Contents

About This Book ........................................................................................................... vii
About The Author ......................................................................................................... xiii
Acknowledgments ........................................................................................................ xv
Chapter 1: Introduction ................................................................................................. 1
The Template Structure................................................................................................. 2
Types of Templates.......................................................................................................... 5
   Data-Dependent Templates ........................................................................................ 5
   Data-Independent Templates .................................................................................... 9
Conclusion .................................................................................................................... 12

Chapter 2: Using Styles in Your Reports ................................................................. 13
Using Pre-Defined Styles ............................................................................................. 14
   Where Are These Styles Coming from? ................................................................ 15
Defining a New Style .................................................................................................... 19
   Cascading Style Sheets and the SAS Enterprise Guide Style Manager ............. 20
PROC TEMPLATE Styles ............................................................................................. 28
   Style Attribute References ....................................................................................... 44
Style Overrides and Conditional Formatting ............................................................. 46
   Dynamic Style Attribute Values ............................................................................. 51
Reporting Procedure Styles ....................................................................................... 56
   PROC PRINT Styles .................................................................................................. 57
   PROC REPORT Styles .............................................................................................. 62
   PROC TABULATE Styles ......................................................................................... 68
Inline Formatting ........................................................................................................ 79
Putting It All Together ................................................................................................. 81
## Chapter 3: Creating and Customizing Table Templates ................................. 95
- Defining a Table ........................................................................................................ 96
- Defining Headers and Footers .................................................................................. 103
- Defining Columns ....................................................................................................... 111
- Using Computed Column Values As Column Data .................................................. 118
- Table Attributes ........................................................................................................ 121
- Style Overrides and Traffic Lighting ......................................................................... 122
  - The CELLSTYLE AS Statement ............................................................................. 125
- Using Table Templates within DATA _NULL_ ......................................................... 130
  - Using Variables in Table Templates ..................................................................... 135
  - Using Generic Columns ......................................................................................... 139
  - DATA Step Tricks .................................................................................................. 140
- Inheritance ................................................................................................................. 142
- Modifying SAS Procedure Tables ........................................................................... 148
  - The PROC FREQ Crosstabulation Table ............................................................... 153
- Putting It All Together ............................................................................................... 167

## Chapter 4: Template Management ................................................................. 177
- Template Stores and the ODS Path ........................................................................... 177
- PROC TEMPLATE Statements ................................................................................... 183
  - The DELETE Statement ......................................................................................... 183
  - The LINK Statement .............................................................................................. 184
  - The LIST Statement ............................................................................................... 187
  - The PATH Statement .............................................................................................. 191
  - The SOURCE Statement ....................................................................................... 191

## Appendix A: Compatibility with Previous Versions of SAS ....................... 195
- Running PROC TEMPLATE Examples ..................................................................... 195
  - Styles ....................................................................................................................... 196
    - Using Cascading Style Sheets (CSS) ................................................................. 196
    - Changes to Style Attribute Names ................................................................... 196
    - Defining Multiple Style Elements Simultaneously .......................................... 197
    - Inheritance ........................................................................................................... 198
    - The CLASS Statement ....................................................................................... 198

## Appendix B: Style Attributes ........................................................................... 199
## Appendix C: Color Names Defined by SAS/GRAPH

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>219</td>
</tr>
<tr>
<td>Basic Hues</td>
<td>220</td>
</tr>
<tr>
<td>Blacks</td>
<td>220</td>
</tr>
<tr>
<td>Blues</td>
<td>221</td>
</tr>
<tr>
<td>Browns</td>
<td>222</td>
</tr>
<tr>
<td>Grays</td>
<td>223</td>
</tr>
<tr>
<td>Greens</td>
<td>224</td>
</tr>
<tr>
<td>Olives</td>
<td>226</td>
</tr>
<tr>
<td>Oranges</td>
<td>226</td>
</tr>
<tr>
<td>Pinks</td>
<td>227</td>
</tr>
<tr>
<td>Purples</td>
<td>228</td>
</tr>
<tr>
<td>Reds</td>
<td>229</td>
</tr>
<tr>
<td>Violets</td>
<td>229</td>
</tr>
<tr>
<td>Whites</td>
<td>230</td>
</tr>
<tr>
<td>Yellows</td>
<td>230</td>
</tr>
<tr>
<td>Index</td>
<td>233</td>
</tr>
</tbody>
</table>

Chapter 3: Creating and Customizing Table Templates

NOTE: This short excerpt was taken from a chapter in the published book.

Defining a Table .................................................................................................................96
Defining Headers and Footers .........................................................................................103
Defining Columns ..............................................................................................................111
Using Computed Column Values As Column Data ......................................................118
Table Attributes ..............................................................................................................121
Style Overrides and Traffic Lighting ..............................................................................122
  The CELLSTYLE AS Statement ..................................................................................125
Using Table Templates within DATA _NULL_ ..............................................................130
  Using Variables in Table Templates ..........................................................................135
  Using Generic Columns ..............................................................................................139
  DATA Step Tricks ........................................................................................................140
Inheritance ......................................................................................................................142
Modifying SAS Procedure Tables ..................................................................................148
  The PROC FREQ Crosstabulation Table ..................................................................153
Putting It All Together ....................................................................................................167

Although many people don’t realize it, almost all tabular output generated by SAS is created using a table template. Table templates describe what columns, headers, and footers should appear in the table. They can also contain style overrides to specify extra style information. In addition, there are even ways to compute new columns based on columns in the existing data set. Because SAS generates tables using table templates, you can modify the output of procedures by editing the table templates that the procedures use. But the real power of table templates is in writing your own from scratch to create tables that aren’t possible to create using any of the reporting procedures.

In this chapter, we show you how to create your own table templates, how to customize styles and add traffic lighting, and how to modify the table templates used by SAS procedures.
Defining a Table

You have already seen in previous chapters how to render some simple table templates, but let’s recap the methods here to refresh our memories. The first method works with all versions of SAS that have ODS. It uses PROC TEMPLATE to define the table. The DATA step is then used to bind a data set to that template to create the tabular output. The code below is the simplest table template that you can create. It simply prints out all columns of the data set using all of the default attributes.

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
END;
RUN;

DATA _NULL_;
  SET sashelp.class(OBS = 10);
  FILE PRINT ODS = (TEMPLATE = "mytable");
  PUT _ODS_;
RUN;
```

Here is the output from the code above.

```
The SAS System

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>M</td>
<td>14</td>
<td>69</td>
<td>112.5</td>
</tr>
<tr>
<td>Alice</td>
<td>F</td>
<td>13</td>
<td>56.5</td>
<td>84</td>
</tr>
<tr>
<td>Barbara</td>
<td>F</td>
<td>13</td>
<td>65.3</td>
<td>98</td>
</tr>
<tr>
<td>Carol</td>
<td>F</td>
<td>14</td>
<td>62.8</td>
<td>102.5</td>
</tr>
<tr>
<td>Henry</td>
<td>M</td>
<td>14</td>
<td>63.5</td>
<td>102.5</td>
</tr>
<tr>
<td>James</td>
<td>M</td>
<td>12</td>
<td>57.3</td>
<td>83</td>
</tr>
<tr>
<td>Jane</td>
<td>F</td>
<td>12</td>
<td>59.8</td>
<td>84.5</td>
</tr>
<tr>
<td>Janet</td>
<td>F</td>
<td>15</td>
<td>62.5</td>
<td>112.5</td>
</tr>
<tr>
<td>Jeffrey</td>
<td>M</td>
<td>13</td>
<td>62.5</td>
<td>84</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
<td>12</td>
<td>59</td>
<td>99.5</td>
</tr>
</tbody>
</table>
```
The DATA _NULL_ statement won’t change in most of the examples in this chapter, so we can simplify this example using a macro. The following macro is the same one that is defined in Chapter 1. It encompasses the DATA _NULL_ statement from the code above. We will be using this macro throughout this chapter to bind data sets to table templates.

