set the attribute for a cell.

Featured in: Example 4 on page 252

FOREGROUND=color

specifies the color of the foreground, which is primarily the color of text.

Applies to: tables or cells

Tip: When you specify this attribute for a table, it 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, you must set the attribute for a cell.

Featured in: Example 4 on page 252 and Example 6 on page 265

FRAME=frame-type

specifies the type of frame to use on a table. The following table shows the possible values of *frame-type* and their meanings:

This value of <i>frame-type</i> ...	Creates this kind of frame around the table
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

Applies to: tables

Featured in: Example 6 on page 265

FRAMEBORDER=ON | OFF

specifies whether or not to put a border around the frame for an HTML file that uses frames.

Applies to: frame

ODS Destinations: HTML

FRAMEBORDERWIDTH=*dimension*

specifies the width of the border around the frames for an HTML file that uses frames.

Applies to: frame

ODS Destinations: HTML

FRAMESPACING=*dimension*

specifies the width of the space between frames for HTML that uses frames.

Applies to: frame

ODS Destinations: HTML

HREFTARGET=*target*

specifies the window or frame in which to open the target of the link. *target* can be

_BLANK

opens the target in a new, blank window. The window has no name.

_PARENT

opens the target in the window from which the current window was opened.

_SEARCH

opens the target in the browser's search pane.

Restriction: Only available in Internet Explorer 5.0 or later.

_SELF

opens the target in the current window.

_TOP

opens the target in the topmost window.

'name'

opens the target in the specified window or the frame.

Default: _SELF

Applies to: cells

ODS Destinations: HTML

HTMLCLASS=*'string'*

specifies the name of the stylesheet class to use for the table or cell.

Applies to: tables and cells

Availability: Version 8 of the SAS System

ODS Destinations: HTML

HTMLCONTENTTYPE=*'string'*

provides the value of the content type for pages that you send directly to a web server rather than to a file.

Tip: The value of *string* is usually "text/html".

Applies to: document

ODS Destinations: HTML

HTMLDOCTYPE='string'

specifies the entire doctype declaration for the HTML document, including the opening “<!DOCTYPE” and the closing “>”.

Tip: Most users will never need to use this attribute.

Applies to: document

ODS Destinations: HTML

HTMLID='string'

specifies an id for the table or cell. The id is for use by a Java script.

Applies to: tables and cells

ODS Destinations: HTML

HTMLSTYLE='string'

specifies individual attributes and values for the table or cell.

Applies to: tables and cells

ODS Destinations: HTML

JUST=justification

specifies justification, where *justification* can be

CENTER

specifies center justification.

Alias: C

Applies to: tables and cells

LEFT

specifies left justification.

Alias: L

Applies to: tables and cells

RIGHT

specifies right justification.

Alias: R

Applies to: tables and cells

Restriction: Not all contexts support RIGHT. If RIGHT is not supported, it is interpreted as CENTER.

LEFTMARGIN=dimension

specifies the left margin for the document.

Applies to: document

LINKCOLOR=color

specifies the color for links that have not yet been visited.

Applies to: document

ODS Destinations: HTML

LISTENTRYANCHOR=ON | OFF

specifies whether or not to make this entry in the table of contents a link to the body file.

Applies to: document

ODS Destinations: HTML

Availability: Version 8 of the SAS System

NOBREAKSPACE=ON | OFF

specifies how to handle space characters.

ON

does not allow SAS to break a line at a space character.

OFF

allows SAS to break a line at a space character if appropriate.

Applies to: cells

OUTPUTHEIGHT=*dimension*

specifies the height for graphics in the document.

Applies to: document

ODS Destinations: HTML

OUTPUTWIDTH=*dimension* | *number*%

specifies the width of the table. If you specify a percent, it represents a percentage of the width of the browser window.

Applies to: tables

Tip: Use OUTPUTWIDTH=100% to make the table as wide as the window that it is open in.

ODS Destinations: HTML

OVERHANGFACTOR=*nonnegative-number*

specifies an upper limit for extending the width of the column. The HTML that is generated by ODS tries to ensure that the text in a column wraps when it reaches the requested column width. If you make the overhang factor greater than 1, the text can extend beyond the specified width.

Tip: Typically, an overhang factor between 1 and 2 works well.

Applies to: document

ODS Destinations: HTML

PAGEBREAKHTML='*string*'

specifies HTML to place at page breaks.

Applies to: document

ODS Destinations: HTML

POSTHTML='*string*'

specifies the HTML code to place after the table or cell.

Applies to: tables or cells

ODS Destinations: HTML

Featured in: Example 6 on page 265

POSTIMAGE='*string*'

specifies an image to place after the table or cell. *string* is the name of a GIF or JPEG file. You can use a simple filename, a complete path, or a URL. However, the most versatile approach is to use a simple filename and to place all image files in the local directory.

Applies to: tables or cells

ODS Destinations: HTML

POSTTEXT='*string*'

specifies text to place after the cell or table.

Applies to: tables or cells

PREHTML='*string*'

specifies the HTML code to place before the table or cell.

Applies to: tables or cells

ODS Destinations: HTML

PREIMAGE='string'

specifies an image to place before the table or cell. *string* is the name of a GIF or JPEG file. You can use a simple filename, a complete path, or a URL. However, the most versatile approach is to use a simple filename and to place all image files in the local directory.

Applies to: tables or cells

ODS Destinations: HTML

PRETEXT='string'

specifies text to place before the cell or table.

Applies to: tables or cells

PROTECTSPECIALCHARACTERS=ON | OFF | AUTO

determines how less-than signs (<), greater-than signs (>), and ampersands (&) are interpreted. In HTML, 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 destination so that HTML does not interpret them as part of the markup language. Using ON enables you to show HTML markup in your document.

OFF

interprets special characters as HTML code. That is, when OFF is in effect, the characters are passed to the HTML destination without any protection so that HTML interprets them as part of the markup language.

AUTO

interprets any string that starts with a < and ends with a > as HTML (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 HTML meaning.

Applies to: cells

ODS Destinations: HTML

RIGHTMARGIN=*dimension*

specifies the right margin for the document.

Applies to: document

RULES=*rule-type*

specifies the types of rules to use in a table. The following table shows the possible values of *rule* and their meanings:

This value of <i>rule</i> ...	Creates rules in these locations
ALL	between all rows and columns
COLS	between all columns
GROUP	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

Applies to: tables

Featured in: Example 6 on page 265

TAGATTR='string'

specifies text to insert in the HTML. The string must be valid HTML for the context in which the style element is rendered. Many style elements are rendered between <TD> and </TD> tags. To determine how a style element is rendered, look at the source for the output.

Applies to: cells

ODS Destinations: HTML

TOPMARGIN=dimension

specifies the top margin for the document.

Applies to: document

URL='uniform-resource-locator'

specifies a URL to link to from the current cell.

Applies to: cells

ODS Destinations: HTML

VISITEDLINKCOLOR=color

specifies the color for links that have been visited.

Applies to: document

ODS Destinations: HTML

VJUST='justification'

specifies vertical justification, where *justification* can be

TOP

specifies top justification.

Alias: T

BOTTOM

specifies bottom justification.

Alias: B

MIDDLE

specifies center justification.

Alias: M

Applies to: cells

WATERMARK=ON | OFF

specifies whether or not to make the image that is specified by BACKGROUNDIMAGE= into a “watermark.” A watermark appears in a fixed position as the window is scrolled.

Applies to: document

ODS Destinations: HTML

END Statement

Ends the style definition.

END;

DEFINE TABLE Statement

Creates a table definition.

Requirement: An END statement must be the last statement in the definition.

Interaction: A table definition can contain one or more column, header, or footer definitions.

Featured in: Example 2 on page 238, Example 3 on page 247, and Example 4 on page 252

```

DEFINE TABLE table-path </ STORE=libname.template-store>;
  <table-attribute-1; <... table-attribute-n; >>
  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 definition-type definition-path;
    statements-and-attributes
  END;
  DYNAMIC variable-1 <'text-1'> <... variable-n <'text-n'>>;
  FOOTER footer-name(s);
  HEADER header-name(s);
  MVAR variable-1 <'text-1'> <... variable-n <'text-n'>>;
  NMVAR variable-1 <'text-1'> <... variable-n <'text-n'>>;
  NOTES 'text';
  TRANSLATE expression-1 INTO expression-2 <... , expression-n INTO
    expression-m; >
  END;

```

To do this ...	Use this statement
Set one or more table attributes.	<i>table-attribute(s)</i>
Set the style element of the cells in the table that contain numeric variables according to the values of the variables.	CELLSTYLE-AS
Declare a symbol as a column in the table and specify the order of the columns.	COLUMN
Create a definition for a column, header, or footer.	DEFINE
Define a symbol that references a value that the data component supplies from the procedure or DATA step.	DYNAMIC
Declare a symbol as a footer in the table and specify the order of the footers.	FOOTER
Declare a symbol as a header in the table and specify the order of the headers.	HEADER

To do this ...	Use this statement
Define a symbol that references a macro variable. ODS will use the value of the variable as a string. References to the macro variable are resolved when ODS binds the definition and the data component to produce an output object.	MVAR
Define a symbol that references a macro variable. ODS will convert the value of the variable to a number (stored as a double) before use. References to the macro variable are resolved when ODS binds the definition and the data component to produce an output object.	NMVAR
Provide information about the table.	NOTES
Translate the specified numeric values to other values.	TRANSLATE-INTO
End a definition, or end the editing of a definition.	END

Required Arguments

table-path

specifies where to store the table definition. 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.) PROC TEMPLATE writes the definition to the first template store in the current path that you can write to.

Options

STORE=*libname.template-store*

specifies the template store in which to store the definition. If the template store does not exist, it is created.