```
%MACRO render(template, dataset);
   DATA _NULL_;
   SET &dataset;
   FILE PRINT ODS=(TEMPLATE="&template");
   PUT _ODS_;
   RUN;
%MEND render;
```

Here is the example from above using the macro in place of the DATA _NULL_ statement.

```
PROC TEMPLATE;
DEFINE TABLE mytable;
END;
RUN;

%render(mytable, sashelp.class);
```

If you are using SAS 9.3, you can use the convenience procedure PROC ODSTABLE. This procedure combines the definition of the table and binding a data set to the table template in one step. The example above can be rewritten using PROC ODSTABLE as follows.

```
PROC ODSTABLE DATA = sashelp.class;
RUN;
```

When you let PROC TEMPLATE use all defaults, it will display all columns in the data set. You can limit which columns are displayed by using the COLUMN statement. This works just like the COLUMN statement in PROC REPORT.

```
PROC TEMPLATE;
DEFINE TABLE mytable;
   COLUMN name age height weight;
END;
RUN;

%render(mytable, sashelp.class(OBS = 10));
```

For SAS 9.3, you can use the simpler PROC ODSTABLE code.

```
PROC ODSTABLE DATA = sashelp.class(OBS = 10);
   COLUMN name age height weight;
RUN;
```
Here is the output when you use the COLUMN statement.

```
The SAS System

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>14</td>
<td>69</td>
<td>112.5</td>
</tr>
<tr>
<td>Alice</td>
<td>13</td>
<td>56.5</td>
<td>84</td>
</tr>
<tr>
<td>Barbara</td>
<td>13</td>
<td>65.3</td>
<td>98</td>
</tr>
<tr>
<td>Carol</td>
<td>14</td>
<td>62.8</td>
<td>102.5</td>
</tr>
<tr>
<td>Henry</td>
<td>14</td>
<td>63.5</td>
<td>102.5</td>
</tr>
<tr>
<td>James</td>
<td>12</td>
<td>57.3</td>
<td>83</td>
</tr>
<tr>
<td>Jane</td>
<td>12</td>
<td>59.8</td>
<td>84.5</td>
</tr>
<tr>
<td>Janet</td>
<td>15</td>
<td>62.5</td>
<td>112.5</td>
</tr>
<tr>
<td>Jeffrey</td>
<td>13</td>
<td>62.5</td>
<td>84</td>
</tr>
<tr>
<td>John</td>
<td>12</td>
<td>59</td>
<td>99.5</td>
</tr>
</tbody>
</table>
```

If you have a lot of columns in your data set and they can be expressed using the list notation (e.g., a1-a5), you can use the list notation in the COLUMN statement as well.

```
DATA ab;
  INPUT a1-a5 b1-b5;
DATALINES;
  1 2 3 4 5 101 102 103 104 105
  6 7 8 9 10 106 107 108 109 110
  11 12 13 14 15 111 112 113 114 115
RUN;

PROC TEMPLATE;
DEFINE TABLE mytable;
  COLUMN a1-a5 b1-b5;
END;
RUN;

%render(mytable, ab);
```

Again, this can be simplified in SAS 9.3 as shown in the following code. For the sake of simplicity, this will be the last PROC ODSTABLE sample shown. Just keep in mind that if you have SAS 9.3, you can reduce most of the examples down by taking the content of the table template and using
that within PROC ODSTABLE to get the same result. The examples that step outside of the simple compile-and-render scenario that will not work with PROC ODSTABLE will be noted.

```sas
PROC ODSTABLE DATA = ab;
  COLUMN a1-a5 b1-b5;
RUN;
```

Here is the output from the list notation code.

![The SAS System table](image)

The COLUMN statement can do more advanced things as well. For example, you can stack values on top of each other within a cell by grouping the column names in parentheses. The COLUMN statement here stacks the age, height, and weight variables on top of each other in the same cell.

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
  COLUMN name (age height weight);
END;
RUN;

%render(mytable, sashelp.class(OBS = 3));
```
The output is shown below.

You might have noticed that the header for the stacked column contains only Age. The variable that is on the top of the stack determines what the header text is. However, you can change the text of the header manually. That technique is discussed later in this chapter.

You can combine this stacking feature with the list notation feature too.

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
  COLUMN (a1-a5) (b1-b5);
END;
RUN;

%render(mytable, ab(OBS = 2));
```
Here is the output from the previous code.

Another way to stack is with groups of columns. In this method, you group the columns together with parentheses and then stack each group on top of each other using an asterisk (*). The following code stacks sex over age and height over weight:

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
   COLUMN name (sex height) * (age weight);
END;
RUN;

%render(mytable, sashelp.class(OBS = 4));
```

The output below is created by this COLUMN statement.
When using this method, it is sometimes easier to see what the output table will look like by putting a line break after each asterisk in the COLUMN statement. That way, your COLUMN statement looks a lot like the resulting table.

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
  COLUMN name (sex height) *
       (age weight);
END;
RUN;

%render(mytable, sashelp.class);
```

Of course, the list notation for variables works with this stacking method as well. Here is an example.

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
  COLUMN (a1-a5) *
       (b1-b5);
END;
RUN;

%render(mytable, ab);
```

The output from this code is shown below.

![The SAS System](image)

Rendering columns in this manner will use the default attributes for the columns themselves. There are many attributes that you can use to change the way the columns look, and they will be discussed later in this chapter.
Defining Headers and Footers

Table templates can contain two types of headers: table headers and column headers. Luckily, both types of headers are defined the same way. This section describes how to define headers and how to apply them to tables. The next section deals with applying headers to a column.

To define a header, use the standard PROC TEMPLATE method of utilizing a DEFINE statement that has a template type and name to start the definition, and an END statement to end it. In this case, the type can be HEADER or FOOTER. Headers go at the top of the table, and footers go at the bottom of the table. The following code shows a definition of each of these types:

```
DEFINE HEADER myheader;
END;

DEFINE FOOTER myfooter;
END;
```

Defining a header or footer doesn’t make much sense unless you associate some text with it. This is done by using the TEXT statement, as shown here:

```
DEFINE HEADER myheader;
    TEXT 'Class Information';
END;
```

The value given in the TEXT statement can be any combination of quoted strings, dynamic variables (as specified in the DYNAMIC statement), macro variables (as specified in the MVAR statement), and built-in variables.

Dynamic, macro, and built-in variables are discussed in the “Using Variables in Table Templates” section of this chapter. The _LABEL_variable in the following code is an example of a built-in variable that inserts the data object’s label.

```
DEFINE HEADER myheader;
    TEXT 'Class Information (' _LABEL_ ')';
END;
```

Now that we have a header definition, we need to attach it to a table. This can be done in one of two ways: 1) use the HEADER statement or 2) define the header within a table definition. The HEADER statement is much like the simple form of the COLUMN statement because you list the names of the headers that you want to appear in the table. However, because most headers you create will be specific to a particular table template, the second method is more common. Here is an example of applying a header to a table using Method 1:

```
PROC TEMPLATE;
    DEFINE HEADER my.header;
        TEXT 'Class Information';
    END;
```
Here is an example of applying a header to a table using Method 2:

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
  DEFINE HEADER myheader;
    TEXT 'Class Information';
  END;
END;
RUN;

%render(mytable, sashelp.class(OBS = 6));
```

As you can see, defining a header outside of the table is more work. In addition, this is a case where PROC ODSTABLE cannot be used to define all parts of the table. PROC ODSTABLE can only contain the statements and objects from a single table template. If you use PROC ODSTABLE, you have to use a separate PROC TEMPLATE step just to define the external header. Defining a header outside of a table is most common when you are going to use that header definition within more than one table definition. The need for this is not as common, so you might not use that particular method much.

Here is what the HTML output for either of the above code blocks looks like.

![The SAS System](image-url)
You can add as many headers to the table as you want. They will appear in the table in the same order that they do in the table definition.

PROC TEMPLATE;
DEFINE TABLE mytable;
  DEFINE HEADER first;
    TEXT 'First';
  END;
  DEFINE HEADER second;
    TEXT 'Second';
  END;
  DEFINE HEADER third;
    TEXT 'Third';
  END;
  DEFINE HEADER fourth;
    TEXT 'Fourth';
  END;
END;
END;
RUN;
%render(mytable, sashelp.class(OBS = 3));

This code produces the output shown below.

As you can see from the last two examples, table headers span the entire width of the table. You can change this by giving end points to the header using the START= and END= attributes. The values of these attributes are the names of the columns where the table header should start and end, respectively. The following code shows the same header as in the first example, but it only spans the first three columns.
PROC TEMPLATE;
DEFINE TABLE mytable;
   DEFINE HEADER myheader;
      TEXT 'Class Information';
      START = name;
      END = age;
   END;
END;
RUN;

%render(mytable, sashelp.class(OBS = 6));

The output is shown below.

![The SAS System](image)

Now let’s combine stacking with spanning. The stacking behavior is similar to the way that the blocks in the game Tetris stack up. The headers fall and fill in empty spaces. If they encounter any other header as they fall, that is their final resting spot. ODS stretches the cells vertically to fill the empty spaces. The following code defines four headers with various spans. It’s probably not something that you would do in reality, but it does show you the flexibility of table headers.

PROC TEMPLATE;
DEFINE TABLE mytable;
   DEFINE HEADER first;
      TEXT 'First';
      START = name;
      END = name;
   END;
   DEFINE HEADER second;
      TEXT 'Second';
      START = sex;
END = weight;
END;
DEFINE HEADER third;
    TEXT 'Third';
    START = sex;
    END = height;
END;
DEFINE HEADER fourth;
    TEXT 'Fourth';
    START = height;
    END = weight;
END;
RUN;

%render(mytable, sashelp.class(OBS = 4));

Here is the output.

![The SAS System](image)

When we mentioned the HEADER statement previously, it might have sounded like the only time that you would use it is to include external header definitions. While it works fine for that, you can also use the HEADER statement on internal header definitions. In fact, you can mix and match internal and external headers in the same table. Going back to our Tetris example, while in the game of Tetris you have to accept the tiles that the game chooses for you in the order that it chooses for you, when dealing with table headers, you can reorder and exclude headers from being applied to a table by using the HEADER statement. Here is the last example again, but this time we are excluding the first header and reversing the order in which the headers are applied to the table.
PROC TEMPLATE;
DEFINE TABLE mytable;
  DEFINE HEADER first;
    TEXT 'First';
    START = name;
    END = name;
  END;
  DEFINE HEADER second;
    TEXT 'Second';
    START = sex;
    END = weight;
  END;
  DEFINE HEADER third;
    TEXT 'Third';
    START = sex;
    END = height;
  END;
  DEFINE HEADER fourth;
    TEXT 'Fourth';
    START = height;
    END = weight;
  END;
  HEADER fourth third second;
END;
END;
RUN;

%render(mytable, sashelp.class(OBS = 4));

The output for this code is shown below.
Note that the position of the HEADER statement is important when you define more headers than will appear in the table. Each time you define a table header, that header is automatically added to the header list. If you were to put the HEADER statement at the beginning of this table definition, you would still have four table headers in the order: fourth, third, second, first.

**Common Table Header and Footer Attributes**

There are various attributes besides START= and END= that you can use to change the behavior of headers and footers. The complete list can be seen in the *SAS 9.3 Output Delivery System: User’s Guide, Second Edition*. There are approximately two dozen attributes from which to choose. If you look closely in the documentation, you’ll see that many of these attributes are used only by the ODS Listing output. Below is a list of the most common header and footer attributes.

- **FIRST PANEL=ON | OFF**
  - specifies whether or not a spanning header appears only on the first panel if the header is broken across multiple panels. This option applies only to output that supports paneling.

- **JUST=LEFT | CENTER | RIGHT**
  - specifies the horizontal alignment of the text of the header.

- **LAST PANEL=ON | OFF**
  - specifies whether or not a spanning footer appears only on the last panel if the footer is broken across multiple panels. This option applies only to output that supports paneling.

- **PARENT=header-path**
  - specifies a header definition to inherit from. Inheritance will be discussed later in this chapter.

- **PREFORMATTED=ON | OFF**
  - specifies whether or not to treat the text of the header as preformatted text. Preformatted text preserves all whitespace and sets the font to a monospace font. Using this option also appends ‘fixed’ to the default style element name used for the header or footer. For example, header would become headerfixed, rowfooter would become rowfooterfixed, etc.

- **PRINT=ON | OFF**
  - specifies whether or not the header should be printed.

- **SPLIT=‘character’**
  - specifies a character that will be treated as a newline character. Whenever the specified character is encountered in the text of the header, a line break is put in its place. Using the SPLIT= attribute isn’t necessary to put line breaks in a header (though it is the recommended way). If you do not specify a character with the SPLIT= attribute and the first character of the header text isn’t an alphanumeric character, a blank, an underscore (_), a hyphen (-), a period (.), or a percent sign (%), the first character of the header text will automatically become the split character. For example, the header text “/Age/Height/Weight” would automatically convert “/” characters into newlines if SPLIT= was not specified.
STYLE=style-override
    specifies a style override to apply to the header. Style overrides are discussed in “Style
    Overrides and Conditional Formatting” in Chapter 2.

VJUST=TOP | CENTER | BOTTOM
    specifies the vertical alignment of the header text.

As opposed to setting values of these attributes explicitly, it is possible to set the attribute values
dynamically at run time as well. This is not as useful when using the PROC ODSTABLE
procedure, but it can be very handy when creating reusable table templates from PROC
TEMPLATE.

One statement within both header and footer definitions is the NOTES statement. The NOTES
statement is like a comment, but it is stored with the header definition so that it can be viewed later.
Here is an example of using the NOTES statement based on the very first header definition that we
created in this chapter.

    PROC TEMPLATE;
    DEFINE HEADER my.header;
      NOTES "Externally defined header for tables that
          use Sashelp.Class";
      TEXT "Class Information";
    END;
    RUN;

The last three statements available in headers and footers that we are going to discuss, DYNAMIC,
MVAR, and NMVAR, are used to set run-time attribute values. These values come from dynamic
variables set in the DATA step and macro variables6. Because this behavior is shared between
tables, columns, headers, and footers, the explanation for this feature can be found in the section
“Using Variables in Table Templates.”

Almost all of the discussion in this section deals with headers; we didn’t say much about footers.
The syntax for creating footers is identical to headers. The only difference is that they are located at
the bottom of the table rather than at the top. The figure below shows the output of the last header
example with all of the headers changed to footers.
So far we have covered the basics of putting together columns to form a table as well as adding headers and footers. Many more advanced operations are available for columns. Because some of these operations involve header definitions, we saved the discussion of column definitions until now.

### Defining Columns

Although the COLUMN statement describes the overall structure of the table columns, the columns themselves can have their own definitions just like headers and footers. A column definition can define what the column should look like, what the column header should contain, and what data column should be used. It can even calculate the data values themselves. Putting columns in a table is just like putting headers in a table. You have two choices:

1. Use the COLUMN statement to create columns based on the data columns in a data set.
2. Define the column within a table definition.

You can also use a combination of these methods where you define the overall structure of the table using the COLUMN statement and then create column definitions for the columns that need special treatment.

Let’s go back to our first table that used the COLUMN statement.

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
  COLUMN name age height weight;
END;
RUN;
```
The following table template is equivalent but is not nearly as convenient.