Availability: Version 8 of the SAS System

Table Attributes

This section lists all the attributes that you can use in a table definition. For all attributes that support a value of ON, the following forms are equivalent:

```
ATTRIBUTE-NAME
ATTRIBUTE-NAME=ON
```

For all attributes that support a value of *variable*, *variable* can be any variable that you declare in the table definition with the DYNAMIC, MVAR, or NMVAR statement. If the attribute is a boolean, the value of *variable* should resolve to one of the following:

ON	YES	0
ON	_YES_	FALSE
1	OFF	NO
TRUE	_OFF_	_NO_

To do this ...*	Use this attribute
Influence the layout of the table	
Specify whether or not 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=
Specify whether or not to center each data panel independently if the entire table does not fit in one data panel	CENTER=
Specify whether or not to force a new page before printing the table	NEWPAGE=
Specify the number of sets of columns to place on a page	PANELS=
Specify the number of blank characters to place between sets of columns when PANELS= is in effect	PANELSPACE=
Specify the number of lines that must be available on the page in order to print the body of the table	REQUIRED_SPACE=
Specify the number of lines to place between the previous output object and the current one	TOPSPACE=
Influence the layout of rows and columns	
Specify the maximum number of blank characters to place between columns	COL_SPACE_MAX=
Specify the minimum number of blank characters to place between columns	COL_SPACE_MIN=
Specify the name of the column whose value provides formatting information about the space before each row of the definition	CONTROL=
Specify whether or not to double space between the rows of the table	DOUBLE_SPACE=
Specify whether or not extra space is evenly divided among all columns of the table	EVEN=
Specify whether or not to split a long stacked column across page boundaries	SPLIT_STACK=
Influence the display of the values in header cells and data cells	

To do this ...*	Use this attribute
Specify whether or not to suppress the blanking of 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=
Specify which format to use if both a column definition and a data component specify one	DATA_FORMAT_OVERRIDE=
Specify whether to justify the format fields within the columns or to justify the values within the columns without regard to the format fields	JUSTIFY=
Specify whether or not to order the columns by their order in the data component	ORDER_DATA=
Specify the source of the values for the format width and the decimal width if they are not specified	USE_FORMAT_DEFAULTS=
Use the column name as the column header if neither the column definition nor the data component specifies a header	USE_NAME=
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=
Specify the number of blank lines to place between the last row of headers and the first row of data	HEADER_SPACE=
Specify whether or not to draw a continuous line above the first table footer (or, if there is no table footer, below the last row of the data on a page)	OVERLINE=
Specify whether or not to print table footers and any overlining of the table footers	PRINT_FOOTERS=
Specify whether or not to print table headers and any underlining of the table headers	PRINT_HEADERS=
Specify whether or not to draw a continuous line under the last table header (or, if there is no table header, under the last row of the data on a page)	UNDERLINE=
Influence the HTML output	
Specify whether or not to place the object in a table of contents if you create one	CONTENT=
Specify the label to use for the output object in the contents file, the Results window, and the trace record	CONTENTS_LABEL=
Specify whether or not to print the current byline before the table	BYLINE=

To do this ...*	Use this attribute
Define the characters to use as the line-drawing characters in the table	FORMCHAR=
Specify a label for the table	LABEL=
Specify the table that the current definition inherits from	PARENT=
Specify the style element to use for the table and any changes to the attributes	STYLE=
Specify the special data set type of a SAS data set	TYPE=

* Different attributes affect different ODS destinations. For details, consult the documentation for a specific attribute.

BALANCE<=ON | OFF | *variable*>

specifies whether or not 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

ODS Destinations: Listing and Printer

BYLINE<=ON | OFF | *variable*>

specifies whether or not to print the current byline before the table.

Default: OFF

Restriction: This attributes 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.

ODS Destinations: All but Output

CENTER<=ON | OFF | *variable*>

specifies whether or not to center each data panel independently if the entire table does not fit in the space that is provided.

Default: ON

ODS Destinations: Listing and Printer

CLASSLEVELS<=ON | OFF | *variable*>

specifies whether or not to suppress the blanking of 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

ODS Destinations: All but Output

Featured in: Example 4 on page 252

COL_SPACE_MAX=*positive-integer* | *variable*

specifies the maximum number of blank characters to place between the columns.

Default: 4

ODS Destinations: Listing

COL_SPACE_MIN=*positive-integer* | *variable*

specifies the minimum number of blank characters to place between the columns.

Default: 2

ODS Destinations: Listing

CONTENTS<=ON | OFF | *variable*>

specifies whether or not to place the object in a table of contents if you create one.

Default: ON

ODS Destinations: HTML

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.

ODS Destinations: HTML

CONTROL=*column-name* | *variable*

specifies the name of the column whose values provide formatting information about the space before each row of the definition. The value of CONTROL= should be the name of a column of type character with a length equal to 1.

If the value in the control column is ...	This occurs before the row is created ...
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
a 'b' or a 'B'	ODS tries to insert a panel break if the entire table does not fit in the space that is provided

Default: none

ODS Destinations: Listing and Printer

DATA_FORMAT_OVERRIDE<=ON | OFF | *variable*>

specifies which format to use if both a column definition and a data component specify one.

ON

uses the format that the data component specifies.

OFF

use the format that the column definition specifies.

Default: OFF

Availability: Version 8 of the SAS System

ODS Destinations: All

DOUBLE_SPACE<=ON | OFF | *variable*>

specifies whether or not to double space between the rows of the table.

Default: OFF

ODS Destinations: Listing

Featured in: Example 1 on page 233 and Example 2 on page 238

EVEN<=ON | OFF | *variable*>

specifies whether or not extra space is evenly divided among all columns of the table.

Default: OFF

ODS Destinations: 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 footers.

Default: 1

ODS Destinations: Listing

FORMCHAR=*string* | *variable*

Defines the characters to use as the line-drawing characters in the table. Currently, PROC TEMPLATE 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, you must specify both the first and second formatting characters. For instance, the following 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.

Tip: You can use any character in formatting-characters, including hexadecimal characters. If you use hexadecimal characters, you must put an x after the closing quote. For instance, the following option assigns the hexadecimal character 2D to the first formatting character, the hexadecimal character 7C to the second character, and does not alter the remaining characters:

```
formchar='2D7C'x
```

ODS Destinations: Listing

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 underlining is a header.

Default: 1

ODS Destinations: Listing

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. For a discussion of how the Listing destination justifies data, see “How Are Values in Table Columns Justified?” on page 138.

Default: OFF

Interaction: JUSTIFY=ON can interfere with decimal alignment.

Tip: If you translate numeric data to character data, you may need to use JUSTIFY= to align the data as you wish.

ODS Destinations: Listing

LABEL=*text* | *variable*

specifies a label for the table.

Default: PROC TEMPLATE uses the first of the following that it finds:

- a label that the table definition provides.
- a label that the data component provides.
- the first spanning header in the table.

ODS Destinations: All

NEWPAGE<=ON | OFF | *variable*>

specifies whether or not 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.

ODS Destinations: All but Output

ORDER_DATA<=ON | OFF | *variable*>

specifies whether or not to order the columns by their order in the data component.

Default: OFF

When ORDER_DATA=OFF, the default order for columns is the order that they are specified in the COLUMN statement. If you do not use a COLUMN statement, the default order for columns is the order in which you define them in the definition.

Tip: The Output destination always uses the order of the columns in the data component when it creates an output data set.

Interaction: ORDER_DATA is most useful for ordering generic columns.

ODS Destinations: All but Output

OVERLINE<=ON | OFF | *variable*>

specifies whether or not 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). PROC TEMPLATE uses the second formatting character to draw the line. (See the discussion of FORMCHAR= on page 215.)

Default: OFF

See also: UNDERLINE= on page 218 (for tables), UNDERLINE= on page 170 (for columns), and OVERLINE= on page 167 (for columns)

ODS Destinations: Listing

Featured in: Example 1 on page 233

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, you can display the data in multiple sets of columns so that rows that would otherwise appear on multiple pages appear on the same page.

Tip: If the number of panels that is specified is larger than the number of panels that can fit on the page, the definition creates as many panels as it can. Let the table definition put your data in the maximum number of panels that can fit on the page by specifying a large number of panels (for example, 99).

ODS Destinations: Listing and Printer

PANEL_SPACE=*positive-integer* | *variable*

specifies the number of blank characters to place between sets of columns when PANELS= is in effect.

Default: 2

ODS Destinations: Listing

PARENT=*table-path*

specifies the table that the current definition 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 definition 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 the attributes and statements that are specified in the parent's definition are used in the current definition unless the current definition specifically overrides them.

ODS Destinations: All

PRINT_FOOTERS<=ON | OFF | *variable*>

specifies whether or not print table footers and any overlining of the table footers.

Default: ON

See also: OVERLINE= on page 216

ODS Destinations: All but Output

PRINT_HEADERS<=ON | OFF | *variable*>

specifies whether or not to print the table headers and any underlining of the table headers.

Default: ON

Interaction: When used in a table definition, PRINT_HEADERS affects only headers for the table, not the headers for individual columns. (See the discussion of the PRINT_HEADERS column attribute on page 168.)

See also: UNDERLINE= on page 218

ODS Destinations: All but Output

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

ODS Destinations: Listing and Printer

SPLIT_STACK<=ON | OFF | *variable*>

specifies whether or not to split a long stacked column across page boundaries.

Default: OFF

ODS Destinations: Listing

STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>

specifies the style element and any changes to its attributes to use for the table. 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 render the table. The style element must be part of a style definition that is registered with the Output Delivery System. SAS Institute provides some style definitions. Users can create their own style definitions with PROC TEMPLATE (see “DEFINE STYLE Statement” on page 188). By default, ODS renders different parts of ODS output with different elements. For instance, by default, a table is rendered with the style element **table**. The style definitions that SAS Institute provides do not provide another style element that you would be likely to want to use instead of **table**. However, you may have a user-defined style element at your site that would be appropriate to specify.

The style element provides the basis for rendering the table. Additional style attributes that you provide can modify the rendering.

For information on finding an up-to-date list of the style definitions and for viewing a style definition so that you can see the style elements that are available, see “Customizing Presentation Aspects of ODS Output” on page 42. For information about the default style definition that ODS uses, see “What Is the Default Style Definition Like?” on page 133.

style-element-name can be either the name of a style element or a variable whose value is a style element.

Default: table

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 on the style attributes that you can specify, see “Style Attributes” on page 192.

ODS Destinations: HTML and Printer

TOP_SPACE=*positive-integer* | *variable*

specifies the number of lines to place between the previous output object and the current one.

Default: 1

ODS Destinations: Listing and Printer

TYPE=*string* | *variable*

specifies the special data set type of a SAS data set.

Restriction: PROC TEMPLATE does *not* verify that

- the SAS data set type that you specify is a valid data set type
- the structure of the data set that you create is appropriate for the type that you have assigned.

Tip: 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.

Destination: Output

UNDERLINE<=ON | OFF | *variable*>

specifies whether or not to draw a continuous line under the last table header (or, if there is no table header, above the first row of data on a page). PROC TEMPLATE uses the second formatting character to draw the line. (See the discussion of FORMCHAR= on page 215.)

Default: OFF

See also: OVERLINE= on page 216 (for tables), UNDERLINE= on page 170 (for columns), and OVERLINE= on page 167 (for columns)

ODS Destinations: Listing

Featured in: Example 1 on page 233 and Example 2 on page 238

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

ODS Destinations: All but Output

USE_NAME<=ON | OFF | *variable*>

uses the column name as the column header if neither the column definition nor the data component specifies a header.

Default: OFF

Tip: Use this attribute when column names are derived from a data set and the columns are generic.

ODS Destinations: All but Output

WRAP<=ON | OFF | *variable*>

specifies whether to split a table that is too wide to fit in the space that is provided 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.

See also: WRAP_SPACE= on page 219 and ID= on page 165

ODS Destinations: Listing and Printer

WRAP_SPACE<=ON | OFF | *variable*>

specifies whether or not 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

See also: WRAP= on page 219

ODS Destinations: Listing and Printer

CELLSTYLE-AS Statement

Sets the style element of the cells in the table that contain numeric variables according to the values of the variables. Use this statement to set the presentation characteristics (such as foreground color, font face, flyover) of individual cells.