    PROC TEMPLATE;
    DEFINE TABLE mytable;
    DEFINE COLUMN name;
    END;
    DEFINE COLUMN age;
    END;
    DEFINE COLUMN height;
    END;
    DEFINE COLUMN weight;
    END;
    END;
    RUN;

    %render(mytable, sashelp.class);

Using a hybrid approach, you can use the COLUMN statement to define the overall structure, and then use column definitions to apply specific behavior. This would look something like the following.

    PROC TEMPLATE;
    DEFINE TABLE mytable;
    COLUMN name age height weight;
    DEFINE height;
    END;
    DEFINE weight;
    END;
    END;
    RUN;

    %render(mytable, sashelp.class);

Note that when you use both a COLUMN statement and column definitions, specifying the template type in the column definition is optional. PROC TEMPLATE already knows that it is a column because of the information in the COLUMN statement. Also, just as with headers, the position of the COLUMN statement is important. In the example above, if you were to define more column templates than there are columns listed in the COLUMN statement, they would automatically be appended to the end of the column list. To explicitly limit which columns go into a table, put the COLUMN statement at the end of the table definition.

All of the column templates in that last code snippet were kind of nonsensical because they didn’t do anything other than the default behavior, which can be gotten simply by listing the columns in the COLUMN statement. However, defining column templates is necessary when you start to add customization. Let’s say that we wanted to change the column header for the column Name to
‘First Name’. The simplest way would be to add a column definition for that column and set the HEADER= attribute.

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
  COLUMN name age height weight;
  DEFINE name;
    HEADER = 'First Name';
  END;
END;
RUN;

%render(mytable, sashelp.class(OBS=6));
```

The resulting output is shown below.

![The SAS System](image)

While this works for simple headers, you might want to use a full header definition instead of just a string. The following form enables you to add styles and other visual effects to headers as we will see later. In the case of column headers, you must use the HEADER= attribute to associate a header definition with a column. It is not automatically applied as is the case in a table header. The output for this code is identical to the output from the previous example:

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
  COLUMN name age height weight;
  DEFINE name;
    DEFINE HEADER nameheader;
      TEXT 'First Name';
    END;
END;
```
Let’s go back to a problem that we had in the column stacking example. When stacking columns, the header for the resulting column is always the header from the column that is on the top of the stack. Let’s put a header on that column to include the headers for all of the columns in the stack. We also use this opportunity to show off the SPLIT= header attribute described in the previous section.

PROC TEMPLATE;
DEFINE TABLE mytable;
   COLUMN name (age height weight);
   DEFINE age;
       DEFINE HEADER ageheightweight;
       TEXT 'Age*Height*Weight';
       SPLIT = '*';
   END;
   HEADER = ageheightweight;
END;
END;
RUN;

%render(mytable, sashelp.class);

The output for this example is shown below.

![The SAS System](image)
Another common thing to do with a column template is to change the data format. This is done using the FORMAT= attribute. The value is simply a SAS format. Let’s continue to use the definition from the last example to try out the FORMAT= attribute. We want all of the numbers in our Age/Height/Weight column to line up. You can do this by formatting all of the numbers the same way. You can make them all integers with the following code.

```sas
PROC TEMPLATE;
DEFINE TABLE mytable;
    COLUMN name (age height weight);
    DEFINE age;
        DEFINE HEADER ageheightweight;
            TEXT 'Age*Height*Weight';
            SPLIT = '**';
        END;
    HEADER = ageheightweight;
    FORMAT = 3.;
END;
DEFINE height;
    FORMAT = 3.;
END;
DEFINE weight;
    FORMAT = 3.;
END;
END;
RUN;
%render(mytable, sashelp.class(OBS = 3));
```

Here is the output from the code above.
Common Table Column Attributes

There are many other column attributes that can be used to affect the behavior of columns. Just as with headers, many of the attributes are used only for ODS Listing output. The Listing-only attributes have been left out of this list. For a full list of column attributes and descriptions, see the SAS 9.3 Output Delivery System: User’s Guide, Second Edition.

BLANK_DUPS=ON | OFF
specifies whether or not to leave a cell empty if the value in the previous row was the same.

DATA_FORMAT_OVERRIDE=ON | OFF
specifies whether or not the format from the data column should take precedence over the format in the template.

DATANAME=column-name
specifies the name of the data column to use rather than using the name of the data column that matches the column name.

DROP=ON | OFF
specifies whether or not to include the column in an output data set. This option applies only to the ODS Output destination.

FORMAT=SAS-format
specifies a SAS format to use for the data in the column.

FORMAT_WIDTH=positive-integer
specifies the format width for the column. This attribute is mainly intended for procedure writers. In most cases, you should use the FORMAT= attribute.

FORMAT_NDEC=positive-integer
specifies the number of decimals for the column. This attribute is mainly intended for procedure writers. In most cases, you should use the FORMAT= attribute.

FUZZ=number
specifies a numeric value that is used as a minimum value for data in the cells. Any value that is less than the value given in FUZZ= is treated as zero.

GENERIC=ON | OFF
specifies whether or not the column template can be used by multiple columns in the table. The details about generic columns will be discussed later in this chapter.

HEADER=header-specification
specifies the header for the column. The value of header-specification can be either a quoted string or the name of a header template.

ID=ON | OFF
specifies whether or not the column should be repeated on each data panel. This applies only to output types that support paneling.
**JUST=LEFT | CENTER | RIGHT**
specifies the horizontal alignment of the column’s content. The default behavior is to left align character content and right align numeric content. Because of some details in the way that alignments are evaluated, setting the alignment using the TEXTALIGN= style attribute in a style override is preferred over this method.

**LABEL=’text’**
specifies a label to use instead of the label supplied in the data set.

**MERGE=ON | OFF**
specifies whether or not the content of the current data cell should be merged with the content of the following column’s data.

**PARENT=column-path**
specifies a column definition to inherit from. Inheritance will be discussed later in this chapter.

**PREFORMATTED=ON | OFF**
specifies whether or not to treat the text of the column as preformatted text. Preformatted text preserves all whitespace and sets the font to a monospace font. Using this option also sets the style element to `datafixed`.

**PRE_MERGE=ON | OFF**
specifies whether or not the content of the current data cell should be merged with the content of the previous column’s data.

**PRINT=ON | OFF**
specifies whether or not the column should be printed.

**PRINT_HEADERS=ON | OFF**
specifies whether or not the column header should be printed.

**STYLE=style-override**
specifies a style override to apply to the header. Style overrides are discussed in the “Style Overrides and Conditional Formatting” section in Chapter 2.

**TEXT_SPLIT=’character’**
specifies a character that will be treated as a newline character. Whenever the specified character is encountered in the text of the column, a line break is put in its place.

**VJUST=TOP | CENTER | BOTTOM**
specifies the vertical alignment of the column’s content.
Just as with headers, columns also have an attribute-like statement called NOTES that embeds a comment into the template that can be viewed later. Also, like headers, the attribute values can be set dynamically at run time as discussed in “Using Table Templates within DATA_NULL_” later in this chapter.

So far we have been using only data that is explicitly defined in a data set, but you can create columns whose data is based on combinations of data in a data set. This type of column is discussed in the following section.

### Using Computed Column Values As Column Data

Although you do need a data set to render a table template, you do not necessarily need a data column for each column in the table. You can create columns that are computed from other data in the table. These are called “computed columns.” The COMPUTE AS statement determines whether a column is associated with a data column or an expression. This statement takes a standard expression as its argument. The expression is evaluated for every observation in the input data set to get the resulting output data value. Here is the general format of the COMPUTE AS statement:

```
    COMPUTE AS expr;
```

Since we have been using the SASHELP.CLASS data set, let’s continue that trend and use that information to compute something useful. One thing that can be calculated using that information is the body mass index (BMI). This is one measure to determine if someone is underweight, normal weight, or overweight. It is computed using the following formula^{10}.

\[
    BMI = \frac{weight \cdot 703}{height^2}
\]

where \( weight \) is expressed in pounds and \( height \) is in inches.

Now that we have all of the information, let’s add a new column to the table that contains each individual’s BMI.

```sas
PROC ODS Table DATA = sashelp.class(OBS = 7);
    COLUMN name age height weight bmi;
    DEFINE bmi;
        COMPUTE AS (weight * 703) / (height * height);
        HEADER = 'BMI';
        FORMAT = 2.;
    END;
RUN;
```
The resulting output is shown below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
<th>BMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>14</td>
<td>69</td>
<td>112.5</td>
<td>17</td>
</tr>
<tr>
<td>Alice</td>
<td>13</td>
<td>56.5</td>
<td>84</td>
<td>18</td>
</tr>
<tr>
<td>Barbara</td>
<td>13</td>
<td>65.3</td>
<td>98</td>
<td>16</td>
</tr>
<tr>
<td>Carol</td>
<td>14</td>
<td>62.8</td>
<td>102.5</td>
<td>18</td>
</tr>
<tr>
<td>Henry</td>
<td>14</td>
<td>63.5</td>
<td>102.5</td>
<td>18</td>
</tr>
<tr>
<td>James</td>
<td>12</td>
<td>57.3</td>
<td>83</td>
<td>18</td>
</tr>
<tr>
<td>Jane</td>
<td>12</td>
<td>59.8</td>
<td>84.5</td>
<td>17</td>
</tr>
</tbody>
</table>

A method to create new data values in a column is by using the TRANSLATE INTO statement. This type of column doesn’t generate a new column; rather it simply takes the data in a column and converts it into a different representation. It is kind of like using a SAS format, but it gives you the full power of an expression in both the matching of a value and the generating of a value. In addition, the value of the column can be based on the values of any number of columns, just like in the case of the COMPUTE AS statement. The general form of the TRANSLATE INTO statement is as follows.

```
TRANSLATE expr-1a INTO expr-1b,
...
expr-na INTO expr-nb;
```

The data in the column is tested against the expressions on the left side of the INTO keyword. If the expression is true, then the data that is printed in the output is generated by the expression on the right side of the INTO keyword. Let’s look at an example.

We do not care about the exact weight values for the individuals in the SASHELP.CLASS data set if they are above or below a certain value. However, those values are different for the male and female students. For male students, we only care if the weight is between 90 and 120. Any values outside that range can be represented by ‘< 90’ and ‘> 110’, respectively. For females, we want to do the same thing, except the values are 70 and 100. We could not accomplish this with a SAS format because it requires the data from two columns.
PROC TEMPLATE;
DEFINE TABLE mytable;
  COLUMN name sex age height weight;
DEFINE sex;
  PRINT = OFF;
END;
DEFINE weight;
  TRANSLATE (sex = 'M' AND weight > 110) INTO '> 110',
            (sex = 'M' AND weight < 90) INTO '< 90',
            (sex = 'F' AND weight > 100) INTO '> 100',
            (sex = 'F' AND weight < 70) INTO '< 70';
  STYLE = {TEXTALIGN = RIGHT};
END;
END;
RUN;

%render(mytable, sashelp.class(OBS = 7));

In this case, because we are using the `sex` variable in the calculation, we need to include it in the COLUMN statement as well. Turn it off by using PRINT=OFF in the column definition. The value of the weight column is referenced as ‘weight’ in the expression. However, you could also use the keyword _VAL_ which refers to the data value of the current column. This does not make any difference when the TRANSLATE INTO statement is on a column; however, the same TRANSLATE INTO statement can be used in the table context to apply to all of the columns in a table. In that case, the _VAL_ keyword would correspond to whichever column the expression was currently being applied to. Here is the output from the code above.

```
The SAS System

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>14</td>
<td>69</td>
<td>&gt; 110</td>
</tr>
<tr>
<td>Alice</td>
<td>13</td>
<td>56.5</td>
<td>84</td>
</tr>
<tr>
<td>Barbara</td>
<td>13</td>
<td>65.3</td>
<td>98</td>
</tr>
<tr>
<td>Carol</td>
<td>14</td>
<td>62.8</td>
<td>&gt; 100</td>
</tr>
<tr>
<td>Henry</td>
<td>14</td>
<td>63.5</td>
<td>102.5</td>
</tr>
<tr>
<td>James</td>
<td>12</td>
<td>57.3</td>
<td>&lt; 90</td>
</tr>
<tr>
<td>Jane</td>
<td>12</td>
<td>59.8</td>
<td>84.5</td>
</tr>
</tbody>
</table>
```
The TRANSLATE INTO statement can be used on the table itself. There are attributes, just like in columns, headers, and footers, that can be used on tables to affect their behaviors as well. Some of these attributes don’t make sense until after you know some things about columns and headers, so we saved the discussion of them until now.

**Table Attributes**

Table attributes are used to affect the behavior of the table itself, as well as some header and column behaviors. Just like with columns and headers, many of the attributes are ODS Listing-specific and will not be discussed here. A complete list of attributes can be found in the *SAS 9.3 Output Delivery System: User’s Guide, Second Edition*.

**BYLINE=ON | OFF**

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

**CLASSLEVELS=ON | OFF**

specifies whether or not to enable the BLANK_DUPS= attribute on a column if the previous column also has BLANK_DUPS= enabled and its value changed in the current row.

**CONTENTS=ON | OFF**

specifies whether or not to put the table in the table of contents.

**CONTENTS_LABEL=’string’**

specifies the label to use for the table in the table of contents, the Results window, and the trace record.

**DATA_FORMAT_OVERRIDE=ON | OFF**

specifies whether or not the format from the data column should take precedence over the format in the template.

**LABEL=’text’**

specifies a label for the table. If no label is provided, the label from the data set will be used. If it does not have a label, the text of the first table header is used.

**NEWPAGE=ON | OFF**

specifies whether or not to force a page break before the table.

**ORDER_DATA=ON | OFF**

specifies whether or not the order of the columns in the data set should override the order of the columns specified in the template.

**PARENT=table-path**

specifies a table definition to inherit from. Inheritance will be discussed later in this chapter.

**PRINT_FOOTERS=ON | OFF**

specifies whether or not to print the table footers.
PRINT_HEADERS=ON | OFF
specifies whether or not to print the table headers.

STYLE=style-override
specifies a style override to apply to the header. Style overrides are discussed in “Style
Overrides and Conditional Formatting” in Chapter 2.

USE_FORMAT_DEFAULTS=ON | OFF
specifies whether or not the width and number of decimals should be taken from the default
values for the format name rather than the template.

USE_NAME=ON | OFF
specifies the column name as the column header if neither the column definition nor the data
set specifies one.

Just as with columns, headers, and footers, tables also support a NOTES statement that enables you
to store a comment with the table that describes what the table is for.

While we have only shown each attribute with explicit values here, it is possible to have attribute
values dynamically set at run time rather than at compile time. This is not as useful when you are
using the PROC ODSTABLE convenience procedure to generate tables, but it can be quite
powerful when you are creating reusable table templates using PROC TEMPLATE.