Restriction: The CELLSTYLE-AS statement in a table definition applies only to numeric variables. To specify style elements for individual values of a character variable, use CELLSTYLE-AS in the definition of that column. (See “DEFINE COLUMN Statement” on page 159.)

```
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 table cell that contains a numeric variable. It can be any numeric expression that is valid in the WHERE statement (or the WHERE= data set option). For information on expressions that you can use in the WHERE statement, see “Statements” in *SAS Language Reference: Dictionary*. Use `_VAL_` to represent the value of the current column. You may also reference symbols that you declared in a DYNAMIC, MVAR, or NVAR statement in the table definition.

If *expression* resolves to TRUE (a non-zero 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.

Restriction: You may not reference the values of other columns in *expression*.

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.

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 on the style attributes that you can set in a table definition, see “Style Attributes” on page 192.

Options

style-element-name

is the name of a style element that is part of a style definition that is registered with the Output Delivery System. SAS Institute provides some style definitions. You can create your own style definitions and style elements with PROC TEMPLATE. (See “DEFINE STYLE Statement” on page 188.)

The style elements that you would be most likely to use with the CELLSTYLE-AS statement are

- data
- datafixed
- dataempty
- dataemphasis
- dataemphasisfixed
- datastrong
- datastrongfixed.

The style element provides the basis for rendering the cell. Additional style attributes that you provide can modify the rendering.

COLUMN Statement

Declares a symbol as a column in the table and specifies the order of the columns.

Featured in: Example 2 on page 238

COLUMN *column(s)*;

Required Arguments

column

is one or more columns. If the column is defined outside the current table definition, you must reference it by its path in the template store. Columns in the definition are laid out from left to right in the same order that you specify them in the COLUMN statement.

To stack values for two or more variables in the same column, put parentheses around the variables that you want to stack. 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 instance, the following COLUMN statement produces a definition in which

- 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);
```

You can 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 instance, the following COLUMN statement produces a report in which

- 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);
```

Default: If you do not use a COLUMN statement, ODS makes a column for each column definition (DEFINE COLUMN statement), and places the columns in the same order that the column definitions have in the table definition.

If you use a COLUMN statement but do not use a DEFINE COLUMN statement for any of the columns, ODS uses a default column definition that is based on the type of data in the column.

Tip: You can use a list of variable names, such as DAY1–DAY10, to specify multiple variables.

DEFINE Statement

Creates a definition inside a table definition.

Main discussion: “DEFINE COLUMN Statement” on page 159, “DEFINE FOOTER Statement” on page 176, and “DEFINE HEADER Statement” on page 177

```
DEFINE definition-type definition-name;  
statements-and-attributes  
END;
```

Required Arguments

definition-type

specifies the type of definition to create, where *definition-type* is one of the following:

COLUMN

FOOTER

HEADER

The *definition-type* determines what other statements and what attributes can go in the definition. For details, see the documentation for the corresponding DEFINE statement.

definition-name

specifies the name of the new object.

Restriction: *definition-name* must be a single-level name.

Note: If you want to reference the definition that you are creating from another definition, you must create it outside the table definition. Δ

DYNAMIC Statement

Defines a symbol that references a value that the data component supplies from the procedure or DATA step.

Scope: You can use the DYNAMIC statement in the definition of a table, column, header, or footer. A dynamic variable that is defined in a definition is available to that definition and to all the definitions that it contains.

Featured in: Example 4 on page 252

DYNAMIC *variable-1* <'text-1'> <... *variable-n* <'text-n'>>;

Required Arguments

variable

Names a variable that the data component supplies. ODS resolves the value of the variable when it binds the definition and the data component.

Tip: Dynamic variables are most useful to the authors of SAS procedures and to DATA step programmers.

Options

text

is text that you can place in the definition to explain the dynamic variable's use. Text of this type becomes part of the compiled definition, which you can view with the SOURCE statement, whereas SAS comments do not.

FOOTER Statement

Declares a symbol as a footer in the table and specifies the order of the footers.

FOOTER *footer-specification(s)*;

Required Arguments

footer-specification

is one or more footers. If the footer is defined outside the current table definition, you must reference it by its path in the template store. Footers in the definition are laid out from top to bottom in the same order that you specify them in the FOOTER statement. Each *footer-specification* can be

"string"

specifies the text to use for the footer. If you use a string, you do not need to use a DEFINE FOOTER statement. However, you cannot specify any footer attributes except for a split character. If SPLIT= 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 treats it as the split character.

See also: SPLIT=

footer-path

is the path of the footer definition to use. A footer-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 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 do not use a FOOTER statement, ODS makes a footer for each footer definition (DEFINE FOOTER statement), and places the footers in the same order that the footer definitions have in the table definition.

HEADER Statement

Declares a symbol as a header in the table and specifies the order of the headers.

HEADER *header-specification(s)*;

Required Arguments

header-specification

is one or more headers. If the header is defined outside the current table definition, you must reference it by its path in the template store. Headers in the definition are laid out from top to bottom in the same order that you specify them in the HEADER statement. Each *header-specification* can be

"string"

specifies the text to use for the header. If you use 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 SPLIT= 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 treats it as the split character.

See also: SPLIT=

header-path

is the path of the header definition 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 do not use a HEADER statement, ODS makes a header for each header definition (DEFINE HEADER statement), and places the headers in the same order that the header definitions have in the table definition.

Featured in: Example 2 on page 238

MVAR Statement

Defines a symbol that references a macro variable. ODS will use the value of the variable as a string. References to the macro variable are resolved when ODS binds the definition and the data component to produce an output object.

Scope: You can use the MVAR statement in the definition of a table, column, header, or footer. A macro variable that is defined in a definition is available to that definition and to all the definitions that it contains.

Featured in: Example 2 on page 238 and Example 4 on page 252

MVAR *variable-1* <'text-1'> <... *variable-n* <'text-n'>>;

Required Arguments

variable

Names a macro variable to reference in the definition. ODS will use the value of the macro variable as a string. ODS does not resolve the value of the macro variable until it binds the definition and the data component.

Tip: You must declare macro variables this way in a definition. For example, to use the automatic macro variable SYSDATE9 in a definition, declare it in an MVAR statement and reference it as SYSDATE9, without an ampersand, in your PROC TEMPLATE step. If you use the ampersand, the macro resolves when the definition is compiled instead of when ODS binds the definition to the data component.

Options

text

is text that you can place in the definition to explain the macro variable's use. Text of this type becomes part of the compiled definition, which you can view with the SOURCE statement, whereas SAS comments do not.

NMVAR Statement

Defines a symbol that references a macro variable. ODS will convert 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 definition and the data component to produce an output object.

Scope: You can use the NMVAR statement in the definition of a table, column, header, or footer. A macro variable that is defined in a definition is available to that definition and to all the definitions that it contains.

Featured in: Example 3 on page 247

NMVAR *variable-1* <'text-1'> <... *variable-n* <'text-n'>>;

Required Arguments

variable

Names a macro variable to reference in the definition. ODS will convert 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 definition and the data component.

Tip: You must declare macro variables this way in a definition. For instance, 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 resolves when the definition is compiled instead of when ODS binds the definition to the data component.

Options

text

is text that you can place in the definition to explain the macro variable's use. Text of this type becomes part of the compiled definition, which you can view with the SOURCE statement, whereas SAS comments do not.

NOTES Statement

Provides information about the table.

Tip: The NOTES statement becomes part of the compiled column definition, which you can view with the SOURCE statement, whereas SAS comments do not.

Featured in: Example 3 on page 247

NOTES '*text*';

Required Arguments

text

provides information about the table.

TRANSLATE-INTO Statement

Translates the specified numeric values to other values.

Restriction: The TRANSLATE-INTO statement in a table definition applies only to numeric variables. To translate the values of a character variable, use TRANSLATE-INTO in the definition of that column. (See “DEFINE COLUMN Statement” on page 159).

Featured in: Example 3 on page 247

TRANSLATE *expression-1* INTO *expression-2* <..., *expression-n* INTO *expression-m*>;

Required Arguments

expression-1

is an expression that is evaluated for each table cell that contains a numeric variable. It can be any numeric expression that is valid in the WHERE statement (or the WHERE= data set option). For information on expressions that you can use in the WHERE statement, see “Statements” in *SAS Language Reference: Dictionary*. Use `_VAL_` to represent the value of the current column. You may also reference symbols that you declared in a DYNAMIC, MVAR, or NVAR statement in the table definition.

If *expression-1* resolves to TRUE (a non-zero 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.

Restriction: You may not 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.

expression-2

is an expression that specifies the value to use in the cell in place of the variable's actual value. It can be any expression that is valid in the WHERE statement (or the WHERE= data set option). For information on expressions that you can use in the WHERE statement, see “Statements” in *SAS Language Reference: Dictionary*. Use `_VAL_` to represent the value of the current column. You may also reference symbols that you declared in a DYNAMIC, MVAR, or NVAR statement in the table definition.

Restriction: *expression-2* must resolve to a character value, not a numeric value.

Restriction: You may not reference the values of other columns in *expression-2*.

Tip: When you translate a numeric value to a character value, the table definition does not try to apply the numeric format that is associated with the column.

Instead, it simply writes the character value into the format field, starting at the left. If you want the value to be right justified, use the JUSTIFY=ON attribute.

See also: JUSTIFY= on page 215

END Statement

Ends the table definition.

END;

DELETE Statement

Deletes the specified definitions.

DELETE *definition-path*;

Required Arguments

definition-path

specifies a definition to delete. A definition-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 definition exists in multiple template stores, PROC TEMPLATE deletes the definition from the first template store in the current path that the user can write to.

CAUTION:

Deleting a Directory in a Template Store If the path that you specify is a directory rather than a definition, PROC TEMPLATE deletes all the definitions in that directory. △

EDIT Statement

Edits an existing definition for a table, column, header, or footer

Requirement: An END statement must follow the EDIT statement, after all the editing instructions.

Interaction: In some cases, you can use an EDIT statement inside a set of editing instructions.

When you edit a table, you can also edit one or more columns, headers, or footers that are defined in the table.

When you edit a column definition, you can also edit one or more headers that are defined for that column.

Restriction: If you edit a definition that is a link, you break the link and create a separate definition.

Featured in: Example 1 on page 233