While the attributes described for columns, headers, footers, and tables have enabled us to change
various aspects of behavior, the look and feel defined by the overall style remains. It is possible to
override style attributes within the table family of templates, and even do traffic lighting based on
the data in the table. These techniques are discussed in the following section.

NOTE: This short excerpt was taken from a chapter in the published book.
Index

A

ACTIVELINKCOLOR= style attribute 200
Add New Style dialog box 21
ALL option, TABLE statement (TABULATE) 74
APPEND option, ODS PATH statement 181
ASIS= style attribute 200
attributes
  See specific types of attributes

B

BACKGROUNDCOLOR= style attribute 200
BACKGROUNDIMAGE= style attribute 200
BACKGROUNDPOSITION= style attribute 200–201
BACKGROUNDREPEAT= style attribute 200–201
Base.Freq.CrossTabFreqs template 153–167
Base.Template.Column template 146
Base.Template.Footer template 146
Base.Template.Header template 146
Base.Template.Style template 43, 146
Base.Template.Table template 146
basic hues (SAS/GRAPH color names) 220
binding data sets to table templates 96–99, 130
blacks (SAS/GRAPH color names) 220
BLANK_DUPS= table column attribute 116, 174
blues (SAS/GRAPH color names) 221–222
BMI (body mass index) 118–119
body mass index (BMI) 118–119
BodyPage style element 35
BORDERBOTTOM= style attribute 202
BORDERBOTTOMCOLOR= style attribute 202
BORDERBOTTOMSTYLE= style attribute 202
BORDERBOTTOMWIDTH= style attribute 202
BORDERCOLLAPSE= style attribute 201–202
BORDERCOLOR= style attribute 202
BORDERLEFTCOLOR= style attribute 202
BORDERLEFTSTYLE= style attribute 202
BORDERLEFTWIDTH= style attribute 202
BORDERRIGHTCOLOR= style attribute 202
BORDERRIGHTSTYLE= style attribute 202
BORDERRIGHTWIDTH= style attribute 202
BORDERSPACING= style attribute 202
BORDERSTYLE= style attribute 202–203
BORDERTOPCOLOR= style attribute 203
BORDERTOPSTYLE= style attribute 203
BORDERTOPWIDTH= style attribute 203
BORDERWIDTH= style attribute 203
BOX= option, TABLE statement (TABULATE) 73–74
browns (SAS/GRAPH color names) 222–223
built-in variables 103, 128, 156
BYLINE= table attribute 121

C

CAPSTYLE= style attribute 212
caret (^) 79
cascading inheritance 38, 44
Cascading Style Sheets (CSS)
  about 19
  Enterprise Guide Style Manager and 20–28
  version compatibility 196
CELLCHI2 option, TABLES statement (FREQ) 160
CELLPADDING= style attribute 203
CELLSPACING= style attribute 203
CELLSTYLE AS statement, TEMPLATE procedure
  about 125–130
  creating interesting effects 146–147, 166
  example using 169
CELL_STYLE= crosstabs attribute 162
CELLVALUE statement, TEMPLATE procedure 159–162
CLASS statement
  TABULATE procedure 71
  TEMPLATE procedure 29, 37, 39–43, 198
CLASS= style attribute (HTML output) 211
CLASSLEVEL statement, TABULATE procedure 71, 78
CLASSLEVELS= table attribute 121
_COL_ built-in variable 128
_COL_LABEL_ built-in variable 156
_COL_NAME_ built-in variable 156
color names (SAS/GRAPH)
   See SAS/GRAPH color names
COLOR= style attribute 204
COLS_HEADER= crosstabs attribute 157, 163
COL_TOTAL_STYLE= crosstabs attribute 162
column attributes 116–118, 124–125
column headers, defining 103–111
COLUMN statement, TEMPLATE procedure
   defining columns 111–112
   defining tables 97–103
   example using 168
columns
   computed values as data in 118–121
defining 111–118
generic 139–140
inheritance and 145
modifying in output 134
setting attribute values at run time 135
COLUMNS= option, ODS FILE PRINT
   statement 134, 138–139
COL_VAR_STYLE= crosstabs attribute 162
COMPUTE AS statement, TEMPLATE procedure
   118–119
COMPUTE statement, REPORT procedure 67
conditional formatting
   reporting procedure styles and 61, 66, 74
   style overrides and 46–56, 122–130
   table templates and 122–130
CONNECT= style attribute 212
CONTENTS procedure 146, 148, 153
CONTENTS= table attribute 121
CONTENTS_LABEL= table attribute 121
CONTENTTYPE= style attribute (HTML output) 211
CONTRASTCOLOR= style attribute 212
crosstabs templates 153–167, 184
crosstabulation tables 153–167, 184
CSS (Cascading Style Sheets)
   about 19
   Enterprise Guide Style Manager and 20–28
   version compatibility 196
CSSSTYLE= option, ODS HTML statement 24, 27
CUMCOL option, TABLES statement (FREQ) 160
curly braces {} 79
CURSOR= style attribute 204–205