```
EDIT definition-path-1 <AS definition-path-2 > </ STORE=libname.template-store>;
  statements-and-attributes
END;
```

Required Arguments

definition-path-1

specifies a definition to edit. *definition-path-1* 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.)

There are two steps to editing: opening a copy of the specified file and saving the modified file. By default, PROC TEMPLATE looks for *definition-path-1* in the list of template stores that is defined by the PATH statement (see “PATH Statement” on page 231). It opens a copy of the first one that it finds in a template store that you can read. PROC TEMPLATE writes the modified definition to the first template store in the current path that you can write to. If you do not specify a second definition-path to write to, it uses *definition-path-1*. Therefore, if you have update access to the template store from which you read *definition-path-1*, you actually modify the original definition. Otherwise, the modified file is written to a template store to which you do have update access.

If you do specify a second definition-path, PROC TEMPLATE writes the edited definition to the specified path in the first template store to which you have write access.

Interaction: The STORE= option specifies a particular template store to read from and to write to.

Tip: You can determine what definitions a procedure or DATA step uses by submitting the ODS TRACE ON statement before you run the SAS program (see “ODS TRACE Statement” on page 71).

Options

AS *definition-path-2*

specifies the location in which to store the edited definition, where *definition-path-2* 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.) By default, PROC TEMPLATE writes the edited definition to the first template store that you can write to in the current path.

Default: If you do not specify AS *definition-path-2*, PROC TEMPLATE writes the edited definition to *definition-path-1* in the first template store that you can write to.

Restriction: You cannot use AS if the current EDIT statement is inside a set of editing instructions.

STORE=*libname.template-store*

specifies the template store in from which to read *definition-path-1* and in which to store *definition-path-2*.

Availability: Version 8 of the SAS System

Statements and Attributes

The EDIT statement supports the same statements and attributes as the DEFINE statement. For more information, see “DEFINE Statement” on page 157.

LINK Statement

Creates a link to an existing definition.

LINK *definition-path-1* *definition-path-2* </ NOTES='text';>

Creating a link to a definition has the same effect as creating a new definition that inherits its characteristics from another definition (see the discussion of PARENT= on page 216). However, using a link is more efficient than using inheritance because linking does not actually create a new definition.

Note: To maximize efficiency, PROC TEMPLATE implements any definition that consists solely of the declaration of a parent and of notes as a link. △

Required Arguments

definition-path-1

specifies the path of the definition to create. PROC TEMPLATE creates the definition in the first template store in the path that you can write to.

definition-path-2

specifies the path of the definition to link to. If the same definition 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 *definition-path-2* exists when it compiles the definition.

Options

NOTES= 'text'

specifies notes to store in the definition.

Tip: Notes of this type become part of the compiled definition, which you can view with the SOURCE statement, whereas SAS comments do not.

LIST Statement

Lists the definitions in one or more template stores.

Availability: Version 8 of the SAS System

LIST <*starting-path*></ *option(s)*>;

Options

starting-path

specifies a level within each template store where PROC TEMPLATE starts listing definitions. For instance, 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 do not specify a *starting-path*, the LIST statement lists all definitions in the specified template stores.

Restriction: This option must precede the forward slash (/) in the LIST statement.

SORT=statistic <sorting-order>

sorts the list of definitions by the specified statistic in the specified sorting order.

statistic

can be one of the following:

CREATED

is the date that the definition 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 definition that the current definition links to (see “LINK Statement” on page 229).

PATH

is the path to the current definition in the template store. (The path does not include the name of the template store).

SIZE

is the size of the definition.

TYPE

is the type of definition: COLUMN, FOOTER, HEADER, LINK, STYLE, or TABLE. If the item is not a definition, but 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 | (statistic-1 <, ... statistic-n>)

specifies the information to include in the list of definitions.

ALL

includes all available information.

(statistic-1 <, ... statistic-n>)

includes the specified information. *statistic* can be one or more of the following:

CREATED

is the date that the definition 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 definition that the current definition links to (see “LINK Statement” on page 229).

SIZE

is the size of the definition.

Default: Whether or not you specify **STATS=**, the list of definitions always includes an observation number, the path to the definition, and its type.

STORE='libname.template-store'

specifies the template store to process.

Default: all template stores in the current template path (see “PATH Statement” on page 231).

PATH Statement

Specifies which locations to search for definitions that were created by PROC TEMPLATE, as well as the order in which to search for them. This statement overrides the ODS PATH statement for the duration of the PROC TEMPLATE step.

PATH *location(s)*;

Required Arguments

location(s)

specifies one or more locations to search for definitions that were created by PROC TEMPLATE. ODS searches the locations in the order that 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 use of the PATH statement completely re-establishes the list of paths.

Each *location* has the following form:

<libname.>item-store <(READ | UPDATE | WRITE)>

<libname>.item-store

identifies an item store that contains style definitions, table definitions, or both.

(READ | UPDATE | WRITE)

specifies the access mode for the definition, where

READ

provides read-only access.

WRITE

provides write access (overwriting an existing definition) as well as read access.

UPDATE

provides update access (adding to the existing definition) as well as read access.

Default: If you do not use the PATH statement, PROC TEMPLATE uses the current path list. The path list is set by the ODS PATH statement (see “ODS PATH Statement” on page 64.) You can see this list by submitting the following statement:

```
ods path show;
```

Tip: If you want to be able to ignore all user-defined definitions, keep them in their own item stores so that you can leave them out of the list of items stores that ODS searches.

SOURCE Statement

Writes the source code for the specified definition to the SAS log.

SOURCE *definition-path* </ FILE=*'file-specification'*>;

Required Arguments

definition-path

specifies the path of the definition to display. If the same definition 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 stores definitions in compiled form. The SOURCE statement actually decompiles the definition. Because SAS comments are not compiled, comments that are in the source code do not appear when you decompile the definition. If you want to annotate your definition, use the NOTES statement inside the definition or the block of editing instructions, or use the NOTES= option in the LINK statement. These notes do become part of the compiled definition. (See “NOTES Statement” on page 225 and the discussion of the NOTES= option on page 229. You can also specify notes as quoted strings in the DYNAMIC, MVAR, NMVAR, REPLACE, and STYLE statements.)

Options

FILE=*'file-specification'*

specifies a file to write the definition to.

file-specification can be one of the following:

'external-file'

is the name of an external file to write to.

fileref

is a fileref that has been assigned to an external file. Use the FILENAME statement to assign a fileref. (For information on the FILENAME statement, see “Statements” in *SAS Language Reference: Dictionary*.)

Default: If you don't specify `FILE=`, the `SOURCE` statement writes to the SAS log.

TEST Statement

Tests the most recently created definition by binding it to the specified data set.

```
TEST DATA=SAS-data-set </ STORE=libname.template-store>;
```

Required Arguments

DATA=*SAS-data-set*

specifies the data set to bind to the most recently created definition. ODS sends this output object to all open ODS destinations.

Options

STORE=*libname.template-store*

specifies the template store in which the most recently created definition was stored. If you specify this option, the template store that you specify must match the template store in the `DEFINE` statement that created the definition.

Availability: Version 8 of the SAS System

Examples

Example 1: Customizing a Table Definition that a SAS Procedure Uses

PROC TEMPLATE features:

EDIT statement

Header attributes

JUST=

STYLE=

Table attributes

DOUBLE_SPACE=

OVERLINE=

UNDERLINE=

Other ODS features:

ODS HTML statement

ODS SELECT statement

Data set: STATEPOP on page 73

This example customizes the table definition for the Moments output object from PROC UNIVARIATE. The first program uses the table definition that SAS Institute supplies to generate both Listing and HTML output of the Moments object.

The second program

- Creates and edits a copy of the default template.
- Edits a header within the template.
- Sets column attributes to enhance the appearance of both the HTML and the Listing output.

Note: This example uses file names that may not be valid in all operating environments. To successfully run the example in your operating environment, you may need to change the file specifications. See Appendix 1, “Alternative ODS HTML Statements for Running Examples in Different Operating Environments,” on page 275. \triangle

Program 1: Using the Table Definition that SAS Provides

The OPTIONS statement controls several aspects of the Listing output. None of these options affects the HTML output.

```
options nodate pageno=1 pagesize=60 linesize=72;
```

The ODS HTML statement opens the HTML destination and creates HTML output. It sends all output objects to the external file `defaultmoments-body.htm` in the current directory. Some browsers require an extension of `.HTM` or `.HTML` on the filename.

```
ods html body='defaultmoments-body.htm';
```

The ODS SELECT statement sends one output object, **Moments**, to the open ODS destinations. Both the Listing and the HTML destinations are open. (To learn the names of the output objects, run the procedure with the ODS TRACE ON statement in effect. See Example 1 on page 73.)

```
ods select moments;
```

PROC UNIVARIATE computes the univariate statistics for one variable, `CityPop_90`. It uses the default table definition, `base.univariate.moments` from the template store `sashelp.tmplmst`.

```
proc univariate data=statepop mu0=3.5;
  var citypop_90;
  title 'Default Moments Table';
run;
```


The ODS HTML statement closes the HTML destination and all the files that are associated with it. You must close the destination before you can view the output with a browser.

```
ods html close;
```

Default Listing Output

Output 5.1 Listing Output from PROC UNIVARIATE (Default Moments Table)

Default Moments Table				1
The UNIVARIATE Procedure				
Variable: CityPop_90 (1990 metropolitan pop in millions)				
Moments				
N	51	Sum Weights		51
Mean	3.87701961	Sum Observations		197.728
Std Deviation	5.16465302	Variance		26.6736408
Skewness	2.87109259	Kurtosis		10.537867
Uncorrected SS	2100.27737	Corrected SS		1333.68204
Coeff Variation	133.21194	Std Error Mean		0.72319608

Default HTML Output

Display 5.10 HTML Output from PROC UNIVARIATE (Default Moments Table)

Default Moments Table

The UNIVARIATE Procedure
Variable: *CityPop_90* (1990 metropolitan pop in millions)

Moments			
N	51	Sum Weights	51
Mean	3.87701961	Sum Observations	197.728
Std Deviation	5.16465302	Variance	26.6736408
Skewness	2.87109259	Kurtosis	10.537867
Uncorrected SS	2100.27737	Corrected SS	1333.68204
Coeff Variation	133.21194	Std Error Mean	0.72319608

Program 2: Using a Customized Table Definition

This statement specifies which locations to search for definitions 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 sasuser.templat(update) sashelp.tmplmst(read);
```

The EDIT statement looks in the available template stores for a table definition called **base.univariate.moments**. By default, it first looks in SASUSER.TEMPLAT, but it finds nothing. Next, it looks in SASHELP.TMPLMST, which contains the table definitions that SAS Institute provides. Because EDIT can read this definition, this is the one that it uses. The program does not specify a destination for the edited definition, so PROC TEMPLATE writes to the first template store in the path that it can write to, which is SASUSER.TEMPLAT. Therefore, it creates a table definition of the same name as the original one in SASUSER.TEMPLAT. (See “ODS PATH Statement” on page 64).

(To learn the name of the table definition that a procedure uses, you run the procedure with the ODS TRACE ON statement in effect. See Example 1 on page 73).

```
proc template;
  edit base.univariate.moments;
```

These three table attributes affect the presentation of the **Moments** output object in the Listing output. They have no effect on its presentation in the HTML output. DOUBLE_SPACE= 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;
underline=on;
overline=on;
```

This EDIT statement edits the table element **head** within the table definition.

```
edit head;
```

The STYLE= attribute alters the style element that renders the **head** table element. The style element **header** is defined in the default style definition, **styles.default**. Many procedures, including PROC UNIVARIATE, use this style element to render headers for tables and columns. (For information on viewing a style definition, see “Customizing Presentation Aspects of ODS Output” on page 42.) In this case, the STYLE= attribute specifies green for the foreground color and italic for the font style. All other attributes that are included in **header** remain in effect. The STYLE= attribute affects only the HTML output.

```
style=header{foreground=green font_style=italic};
```

The JUST= attribute left-justifies the text of the header in both the Listing and the HTML output.

```
just=left;
```

The first END statement ends the editing of the the table element **head**. The second END statement ends the editing of the table **base.univariate.moments**.

```
end;
end;
run;
```

The ODS HTML statement opens the HTML destination and creates HTML output. It sends all output objects to the external file **custommoments-body.htm** in the current directory. Some browsers require an extension of .htm or .html on the filename.

```
ods html body='custommoments-body.htm';
```

The ODS SELECT statement sends one output object, **Moments**, to the open ODS destinations. Both the Listing and HTML destinations are open. (To learn the names of the output objects, run the procedure with the ODS TRACE ON statement in effect. See Example 1 on page 73.)