D
data-dependent templates 5–9
data-independent templates 9–12
DATA _NULL_ statement 7, 130–142
data sets, binding to table templates 96–99, 130
DATA step
   binding data sets to table templates 96, 130
   conditional formatting and 55
   data-dependent templates and 5
   PUT statement 140–141
   table template tricks 134–137, 140–142
Data style element 23
DATA_FORMAT_OVERRIDE= attribute
   for crosstabs 161
   for table columns 116
   for tables 121
_DATANAME_ built-in variable 128
DATANAME= table column attribute 116
DEFINE CROSSTABS statement, TEMPLATE procedure 153–158
DEFINE FOOTER statement, TEMPLATE procedure 103, 155
DEFINE HEADER statement, TEMPLATE procedure
   about 103, 107–109
defining crosstabs templates 154–157
DEFINE statement, TEMPLATE procedure
   about 3–4, 103
   style definitions and 29
DEFINE STYLE statement, TEMPLATE procedure 40–43
DELETE statement, TEMPLATE procedure 167, 183–184
DEVIOATION option, TABLES statement (FREQ) 160
DISPLAYOPTS= style attribute 212
DOCTYPE= style attribute (HTML output) 211
DROP= table column attribute 116
DYNAMIC= option, ODS FILE PRINT statement 135–137
DYNAMIC statement, TEMPLATE procedure 103, 110, 135–137
dynamic style attribute values 51–56, 165
dynamic variables 156
E
EDIT statement, TEMPLATE procedure 150–152
END= header attribute 105–106
END statement, TEMPLATE procedure about 3–4, 103
style definitions and 29
ENDCOLOR= style attribute 212
Enterprise Guide Style Manager 20–28
equals (=) sign 28
escape characters 79–80
Espresso editor 25–26
EXPECTED option, TABLES statement (FREQ) 160
explicit inheritance 38, 44
EXPRESSION() function 55–56
F
FILE= option, SOURCE statement (TEMPLATE) 192
FILLRULEWIDTH= style attribute 205
FIRST_PANEL= attribute
for table footers 109
for table headers 109
FLYOVER= style attribute 205
FMISSING dynamic variable 156
FONT= style attribute 205
FONTFAMILY= style attribute 205
FONTSIZE= style attribute 205
FONTSTYLE= style attribute 206
FONTWEIGHT= style attribute 206
FONTWIDTH= style attribute 206
footers defining 103–111
inheritance and 145
FORMAT= attribute
for crosstabs 161
for table columns 115–116
FORMAT_NDEC table column attribute 116
formatting inline 79–81
traffic lighting effects 52
FORMAT_WIDTH= table column attribute 116
forward slash (/) 31
FRAME= style attribute 206
FREQ procedure crosstabulation tables and 153–167, 184
TABLES statement 160–161
FROM option, STYLE statement (TEMPLATE) 37–41
FUZZ= table column attribute 116
G
generic= table column attribute 116, 139
GRADIENTDIRECTION= style attribute 212
GRAND_TOTAL_STYLE= crosstabs attribute 162
graph-specific style attributes 212–218
graphs creating 8–9
style attributes specifically for 212–218
grays (SAS/GRAPH color names) 223–224
greens (SAS/GRAPH color names) 224–226
H
HEADER= attribute
for crosstabs 161
for table columns 113, 116
Header style element 23, 31
HEIGHT= style attribute 207
HREFTARGET= style attribute (HTML output) 211
HTML output
   HTML-specific style attributes 211
   style considerations 9, 14, 17, 21, 24, 27, 29, 32, 81–87
   table template examples 104
HTMLID= style attribute (HTML output) 211
HTMLSTYLE= style attribute (HTML output) 211
I
ID statement, PRINT procedure 57–59
ID= table column attribute 116
implied inheritance 38
inheritance
   cascading 38, 44
   explicit 38, 44
   implicit 38
   style attributes 44
   style definition 40–43
   style elements 40–43
   table templates and 142–147
   version compatibility 198
inline formatting 79–81
J
JUST= attribute
   for styles 207
   for table columns 117
   for table headers and footers 109
K
key-value pairs 28
KEYWORD statement, TABULATE procedure 71, 73
L
LABEL= attribute
   for crosstabs 161
   for table columns 117
   for tables 121
   _LABEL_ built-in variable 103, 128
LAST_PANEL= attribute
   for table footers 109
   for table headers 109
LEGEND_STYLE= crosstabs attribute 162–163
LINECOLOR= style attribute 212
LINESTYLE= style attribute 212–213
LINETHICKNESS= style attribute 214
LINK statement, TEMPLATE procedure 184–187
LINKCOLOR= style attribute 207
LIST statement, TEMPLATE procedure
   about 178, 187–190
   example using 15, 19
   SORT= option 188
   STATS= option 187–188
   STORE= option 187
   WHERE= option 190
LISTENTRYANCHOR= style attribute 207
listing styles 15–19
LISTSTYLEIMAGE= style attribute 207
LISTSTYLETYPE= style attribute 207–208
M
macro variables 135–138
managing templates
   DELETE statement (TEMPLATE) 183–184
   LINK statement (TEMPLATE) 184–187
   LIST statement (TEMPLATE) 187–191
   PATH statement (TEMPLATE) 191
   SOURCE statement (TEMPLATE) 191–193
   template stores 19, 177–183
MARGIN= style attribute 208
MARGINBOTTOM= style attribute 208
MARGINLEFT= style attribute 208
MARGINRIGHT= style attribute 208
MARGINTOP= style attribute 208
MARKERSIZE= style attribute 214
MARKERSYMBOL= style attribute 214–217
MERGE= table column attribute 117
MINOR= style attribute 217
MISSTEXT= option, TABLE statement (TABULATE) 73
MVAR statement, TEMPLATE procedure 110, 135–137
MYLIB.TEMPLAT template store 180–182

**N**
NAME= option, ODSTABLE procedure 7, 131, 196
NEUTRALCOLOR= style attribute 217
NEWPAGE= table attribute 121
NMVAR statement, TEMPLATE procedure 110, 135
NOTES statement, TEMPLATE procedure 110, 118

**O**
ODS ESCAPECHAR statement 79–80
ODS FILE PRINT statement
   COLUMNS= option 134, 138–139
   DYNAMIC= option 135–137
ODS Graphics Designer 8
ODS HTML statement
   CSSSTYLE= option 24, 27
   STYLE= option 9, 14, 17, 29, 32, 179–180, 182
   STYLESHEET= option 21, 27
ODS Path 178–183
ODS PATH statement
   access mode 179
   APPEND option 181
   PREPEND option 181
   REMOVE option 181
   syntax 178–180
ODS PDF statement 15, 17, 32
ODS RTF statement 17
ODSTABLE procedure
   about 7, 195–196
   NAME= option 7, 131, 196
   style overrides and 47
   table templates and 97–99, 104, 110
   olives (SAS/GRAPH color names) 226
   oranges (SAS/GRAPH color names) 226
ORDER_DATA= table attribute 121
overriding styles
See style overrides

**P**
PADDDING= style attribute 208
PADDDINGBOTTOM= style attribute 208
PADDDINGLEFT= style attribute 208
PADDDINGRIGHT= style attribute 208
PADDDINGTOP= style attribute 208
PAGEBREAKHTML= style attribute (HTML output) 211
PageNo style element 35
<PARENT> keyword 77–78
PARENT= attribute
   for table columns 117
   for table headers and footers 109
   for tables 121, 143
PARENT= statement, DEFINE STYLE statement (TEMPLATE) 40–43
PATH statement, TEMPLATE procedure 191
PDF output examples 15, 17, 32, 87–91
pinks (SAS/GRAPH color names) 227
POSTHTML= style attribute 209
POSTIMAGE= style attribute 209
POSTTEXT= style attribute 209
pre-defined styles 14–19
PREFORMATTED= attribute
   for table columns 117
   for table headers and footers 109
PREHTML= style attribute 208
PREIMAGE= style attribute 208
PRE_MERGE= table column attribute 117
PREPEND option, ODS PATH statement 181
PRETEXT= style attribute 209
PRINT= attribute
   for crosstabs 161
   for table columns 117, 120, 169–170
   for table headers and footers 109
PRINT procedure
   ID statement 57–59
   reporting procedures styles 57–62
PRINT procedure (continued)
  specifying styles for table regions 162
  STYLE= option 57–59
  SUM statement 57–59
  templates and 148, 177
  VAR statement 57–59
PRINT_FOOTERS= table attribute 121
PRINT_HEADERS= attribute
  for table columns 117
  for tables 121
procedure tables 148–167
purples (SAS/GRAPH color names) 228
PUT‘ ‘; function 141
PUT() function 55
PUT statement 140–141

R

reds (SAS/GRAPH color names) 229
REMOVE option, ODS PATH statement 181
%render macro 7, 47, 132
REPLACE statement 198
REPORT procedure
  COMPUTE statement 67
  reporting procedures styles 62–68
  specifying styles for table regions 162
  STYLE= option 63–65
  templates and 148, 177
reporting procedures styles
  about 56–57
  PRINT procedure 57–62
  REPORT procedure 62–68
  TABULATE procedure 68–78
RESOLVE() function 53
_ROW_ built-in variable 128
RowHeader style element 23
_ROW_LABEL_ built-in variable 156
_ROW_NAME_ built-in variable 156
ROWS_HEADER= crosstabs attribute 157, 163
ROW_VAR_STYLE= crosstabs attribute 162
RSASUSER system option 180
RTF output example 17
RULES= style attribute 209–210