```
ods select moments;
```

PROC UNIVARIATE computes the univariate statistics for one variable, **CityPop_90**. This is the same PROC UNIVARIATE step that was used in “Program 1: Using the Table Definition that SAS Provides” on page 234. 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 definition. 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 Institute-supplied table definition at this point, you would have to change the path with the ODS PATH statement (see “ODS PATH Statement” on page 64).

```
proc univariate data=statepop mu0=3.5;
  var citypop_90;
  title 'Custom Moments Table';
run;
```

The ODS HTML statement closes the HTML destination and all the files that are associated with it. You must close the destination before you can view the output with a browser.

```
ods html close;
```

Customized Listing Output

Output 5.2 Listing Output from PROC UNIVARIATE (Customized Moments Table)

Custom Moments Table				1
The UNIVARIATE Procedure				
Variable: CityPop_90 (1990 metropolitan pop in millions)				
<i>Moments</i>				
N	51	Sum Weights	51	
Mean	3.87701961	Sum Observations	197.728	
Std Deviation	5.16465302	Variance	26.6736408	
Skewness	2.87109259	Kurtosis	10.537867	
Uncorrected SS	2100.27737	Corrected SS	1333.68204	
Coeff Variation	133.21194	Std Error Mean	0.72319608	

Customized HTML Output

Display 5.11 HTML Output from PROC UNIVARIATE (Customized Moments Table)

<i>Custom Moments Table</i>			
<i>The UNIVARIATE Procedure</i>			
<i>Variable: CityPop_90 (1990 metropolitan pop in millions)</i>			
<i>Moments</i>			
N	51	Sum Weights	51
Mean	3.87701961	Sum Observations	197.728
Std Deviation	5.16465302	Variance	26.6736408
Skewness	2.87109259	Kurtosis	10.537867
Uncorrected SS	2100.27737	Corrected SS	1333.68204
Coeff Variation	133.21194	Std Error Mean	0.72319608

Example 2: Creating a New Table Definition

PROC TEMPLATE features:

Table attributes:

```
DOUBLE_SPACE=
OVERLINE=
UNDERLINE=
```

DEFINE TABLE statement:

```
COLUMN statement
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:

```
ODS HTML statement
FILE statement with ODS= option
PUT statement with _ODS_ argument
```

This example creates a custom table definition for an output data set that PROC MEANS produces.

Note: This example uses file names that may not be valid in all operating environments. To successfully run the example in your operating environment, you may need to change the file specifications. See Appendix 1, "Alternative ODS HTML Statements for Running Examples in Different Operating Environments," on page 275. Δ

Program 1: Producing an Output Data Set with PROC MEANS

The OPTIONS statement controls several aspects of the Listing output. None of these options affects the HTML output.

```
options nodate pageno=1 pagesize=60 linesize=72;
```

PROC FORMAT creates formats for Year and School.

```
proc format;
  value yrFmt . = " All";
  value $schFmt ' ' = "All  ";
run;
```

The data set Charity contains information about high-school students' volunteer work for charity. The variables give the name of the high school, the year of the fundraiser, the first name of each student, the amount of money that each student raised, and the number of hours that each student volunteered. The RETAIN statement and two sum statements compute the minimum and maximum values of year. The CALL SYMPUT statements store these values in the macro variables FIRST_YEAR and LAST_YEAR. A DATA step on page 279 creates this data set.

```
data Charity;
  input School $ 1-7 Year 9-12 Name $ 14-20 moneyRaised 22-26
        hoursVolunteered 28-29;
  format moneyRaised dollar8.2;
  format hoursVolunteered f3.0;
  format Year yrFmt.;
  format School schFmt.;
  label School = "Schools";
  label Year = "Years";
  retain yearmin yearmax;
  yearmin=min(yearmin,year);
  yearmax=max(yearmax,year);
  call symput('first_year',put(yearmin,4.));
  call symput('last_year', put(yearmax,4.));
  datalines;
Monroe 1992 Allison 31.65 19
Monroe 1992 Barry 23.76 16
Monroe 1992 Candace 21.11 5

      ... more lines of data ...

Kennedy 1994 Sid 27.45 25
Kennedy 1994 Will 28.88 21
Kennedy 1994 Morty 34.44 25
;
```

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.

For a detailed explanation of a similar PROC MEANS step, see the example “Identifying the Top Three Extreme Values with the Output Statistics” in the documentation for the MEANS procedure in **SAS Procedures Guide**.

```
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;
```

This PROC PRINT step generates traditional Listing 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;
```

Listing Output from PROC PRINT

Output 5.3 PROC PRINT Listing Output for the Output Data Set from PROC MEANS

Simple PROC PRINT of the Output Data Set									1
School	Year	_TYPE_	_FREQ_	money	money	money	money	money	
				Raised_ Sum	Raised_ Mean				
Kennedy	All	10	53	\$1575.95	\$29.73	\$72.22	\$52.63	\$43.89	
Monroe	All	10	56	\$1616.80	\$28.87	\$78.65	\$65.44	\$56.87	
All	1992	01	31	\$892.92	\$28.80	\$55.16	\$53.76	\$52.63	
All	1993	01	32	\$907.92	\$28.37	\$65.44	\$47.33	\$42.23	
All	1994	01	46	\$1391.91	\$30.26	\$78.65	\$72.22	\$56.87	
All	All	00	109	\$3192.75	\$29.29	\$78.65	\$72.22	\$65.44	
Name_1	Name_2	Name_3	School_1	School_2	School_3	Year_1	Year_2	Year_3	
Luther	Thelma	Jenny	Kennedy	Kennedy	Kennedy	1994	1992	1992	
Willard	Cameron	L.T.	Monroe	Monroe	Monroe	1994	1993	1994	
Tonya	Edward	Thelma	Monroe	Monroe	Kennedy	1992	1992	1992	
Cameron	Myrtle	Bill	Monroe	Monroe	Kennedy	1993	1993	1993	
Willard	Luther	L.T.	Monroe	Kennedy	Monroe	1994	1994	1994	
Willard	Luther	Cameron	Monroe	Kennedy	Monroe	1994	1994	1993	

Program 2: Building a Custom Table Definition for the TopN Report

The OPTIONS statement controls several aspects of the Listing output. None of these options affects the HTML output.

```
options nodate pageno=1 pagesize=60 linesize=72;
```

The ODS HTML statement opens the HTML destination and creates HTML output. It sends all output objects to the external file `topn-body.htm` in the current directory. Some browsers require an extension of `.htm` or `.html` on the filename.

```
column name (city state) * (homephone workphone);
```

```
ods html body='topn-body.htm';
```

The DEFINE statement creates the table definition `means.topn` 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 means.topn;
```

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 definition 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;
```

The COLUMN statement declares these symbols as columns in the table and specifies their order in the table. If a column name appears in parentheses, PROC TEMPLATE stacks the values of all variables that use that column definition one below the other in the output object. Variables are assigned a column definition in the DATA step that appears later in the program.

```
column class sum mean (raised) (name) (school) (year);
```

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= double spaces 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;
```

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;
```

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 renders the table header. The style element `header` is defined in the default style definition, `styles.default`. (For information on viewing a style definition, see “Customizing Presentation Aspects of ODS Output” on page 42.) 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 definition.

```
define table_header_1;
  text "Top Three Fund Raisers";
  style=header{font_size=6};
end;
```

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 definition (in the DATA step that follows), it will resolve the values of the macro variables FIRST_YEAR and LAST_YEAR. The table definition 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 definition.

```
define table_header_2;
  text "from " first_year " to " last_year;
  space=1;
end;
```


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 definition (in the `DATA` step that follows), it will resolve the value of the macro variable `SYSDATE9`. The table definition 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 were specified, ODS would interpret the nonalphabetic, leading character as the split character (see the discussion of *text-specification(s)* in “TEXT Statement” on page 187.) Alternatively, you could place a space character before the open parenthesis.

The `STYLE=` attribute alters the style element that renders the table header. The style element `header` is defined in the default style definition, `styles.default`. (For information on viewing a style definition, see “Customizing Presentation Aspects of ODS Output” on page 42.) In this case, the `STYLE=` attribute specifies a small 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 footer definition.

```
define footer table_footer;
  text "(report generated on " sysdate9 ")";
  split="*";
  style=header{font_size=2};
end;
```

The `DEFINE` statement and its attributes create the column definition `class`. (The `COLUMN` statement earlier in the program declared `class` as a column.) `GENERIC=` specifies that multiple variables can use the same column definition. `ID=` specifies that this column should be repeated on every data panel if the report uses multiple data panels. `VJUST=` specifies that the text appear at the top of the HTML table cell that it is in. `STYLE=` specifies that the column uses the `DATA` table element. This table element is defined in the default style definition, which is the style definition that is being used. `VJUST=` and `STYLE=` affect only the HTML output. `ID=` affects only the Listing output. `GENERIC=` is not specific to a destination. The `END` statement ends the definition.

Notice that, unlike subsequent column definitions, this column definition does not include a header. This is because the same header is not appropriate for all the variables that use this column definition. 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;
```

Each of these `DEFINE` statements and its attributes creates a column definition. `GENERIC=` specifies that multiple variables can use a column definition (although in the case of `sum` and `mean`, only one variable uses the definition). `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 definition.

```
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;

```

This END statement ends the table definition. The RUN statement ends the PROC TEMPLATE step.

```

        end;
    run;

```

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, which PROC MEANS created.

```

data _null_;
    set top3list;

```

The combination of the fileref PRINT and the ODS option in the FILE statement routes the results of the DATA step to ODS. (For more information on using the DATA step with ODS, see Chapter 4, “Using the Output Delivery System in the DATA Step,” on page 105.) The TEMPLATE= suboption tells ODS to use the table definition named `means.topn`, which was just created with PROC TEMPLATE.

```

        file print ods = (
            template='means.topn'

```

The COLUMNS= suboption places DATA step variables into columns that are defined in the table definition. For instance, 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 definition named `class`. GENERIC= must be set to ON in both the table definition and each column assignment in order for multiple variables to use the same column definition.

```

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)
)
);

```

The `_ODS_` option and the `PUT` statement write the data values for all columns to the data component.

```

put _ods_;
run;

```

The `ODS HTML` statement closes the HTML destination and all the files that are associated with it. You must close the destination before you can view the output with a browser.

```

ods html close;

```

Listing Output for the TopN Report

Compare this customized output to the `PROC PRINT` Listing output in Output 5.3 on page 241.