S

SAMPLESIZE dynamic variable 156
SAS Enterprise Guide Style Manager 20–28
SAS formats 52–53
SAS/GRAPH color names
  about 219
  basic hues 220
  blacks 220
  blues 221–222
  browns 222–223
  grays 223–224
  greens 224–226
  olives 226
  oranges 226
  pinks 227
  purples 228
  reds 229
  violets 229
  whites 230
  yellows 230–231
SAS procedure tables 148–167
SASHELP.TMPLMST template store 178–180
SASUSER.TEMPLAT template store 178–183
_SELF_ option, STYLE statement
  (TEMPLATE) 39
SGPLOT procedure 8
SGRENDER procedure 8
SGSCATTER procedure 8
SORT= option, LIST statement (TEMPLATE) 188
SOURCE statement, TEMPLATE procedure
  about 37, 178, 184, 191–193
  FILE= option 192
  WHERE= option 192
SPLIT= attribute
  for table footers 109
  for table headers 109
SPLIT= header attribute 114
START= header attribute 105–106
STARTCOLOR= style attribute 217
StatGraph templates
  about 5, 8–9
  grouping 178
STATS= option, LIST statement (TEMPLATE) 187–188
STORE= option, LIST statement (TEMPLATE) 187
STRATAVARIABLELABELS dynamic variable 156
STRATAVARIABLENAMES dynamic variable 156
STRATNUM dynamic variable 156
STYLE= attribute
  for crosstabs 162–163
  for table columns 117, 123–125, 129
  for table headers and footers 110, 123–125, 162–163
  for tables 121, 123, 129
style attribute references 44–46
style attributes
  about 28
  common 200–210
  dynamic values 51–56
  graph-specific 212–218
  HTML-specific 211
  inheriting 44
  name changes for 196–197
  style elements and 29, 31
  types of 199–200
  version compatibility 196–197
_STYLE_ built-in variable 128
style definition
  about 29
  inheritance 40–43
Style Editor window 21–27
style elements
  about 29
  creating 31
  defining multiple elements simultaneously 197–198
  inheriting 40–43
  style definitions and 29, 31
Style Manager 20–28
STYLE= option
  ODS HTML statement 9, 14, 17, 29, 32, 179–180, 182
  ODS PDF statement 15, 17, 32
  ODS RTF statement 17
  PRINT procedure 57–59
  REPORT procedure 63–65
style overrides
  about 56–57
  general form for 122–123
  PRINT procedure 57–62
  REPORT procedure 62–68
  table templates and 122–130
  TABULATE procedure 68–78
  traffic lighting and 46–56, 122–130
STYLE statement, TEMPLATE procedure
  FROM option 37–41
  _SELF_ option 39
  style elements and 29, 31
  syntax for 37–40
style templates 5, 9–10, 178
STYLE_PRECEDENCE= option, TABLE statement (TABULATE) 78
styles
  defining new 19–46
  examples using 81–92
  for reporting procedures 56–78
  inline formatting 79–81
  listing 15–19
  overriding 46–56
  pre-defined 14–19
  traffic lighting for 46–56
  version compatibility 196–198
styles.css file 21
SUM statement, PRINT procedure 57–59
SYMGET() function 51
SystemTitle style element 35, 84
T
  table attributes 121–122, 135
  table column attributes 116–118, 124–125
  table columns
    See columns
  table footer attributes 109–111, 135
  table footers
    defining 103–111
table footers (continued)  
  inheritance and 145

table header attributes  
  about 105–106, 114  
  common 109–111  
  setting values at run time 135

table headers  
  defining 103–111  
  inheritance and 145

TABLE statement, TABULATE procedure  
  ALL option 74  
  BOX= option 73–74  
  MISSTEXT= option 73  
  style overrides and 68–76  
  STYLE_PRECEDENCE= option 78

Table style element 23, 31

table templates  
  about 95  
  binding data sets to 96–99, 130  
  built-in variables 103, 128  
  computed column values as column data 118–121  
  DATA _NULL_ statement and 7, 130–142  
  DATA step tricks 140–142  
  defining and rendering 5–7  
  defining columns 111–118  
  defining headers and footers 103–111  
  defining tables 96–102  
  examples using 167–176  
  FREQ procedure crosstabulation tables 153–167  
  generic columns in 139–140  
  grouping 178  
  inheritance and 142–147  
  modifying SAS procedure tables 148–167  
  style overrides and 47–50, 122–130  
  table attributes 121–122  
  traffic lighting and 122–130  
  variables in 135–138

tables  
  crosstabulation 153–167  
  defining 96–102

TABLES statement, FREQ procedure  
  about 160–161  
  CELLCHI2 option 160  
  CUMCOL option 160  
  DEVIATION option 160  
  EXPECTED option 160  
  TOTPCT option 160

TABULATE procedure  
  CLASS statement 71  
  CLASSLEV statement 71, 78  
  KEYWORD statement 71, 73  
  reporting procedures styles 68–78  
  specifying styles for table regions 162  
  TABLE statement 68–76  
  templates and 148, 153, 177  
  VAR statement 71

TAGATTR= style attribute (HTML output) 211

tagset templates 5, 9–12, 178

tagsets 10–12

TAGSETS.STYLE_POPUP tagset 33, 84

Template Browser 178

TEMPLATE procedure  
  about 1–2, 198  
  CELLSTYLE AS statement 125–130, 146–147, 166, 169  
  CELLVALUE statement 159–162  
  CLASS statement 29, 37, 39–43  
  COLUMN statement 97–103, 111–112, 168  
  compatibility with previous versions 195–198  
  COMPUTE AS statement 118–119  
  DEFINE CROSSTABS statement 153–158  
  DEFINE FOOTER statement 103, 155  
  DEFINE HEADER statement 103, 107–109, 154–157  
  DEFINE statement 3–4, 29, 103  
  DELETE statement 167, 183–184  
  DYNAMIC statement 103, 110, 135–137  
  EDIT statement 150–152  
  END statement 3–4, 29, 103  
  LINK statement 184–187  
  LIST statement 15, 19, 178, 187–190
MVAR statement 110, 135–137
NMVAR statement 110, 135
NOTES statement 110, 118
PARENT= statement 40–43
PATH statement 191
SOURCE statement 37, 178, 184, 191–193
STYLE statement 29, 31, 37–41
TEXT statement 103, 156
TRANSLATE INTO statement 119–121

template stores 19, 177–183
templates
See also specific types of templates
about 1–2
data-dependent 5–9
data-independent 9–12
managing 177–193
structure of 2–5
types of 5–12
TEXT statement, TEMPLATE procedure 103, 156
TEXTALIGN= style attribute 209
TEXTDECORATION= style attribute 209
TEXTINDENT= style attribute 209
TEXTJUSTIFY= style attribute 209
TEXT_SPLIT= table column attribute 117
TICKDISPLAY= style attribute 218
TOTPCT option, TABLES statement (FREQ) 160
traffic lighting
reporting procedure styles and 61, 66, 74
style overrides and 46–56, 122–130
table templates and 122–130
TRANSLATE INTO statement, TEMPLATE procedure 119–121
TRANSPARENCY= style attribute 218

V

_VAL_ built-in variable 55, 128
_VAL_ keyword 120
VAR statement
  PRINT procedure 57–59
  TABULATE procedure 71
variables in table templates 135–138
VARNAME= table column attribute 117
version compatibility
  running examples 195–196
  style considerations 196–198
VERTICALALIGN= style attribute 210
violets (SAS/GRAPH color names) 229
VISITEDLINKCOLOR= style attribute 210
VJUST= attribute
  for styles 210
  for table columns 117
  for table headers and footers 110

W

WHERE= option
  LIST statement (TEMPLATE) 190
  SOURCE statement (TEMPLATE) 192
whites (SAS/GRAPH color names) 230
WHITESPACE= style attribute 210
WIDTH= style attribute 210

Y

yellows (SAS/GRAPH color names) 230–231

Symbols

/ (forward slash) 31
^ (caret) 79
{} (curly braces) 79
= (equals) sign 28
Kevin D. Smith has been a software developer at SAS since 1997. He has supported PROC TEMPLATE and other underlying ODS technologies for most of that time. Kevin has spoken at numerous SAS Global Forums as well as at regional and local SAS users groups with the “From Scratch” series of presentations that were created to help users of any level master various ODS technologies. Kevin is also the author of many of the ODS tip sheets available on the SAS support Web site.

Learn more about this author by visiting his author page at http://support.sas.com/smithk. There you can download free chapters, access example code and data, read the latest reviews, get updates, and more.
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore
for additional books and resources.