Output 5.4 Using a Customized Table to Produce Listing Output

1

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	Luther	Kennedy	1994
				\$52.63	Thelma	Kennedy	1992
				\$43.89	Jenny	Kennedy	1992
Monroe	All	\$1616.80	\$28.87	\$78.65	Willard	Monroe	1994
				\$65.44	Cameron	Monroe	1993
				\$56.87	L.T.	Monroe	1994
All	1992	\$892.92	\$28.80	\$55.16	Tonya	Monroe	1992
				\$53.76	Edward	Monroe	1992
				\$52.63	Thelma	Kennedy	1992
All	1993	\$907.92	\$28.37	\$65.44	Cameron	Monroe	1993
				\$47.33	Myrtle	Monroe	1993
				\$42.23	Bill	Kennedy	1993
All	1994	\$1391.91	\$30.26	\$78.65	Willard	Monroe	1994
				\$72.22	Luther	Kennedy	1994
				\$56.87	L.T.	Monroe	1994
All	All	\$3192.75	\$29.29	\$78.65	Willard	Monroe	1994
				\$72.22	Luther	Kennedy	1994
				\$65.44	Cameron	Monroe	1993

(report generated on 12MAR1999)

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	Luther	Kennedy	1994
				\$52.63	Thelma	Kennedy	1992
				\$43.89	Jenny	Kennedy	1992
Monroe	All	\$1616.80	\$28.87	\$78.65	Willard	Monroe	1994
				\$65.44	Cameron	Monroe	1993
				\$56.87	L.T.	Monroe	1994
All	1992	\$892.92	\$28.80	\$55.16	Tonya	Monroe	1992
				\$53.76	Edward	Monroe	1992
				\$52.63	Thelma	Kennedy	1992
All	1993	\$907.92	\$28.37	\$65.44	Cameron	Monroe	1993
				\$47.33	Myrtle	Monroe	1993
				\$42.23	Bill	Kennedy	1993
All	1994	\$1391.91	\$30.26	\$78.65	Willard	Monroe	1994
				\$72.22	Luther	Kennedy	1994
				\$56.87	L.T.	Monroe	1994
All	All	\$3192.75	\$29.29	\$78.65	Willard	Monroe	1994
				\$72.22	Luther	Kennedy	1994
				\$65.44	Cameron	Monroe	1993

(report generated on 12MAR1999)

Example 3: Setting the Style Element for Cells Based on Their Values

PROC TEMPLATE 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:

- ODS HTML statement
- FILE statement with ODS= option
- PUT statement with _ODS_ argument

Data set: GRAIN_PRODUCTION on page 93

Format: \$CNTRY. on page 94

This example creates a template that uses different colors for the text inside cells, depending on their values.

Note: This example uses file names that may not be valid in all operating environments. To successfully run the example in your operating environment, you may need to change the file specifications. See Appendix 1, “Alternative ODS HTML Statements for Running Examples in Different Operating Environments,” on page 275. Δ

Program

The OPTIONS statement controls several aspects of the Listing output. None of these options affects the HTML output. The TITLE statement specifies a title.

```
options nodate pageno=1 pagesize=60 linesize=72;
title 'Leading Grain Producers';
```

The DEFINE statement creates the table definition `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;
```

The TRANSLATE-INTO statement translates missing values (.) into the string `No data`.

```
translate _val_=. into 'No data';
```

The NOTES statements provides information about the table. NOTES statements remain a part of the compiled table definition whereas SAS comments do not.

```
notes "NMVAR defines symbols that will be used";
notes "to determine the colors of the cells.";
```

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 definition 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 definition whereas SAS comments do not.

LOW, MEDIUM, and HIGH will contain the values to use as the determinants of the style element that is used to render the cell. The values are provided just before the DATA step that produces the report.

```
nmvar low 'Use default style.'
      medium 'Use yellow foreground and bold font weight'
      high 'Use red foreground and a bold, italic font.';
```

The CLASSLEVELS= attribute suppresses the blanking 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, it is wise to set this attribute as well.

```
classlevels=on;
```

The DEFINE statement and its attributes create the column definition **char_var**. GENERIC= specifies that multiple variables can use the same column definition. 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 definition.

```
define column char_var;
  generic=on;
  blank_dups=on;
end;
```

The DEFINE statement and its attributes create the column definition **num_var**. GENERIC= specifies that multiple variables can use the same column definition.

```
define column num_var;
  generic=on;
```

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 definition, the value **No data** is formatted as a character variable (left justified) within a format field that is the same width as the column.

```
justify=on;
```

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.

The `END` statement ends the column definition.

```

cellstyle _val_ <= low as data,
          _val_ <= medium as data
            {foreground=green font_style=italic},
          _val_ <= high as data
            {foreground=yellow font_weight=bold},
          1 as data
            {foreground=red font_style=italic
             font_weight=bold};

end;
```

This `END` statement ends the table definition. The `RUN` statement ends the `PROC TEMPLATE` step.

```

end;
run;
```

The `ODS HTML` statement opens the HTML destination and creates HTML output. It sends all output objects to the external file `cellstyle-body.htm` in the current directory. Some browsers require an extension of `.htm` or `.html` on the filename.

```
ods html body='cellstyle-body.htm';
```

The `%LET` statements assign values to the macro variables `LOW`, `MEDIUM`, and `HIGH`.

```
%let low=10000;
%let medium=50000;
%let high=100000;
```

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;
```

The combination of the fileref `PRINT` and the `ODS` option in the `FILE` statement routes the results of the `DATA` step to `ODS`. (For more information on using the `DATA` step with `ODS`, see Chapter 4, “Using the Output Delivery System in the `DATA` Step,” on page 105.) The `TEMPLATE=` suboption tells `ODS` to use the table definition named `shared.cellstyle`, which was just created with `PROC TEMPLATE`.

```
file print ods=(
  template='shared.cellstyle'
```

The COLUMNS= suboption places DATA step variables into columns that are defined in the table definition. For instance, 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 definition named `char_var`. GENERIC= must be set to ON, both in the table definition and in each column assignment, in order for multiple variables to use the same column definition.

```
columns=(
  char_var=year(generic=on)
  char_var=country(generic=on format=$centry.)
  char_var=type(generic=on)
  num_var=kilotons(generic=on format=comma12.)
)
);
```

The _ODS_ option and the PUT statement write the data values for all columns to the data component.

```
put _ods_;
run;
```

The ODS HTML statement closes the HTML destination and all the files that are associated with it. You must close the destination before you can view the output with a browser.

```
ods html close;
```

Listing Output

Output 5.5 Listing Output of a Customized Table

Only the table customizations appear in the Listing 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 that are specified in the CELLSTYLE-AS statement do not appear in the Listing output.

Leading Grain Producers				1
Year	Country	Type	Kilotons	
1995	Brazil	Corn	36,276	
		Rice	11,236	
		Wheat	1,516	
	China	Corn	112,331	
		Rice	185,226	
		Wheat	102,207	
	India	Corn	9,800	
		Rice	122,372	
		Wheat	63,007	
	Indonesia	Corn	8,223	
		Rice	49,860	
		Wheat	No data	
United States	Corn	187,300		
	Rice	7,888		
	Wheat	59,494		
1996	Brazil	Corn	31,975	
		Rice	10,035	
		Wheat	3,302	
	China	Corn	119,350	
		Rice	190,100	
		Wheat	109,000	
	India	Corn	8,660	
		Rice	120,012	
		Wheat	62,620	
	Indonesia	Corn	8,925	
		Rice	51,165	
		Wheat	No data	
United States	Corn	236,064		
	Rice	7,771		
	Wheat	62,099		

HTML Output

Display 5.12 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.

Year	Country	Type	Kilotons
1995	Brazil	Corn	36,276
		Rice	11,236
		Wheat	1,516
	China	Corn	112,331
		Rice	185,226
		Wheat	102,207
	India	Corn	9,800
		Rice	122,372
		Wheat	63,007
	Indonesia	Corn	8,223
		Rice	49,860
		Wheat	No data
	United States	Corn	187,300
		Rice	7,888
		Wheat	59,494
1996	Brazil	Corn	31,975

Example 4: Creating a Stand-alone Style Definition

PROC TEMPLATE features:

DEFINE STYLE statement

STYLE statement

BACKGROUND=
 BORDERCOLORDARK=
 BORDERCOLORLIGHT=
 BORDERWIDTH=
 CELLSPACING=
 FONT_FACE=
 FONT_SIZE=
 FONT_STYLE=
 FONT_WEIGHT=
 FOREGROUND=

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
 ODS LISTING statement
 FILE statement with ODS= option
 PUT statement with `_ODS_` argument

Data set: GRAIN_PRODUCTION on page 93

Format: \$COUNTRY. on page 94

This example creates a style definition that is not based on any other style definition. In general, when you create a style definition, you are likely to base it on one of the definitions that SAS Institute provides (see Example 6 on page 265). However, this example is provided to show you some of the basics of creating a style definition.

It is important to understand that by default, certain table elements are rendered with certain style elements. For instance, unless you specify a different style element with the `STYLE=` attribute, ODS renders SAS titles with the `systemtitle` style element. Similarly, unless you specify otherwise, ODS renders headers with the `header` style element. (For information of what each style element does, see “What Is the Default Style Definition Like?” on page 133.)

Program

The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style definition called `newstyle`. This STYLE statement defines the style element `cellcontents`. This style element is composed of the style attributes that appear on the STYLE statement. The `FONT_FACE=` 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 /
      background=blue
      foreground=white
      font_face="arial, helvetica"
      font_weight=medium
      font_style=roman
      font_size=4;
```

This STYLE statement creates the style element **header**. By default, ODS uses **header** to render both spanning headers and column headers. This style element uses different foreground and background colors from **cellcontents**. It uses the same font face (Arial or Helvetica) and the same font_style (roman) as **cellcontents**. However, it uses a bold font weight and a large font size.

```
style header /
  background=very light blue
  foreground=blue
  font_face="arial, helvetica"
  font_weight=bold
  font_style=roman
  font_size=5;
```

This STYLE statement creates the style element **systemtitle**. By default, ODS uses **systemtitle** to render SAS titles. This style element uses a color scheme of a red foreground on a white background. It uses the same font face and font_weight as **header**, but it adds an italic font style and uses a larger font size.

```
style systemtitle /
  background=white
  foreground=red
  font_face="arial, helvetica"
  font_weight=bold
  font_style=italic
  font_size=6;
```

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 FONT_SIZE= attribute in its definition.

```
style footer from systemtitle /
  font_size=3;
```

This STYLE statement creates the style element **table**. By default, ODS uses this style element to render tables.

```
style table /
  cellspacing=5
  borderwidth=10
  bordercolorlight=very light blue
  bordercolordark=blue;
```

The END statement ends the style definition. The RUN statement executes the TEMPLATE procedure.

```
end;
run;
```

The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE TABLE statement creates a new table definition called **table1**.

```
proc template;
  define table table1;
```

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 definition to the data component to produce an output object. `SYSDATE9` is an automatic macro variable whose value is always available.

```
mvar sysdate9;
```

The DYNAMIC statement defines a symbol, `colhd`, that references a value that the data component supplies when ODS binds the definition 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 headers gives you more flexibility than does hard-coding the headers in the table definition.

```
dynamic colhd;
```

The CLASSLEVELS= attribute suppresses the blanking 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, it is wise to set this attribute as well.

```
classlevels=on;
```

This DEFINE statement and its attributes create the column definition `char_var`. `GENERIC=` specifies that multiple variables can use the same column definition. `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 definition is `cellcontents`.

The END statement ends the definition.

```
define column char_var;
  generic=on;
  blank_dups=on;
  header=colhd;
  style=cellcontents;
end;
```

This DEFINE statement and its attributes create the column definition `num_var`. `GENERIC=` specifies that multiple variables can use the same column definition. `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 definition is `cellcontents`.

The END statement ends the definition.

```
define column num_var;
  generic=on;
  header=colhd;
  style=cellcontents;
end;
```

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 definition (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;

```

This END statement ends the table definition. The RUN statement executes the PROC TEMPLATE step.

```

end;
run;

```

The ODS LISTING statement closes the Listing destination to conserve resources. The Listing destination is open by default.

```
ods listing close;
```

The ODS HTML statement opens the HTML destination and creates HTML output. It sends all output objects to the external file `newstyle-body.htm` in the current directory. The `STYLE=` option tells ODS to use `newstyle` as the style definition when it formats the output.

```
ods html body='newstyle-body.htm'
        style=newstyle;
```

The TITLE statements provide two titles for the output.

```

title 'Leading Grain Producers';
title2 'in 1996';

```

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. (This is done simply to make the entire table visible in the screen dumps that are used in the hardcopy book.)

```

data _null_;
    set grain_production;
    where type in ('Rice', 'Corn') and year=1996;

```

The combination of the fileref `PRINT` and the ODS option in the FILE statement routes the results of the DATA step to ODS. (For more information on using the DATA step with ODS, see Chapter 4, “Using the Output Delivery System in the DATA Step,” on page 105.) The `TEMPLATE=` suboption tells ODS to use the table definition named `table1`, which was just created with PROC TEMPLATE.

```

file print ods=(
    template='table1'

```

The `COLUMNS=` suboption places DATA step variables into columns that are defined in the table definition. For instance, 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 definition named `char_var`. `GENERIC=` must be set to `ON` in both the table definition and each column assignment in order for multiple variables to use the same column definition. 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 definition, the column header is `Year`.

```
columns=(
  char_var=country(generic=on format=$centry.
                  dynamic=(colhd='Country'))
  char_var=type(generic dynamic=(colhd='Year'))
  num_var=kilotons(generic=on format=comma12.
                  dynamic=(colhd='Kilotons'))
)
);
```

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;
```

The `ODS HTML` statement closes the `HTML` destination and all the files that are associated with it. You must close the destination before you can view the output with a browser. The `ODS LISTING` statement opens the `Listing` destination to return `ODS` to its default setup.

```
ods html close;
ods listing;
```

HTML Output

Display 5.13 Specifying Colors and Fonts with User-Defined Attributes

You can use the fonts to confirm that SAS titles use the `systemtitle` style element, that column headers 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 rendered with the `table` style element.

Leading Grain Producers in 1996

Country	Year	Kilotons
Brazil	Corn	31,975
	Rice	10,035
China	Corn	119,350
	Rice	190,100
India	Corn	8,660
	Rice	120,012
Indonesia	Corn	8,925
	Rice	51,165
United States	Corn	236,064
	Rice	7,771
<i>Prepared on 10FEB1999</i>		

Example 5: Creating and Modifying a Style Definition with User-Defined Attributes

PROC TEMPLATE 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 COLUMN statement
  BLANK_DUPS= attribute
CELLSTYLE-AS statement
  GENERIC= attribute
DEFINE FOOTER statement
  TEXT statement
Other ODS features:
  ODS HTML statement
  ODS LISTING statement
  FILE statement with ODS= option
  PUT statement with _ODS_ argument

Data set:  GRAIN_PRODUCTION on page 93
Format:   $CNTRY. on page 94

```

This example creates a style definition that is equivalent to the style definition that Example 4 on page 252 creates. However, this style definition uses user-defined attributes to specify colors and fonts. This technique makes it possible to easily make changes in multiple places in your output.

Program 1: Creating the Style Definition

The PROC *TEMPLATE* statement starts the *TEMPLATE* procedure. The *DEFINE STYLE* statement creates a new style definition called **newstyle2**. This *STYLE* statement defines the style element **fonts**. This style element is composed of three user-defined attributes: **cellfont**, **headingfont**, and **titlefont**. Each of these attributes describes a font.

```

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);

```

This *STYLE* statement defines the style element **colors**. This style element is composed 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;

```

These STYLE statements create four style elements. Each style element is composed of attributes that define the foreground and background colors as well as the font. Notice that the style attributes are defined in terms of the user-defined attributes that were created earlier in the style definition. For example, the foreground color in `cellcontents` is set to `colors("light")`. Looking at the definition 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 /
  background=colors("dark")
  foreground=colors("light")
  font=fonts("cellfont");
style header /
  background=colors("medium")
  foreground=colors("dark")
  font=fonts("headingfont");
style systemtitle /
  background=colors("light")
  foreground=colors("bright")
  font=fonts("titlefont");
style footer from systemtitle /
  font_size=3;
style table /
  cellspacing=5
  borderwidth=10
  bordercolorlight=colors("medium")
  bordercolordark=colors("dark");

```

The END statement ends the style definition. The RUN statement executes PROC TEMPLATE.

```

end;
run;

```

The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE TABLE statement creates a new table definition called `table1`.

```

proc template;
  define table table1;

```

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 definition to the data component to produce an output object. SYSDATE9 is an automatic macro variable whose value is always available.

```

mvar sysdate9;

```

The DYNAMIC statement defines a symbol, `colhd`, that references a value that the data component supplies when ODS binds the definition 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 headers gives you more flexibility than hard-coding the headers in the table definition does.

```

dynamic colhd;

```

The `CLASSLEVELS=` attribute suppresses the blanking 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, it is wise to set this attribute as well.

```
classlevels=on;
```

This `DEFINE` statement and its attributes create the column definition `char_var`. `GENERIC=` specifies that multiple variables can use the same column definition. `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 definition is `cellcontents`.

The `END` statement ends the definition.

```
define column char_var;
  generic=on;
  blank_dups=on;
  header=colhd;
  style=cellcontents;
end;
```

This `DEFINE` statement and its attributes create the column definition `num_var`. `GENERIC=` specifies that multiple variables can use the same column definition. `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 definition is `cellcontents`.

The `END` statement ends the definition.

```
define column num_var;
  generic=on;
  header=colhd;
  style=cellcontents;
end;
```

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 definition (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;
```

This `END` statement ends the table definition. The `RUN` statement executes the `PROC TEMPLATE` step.

```
end;
run;
```

The `ODS LISTING` statement closes the Listing destination to conserve resources. The Listing destination is open by default.

```
ods listing close;
```

The ODS HTML statement opens the HTML destination and creates HTML output. It sends all output objects to the external file `newstyle2-body.htm` in the current directory. The `STYLE=` option tells ODS to use `newstyle` as the style definition when it formats the output.

```
ods html body='newstyle2-body.htm'
       style=newstyle2;
```

The `TITLE` statements provide two titles for the output.

```
title 'Leading Grain Producers';
title2 'in 1996';
```

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. (This is done simply to make the entire table visible in the screen dumps that are used in the hardcopy book.)

```
data _null_;
  set grain_production;
  where type in ('Rice', 'Corn') and year=1996;
```

The combination of the `fileref PRINT` and the `ODS` option in the `FILE` statement routes the results of the `DATA` step to ODS. (For more information on using the `DATA` step with ODS, see Chapter 4, “Using the Output Delivery System in the `DATA` Step,” on page 105. The `TEMPLATE=` suboption tells ODS to use the table definition named `table1`, which was just created with `PROC TEMPLATE`.

```
file print ods=(
  template='table1'
```

The `COLUMNS=` suboption places `DATA` step variables into columns that are defined in the table definition. For instance, 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 definition named `char_var`. `GENERIC=` must be set to `ON` in both the table definition and each column assignment in order for multiple variables to use the same column definition. 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 definition, 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'))
)
);
```

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;

```

The ODS HTML statement closes the HTML destination and all the files that are associated with it. You must close the destination before you can view the output with a browser. The ODS LISTING statement opens the Listing destination to return ODS to its default setup.

```

ods html close;
ods listing;

```

HTML Output

Display 5.14 Specifying Colors and Fonts with User-Defined Attributes

This HTML output is identical to Display 5.13 on page 258, which was produced with a style definition that used predefined style attributes. You can use the fonts to confirm that SAS titles use the **systemtitle** style element, that column headers 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 rendered with the **table** style element.

Leading Grain Producers in 1996

Country	Year	Kilotons
Brazil	Corn	31,975
	Rice	10,035
China	Corn	119,350
	Rice	190,100
India	Corn	8,660
	Rice	120,012
Indonesia	Corn	8,925
	Rice	51,165
United States	Corn	236,064
	Rice	7,771
<i>Prepared on 10FEB1999</i>		

Program 2: Changing User-Defined Attributes

In the program in Example 4 on page 252, if you want to change the color scheme so that the blues are replaced by pink and red, you must change each occurrence of “blue” and “very light blue”. In this program, because colors are defined as user-defined attributes, you need to make the change only once. To make this change, you need only change the following section of code from

```

style colors /
  "light"=white
  "medium"=cxaaaaaff
  "dark"=cx0000ff
  "bright"=red;

```

to

```

style colors /
  "light"=white
  "medium"=pink
  "dark"=red
  "bright"=red;

```

Similarly, to change the font in any style element that uses `cellfont`, you can change the following section of code from

```
"cellfont"=("arial, helvetica", 4, medium roman)
```

to

```
"cellfont"=("courier, arial, helvetica", 4, medium roman)
```

The following HTML output shows the results of running the same program with these changes.

HTML Output

Display 5.15 Changing Colors and Fonts with User-Defined Attributes

You can see that the font face that is used 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.

Leading Grain Producers in 1996

Country	Year	Kilotons
Brazil	Corn	31,975
	Rice	10,035
China	Corn	119,350
	Rice	190,100
India	Corn	8,660
	Rice	120,012
Indonesia	Corn	8,925
	Rice	51,165
United States	Corn	236,064
	Rice	7,771
<i>Prepared on 10FEB1999</i>		

Example 6: Modifying the Default Style Definition for the HTML Destination

PROC TEMPLATE features:

DEFINE STYLE statement

PARENT= attribute

REPLACE statement

 style attributes

 user-defined attributes

BACKGROUND=

BORDERCOLORDARK=

BORDERCOLORLIGHT=

BORDERWIDTH=

CELLPADDING=

```

CELLSPACING=
FONT=
FONT_STYLE=
FOREGROUND=
FRAME=
POSTHTML=
RULES=

```

Other ODS features:

```

ODS HTML statement
    STYLE= option
ODS LISTING statement
ODS PATH statement

```

Data set: ENERGY on page 81

Formats: DIVFMT. and USETYPE. on page 81

Generally, when you are working with style definitions, you are more likely to modify a style definition that SAS Institute supplies than to write a completely new style definition. This example shows you how to make changes to the default style definition for the HTML destination. The new style definition affects both the body file and the contents file in the HTML output. In particular, in the body file, it makes changes to

- 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 headers. 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.

In the contents file, the style definition makes changes to

- 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.

Note: Remember that when a STYLE statement creates a style element in the new style definition, only style elements that explicitly inherit from that style element in the new definition inherit the change. When a REPLACE statement creates a style element in the new style definition, all style elements that inherit from that element inherit the definition that is in the new style definition, so the change appears in all children of the element. \triangle

Program 1: Using the Default Style Definition with PROC PRINT

This statement specifies which locations to search for definitions 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 sasuser.templat(update) sashelp.tmplmst(read);
```

The ODS LISTING statement closes the Listing destination to conserve resources. The Listing destination is open by default.

```
ods listing close;
```

The ODS HTML statement opens the HTML destination and creates HTML output. The output from PROC PRINT goes 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 **styles.default** as the style definition when it formats the output. Strictly speaking, this option is unnecessary because it specifies the default style definition, but it is included for clarity.

```
ods html body='sasdefaultstyle-body.htm'
         contents='sasdefaultstyle-content.htm'
         frame='sasdefaultstyle-frame.htm'
         style=styles.default;
```

The TITLE and FOOTNOTE statements provide two titles and a footnote for the output. The FOOTNOTE statement uses double rather than single quotes so that the macro variable resolves.

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
footnote "Report prepared on &sysdate9";
```

PROC PRINT creates a report that includes three variables. ODS writes the report to the BODY file.

```
proc print data=energy noobs;
  var state type expenditures;
  format division divfmt. type usetype. expenditures comma12.;
  by division;
  where division=2 or division=3;
run;
```

HTML Output from PROC PRINT with the Default Style Definition

Display 5.16 HTML Output from PROC PRINT with the Default Style Definition

Energy Expenditures for Each Region (millions of dollars)		
Division=Middle Atlantic		
State	Type	Expenditures
NY	Residential Customers	8,786
NY	Business Customers	7,825
NJ	Residential Customers	4,115
NJ	Business Customers	3,558
PA	Residential Customers	6,478
PA	Business Customers	3,695
Division=Mountain		
State	Type	Expenditures
MT	Residential Customers	322
MT	Business Customers	232
ID	Residential Customers	392
ID	Business Customers	298

Program 2: Modifying the Default Style Definition and Using It with PROC PRINT

The PROC TEMPLATE statement starts the TEMPLATE procedure. The DEFINE STYLE statement creates a new style definition called **customdefault**.

```
proc template;
  define style customdefault;
```

PARENT= specifies *styles.default* as the style definition that the current style definition inherits from. All the style elements and attributes and statements that are specified in the parent's definition are used in the current definition unless the current definition overrides them.

```
  parent=styles.default;
```

This REPLACE statement adds to the child style definition the style element *color_list*, which also exists in the parent style definition. You can think of the REPLACE statement as replacing the definition of *color_list* in the parent style definition. The REPLACE statement doesn't actually change the parent style definition, but PROC TEMPLATE builds the child style definition as if it had changed the parent. All style elements that use the user-defined attributes that *color_list* defines (**fgB2**, **fgB1**, and so forth) use the attributes that are specified in the REPLACE statement, not the ones that are specified in *styles.default*. Therefore, if you change a color here, you change every occurrence of the color in the HTML output. This REPLACE statement changes the values of **fgA2** and **fgA** from a greenish blue to a pure blue and from a slightly darker greenish blue to a purple. (The first two digits of the hex value represent red, the next two represent green, and the last two represent blue.)

```
replace color_list /
  'fgB2' = cx0066AA
  'fgB1' = cx004488
  'fgA4' = cxAAFFAA
  'bgA4' = cx880000
  'bgA3' = cxD3D3D3
/* changed from cx0033AA */
  'fgA2' = cx0000FF
  'bgA2' = cxB0B0B0
  'fgA1' = cx000000
  'bgA1' = cxF0F0F0
/* changed from cx002288 */
  'fgA' = cx660099
  'bgA' = cxE0E0E0;
```

This REPLACE statement adds to the child style definition the style element *titlesandfooters*, which also exists in the parent style definition. The new definition does not inherit attributes from any style element, but it will pass its attributes to any style element that inherits from **titlesandfooters** or from a child of **titlesandfooters**. This style element uses **systitlefg** and **systitlebg** for colors, but it changes the font size from the relative size of 4 that is specified in **titlefont2** to a relative size of 3. As a result, the titles and footnotes in Display 5.17 on page 272 are smaller than the ones in Display 5.16 on page 268.

```
replace titlesandfooters /
  foreground=colors("systitlefg")
  background=colors("systitlebg")
  font=fonts("titlefont2") font_size=3;
```

This REPLACE statement adds to the child style definition the style element **byline**, which also exists in the parent style definition. This style element inherits all attributes from **titlesandfooters** as it is specified in the previous REPLACE statement. Therefore, the initial definition for the byline includes the foreground and background colors that are used for system titles, and a smaller version of **titlefont2**. However, the FOREGROUND= attribute replaces the foreground color with the foreground color that is used for headers. Note that in the default style definition, the background color for the byline differs from the background color for the document, so it appears as a gray stripe in Display 5.16 on page 268. In this customized style definition, the stripe disappears because the background color for the byline and the document are the same.

```
replace byline from titlesandfooters /
  foreground=colors("headerfg");
```

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 definition. The change does not affect **headerfixed** and the other style elements that inherit from **header** in the parent style definition.

```
style header from header /
    font_style=italic;
```

This REPLACE statement alters the text that is used in parts of the HTML output. In the contents file, the default style definition uses “The” as the value of **prefix1** and “Procedure” as the value of **suffix1**. Thus, in HTML output that uses the default style definition, the output from PROC PRINT is identified by “1. The PRINT Procedure” (see Display 5.16 on page 268). In the customized style definition, 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 definition in a variety of circumstances.)

```
replace text /
    "prefix1" = "PROC "
    "suffix1" = ":"
    "Content Title" = "Contents"
    "Pages Title" = "Pages"
    "Note Banner" = "Note:"
    "Warn Banner" = "Warning:"
    "Error Banner" = "Error:"
    "Fatal Banner" = "Fatal:"
    ;
```

This STYLE statement changes the presentation of the HTML table that contains the output from PROC PRINT. The background color, the kind of box that surrounds the table, and the cell padding remain the same as in **styles.default**, but all the other attributes are changed. **RULES=COLS** draws rules only between the columns of the table. **CELLSPACING=0** removes the spacing between the cells of the table so that the data appear on a continuous background. The border color that the default style definition uses is replaced by **BORDERCOLORDARK=** and **BORDERCOLORLIGHT=**, and **BORDERWIDTH=** increases the width of the table’s border. The changes dramatically alter the appearance of the HTML output.

```
style table from table /
    rules=cols
    cellspacing=0
    bordercolorlight=colors("headerfg")
    bordercolordark=colors("systitlefg")
    borderwidth=5;
```

This STYLE statement changes the value of the **VISITEDLINKCOLOR=** attribute in the style element **contents** so that the links in the table of contents appear in the same color as the rest of the table of contents. It also changes the foreground color so that the title of the table of contents appears in the same color as system titles.

```
style contents from contents /
    visitedlinkcolor=colors("systitlefg")
    foreground = colors('systitlefg');
```

This STYLE statement adds the POSTHTML= attribute so that the items in the table of contents are displayed with extra space between them.

```
style contentitem from contentitem /
  posthtml='<p>';
```

The END statement ends the style definition. The RUN statement executes the PROC TEMPLATE step.

```
end;
run;
```

The ODS HTML statement opens the HTML destination and creates HTML output. The output from PROC PRINT goes 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 definition when it formats the output.

```
ods html body='customdefaultstyle-body.htm'
  contents='customdefaultstyle-content.htm'
  frame='customdefaultstyle-frame.htm'
  style=customdefault;
```

The TITLE and FOOTNOTE statements provide two titles and a footnote for the output. The FOOTNOTE statement uses double rather than single quotes so that the macro variable resolves.

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
footnote "Report prepared on &sysdate9";
```

PROC PRINT creates a report that includes three variables. ODS writes the report to the body file. This PROC PRINT step is the same one that was used with the default style definition earlier.

```
proc print data=energy noobs;
  var state type expenditures;
  format division divfmt. type usetype. expenditures comma12.;
  by division;
  where division=2 or division=3;
run;
```

The ODS HTML statement closes the HTML destination and all the files that are associated with it. You must close the destination before you can view the output with a browser. The ODS LISTING statement opens the Listing destination to return ODS to its default setup.

```
ods html close;
ods listing;
```

HTML Output from PROC PRINT with the Customized Style Definition

Display 5.17 HTML Output from PROC PRINT with the Customized Style Definition

<p>Contents</p> <p>1. PROC Print : -Data Set <u>WORK.ENERGY</u></p> <p> -<u>Division=Middle Atlantic</u></p> <p> -<u>Division=Mountain</u></p>	<p style="text-align: center;"><i>Energy Expenditures for Each Region (millions of dollars)</i></p> <p style="text-align: center;"><i>Division=Middle Atlantic</i></p> <table border="1" style="width: 100%; border-collapse: collapse; margin-bottom: 10px;"> <thead> <tr> <th style="text-align: left;">State</th> <th style="text-align: left;">Type</th> <th style="text-align: right;">Expenditures</th> </tr> </thead> <tbody> <tr><td>NY</td><td>Residential Customers</td><td style="text-align: right;">8,786</td></tr> <tr><td>NY</td><td>Business Customers</td><td style="text-align: right;">7,825</td></tr> <tr><td>NJ</td><td>Residential Customers</td><td style="text-align: right;">4,115</td></tr> <tr><td>NJ</td><td>Business Customers</td><td style="text-align: right;">3,558</td></tr> <tr><td>PA</td><td>Residential Customers</td><td style="text-align: right;">6,478</td></tr> <tr><td>PA</td><td>Business Customers</td><td style="text-align: right;">3,695</td></tr> </tbody> </table> <p style="text-align: center;"><i>Division=Mountain</i></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">State</th> <th style="text-align: left;">Type</th> <th style="text-align: right;">Expenditures</th> </tr> </thead> <tbody> <tr><td>MT</td><td>Residential Customers</td><td style="text-align: right;">322</td></tr> <tr><td>MT</td><td>Business Customers</td><td style="text-align: right;">232</td></tr> <tr><td>ID</td><td>Residential Customers</td><td style="text-align: right;">392</td></tr> <tr><td>ID</td><td>Business Customers</td><td style="text-align: right;">298</td></tr> <tr><td>WY</td><td>Residential Customers</td><td style="text-align: right;">194</td></tr> <tr><td>WY</td><td>Business Customers</td><td style="text-align: right;">184</td></tr> </tbody> </table>	State	Type	Expenditures	NY	Residential Customers	8,786	NY	Business Customers	7,825	NJ	Residential Customers	4,115	NJ	Business Customers	3,558	PA	Residential Customers	6,478	PA	Business Customers	3,695	State	Type	Expenditures	MT	Residential Customers	322	MT	Business Customers	232	ID	Residential Customers	392	ID	Business Customers	298	WY	Residential Customers	194	WY	Business Customers	184
State	Type	Expenditures																																									
NY	Residential Customers	8,786																																									
NY	Business Customers	7,825																																									
NJ	Residential Customers	4,115																																									
NJ	Business Customers	3,558																																									
PA	Residential Customers	6,478																																									
PA	Business Customers	3,695																																									
State	Type	Expenditures																																									
MT	Residential Customers	322																																									
MT	Business Customers	232																																									
ID	Residential Customers	392																																									
ID	Business Customers	298																																									
WY	Residential Customers	194																																									
WY	Business Customers	184																																									

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *The Complete Guide to the SAS® Output Delivery System, Version 8*, Cary, NC: SAS Institute Inc., 1999. 310 pp.

The Complete Guide to the SAS® Output Delivery System, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

ISBN 1-58025-425-X

All rights reserved. Produced in the United States of America. 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.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of the software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19 Commercial Computer Software-Restricted Rights (June 1987).

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

1st printing, October 1999

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 registered trademarks or trademarks of their respective companies.

The Institute is a private company devoted to the support and further development of its software and related services.