

Paper 221-2007

PROC TEMPLATE Tables from Scratch

Kevin D. Smith, SAS Institute Inc., Cary, NC

ABSTRACT

Did you know that nearly every table created by the SAS® System uses a template to describe what it looks like? Did you know that you can modify these templates, or even write your own templates to create custom tables? In this paper, you learn how to create and modify table templates, including how to add, remove, and move columns as well as headers and footers. You also learn how to apply styles, formats, and other visual effects, all from scratch.

INTRODUCTION

Although many people don't realize this, almost every table that is generated by the SAS System has a table template behind it that defines what that table should look like¹. These table templates include information such as which columns should appear, where they should appear in relation to each other, what the column header should contain, what headers and footers should be on the table, and such style information as what colors and fonts to use. Because you can edit these templates with PROC TEMPLATE, you can actually change the way SAS tables are structured. In addition, you can use `data _null_` with table templates to construct a table completely from scratch. This paper walks you through how to create your own tables, how to apply data to a template, and how to modify the templates used by SAS.

TEMPLATE SAFARI

Before we get started writing table templates, it might be a good idea to explore some of the SAS-supplied table templates to get an idea of what they look like. It is also a good excuse for getting to know the Template Browser. If you type `odstemplates` in the run box of the SAS window, the Template Browser opens (Figure 1).

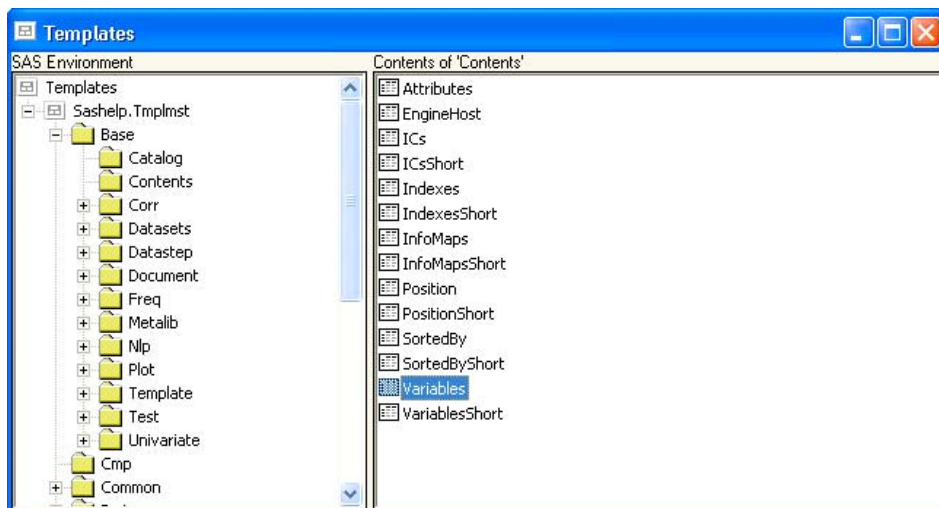


Figure 1. SAS Template Browser Window

You will see at least one template store² in the left pane of the Template Browser called `Sashelp.Tmplmst`. This template store contains all of the SAS-supplied templates. There are more than just table templates in the folders, though. These folders contain table templates, style templates, STATGRAPH templates, and tag sets³. Table templates have an icon that looks like a little table. If you expand the Base folder and select the Contents folder, you'll see all of the templates that belong to PROC CONTENTS. Double-clicking on the Variables template displays the source of the table template (shown below)⁴.

¹ The only table-producing procedures that don't use templates are PROC PRINT, PROC REPORT, and PROC TABULATE. They all create their own table structures through statements and options.

² A *template store* is simply a specially formatted item store created by PROC TEMPLATE.

³ The folder structure you see is roughly organized by product and then by procedure.

⁴ You can also use PROC TEMPLATE's SOURCE statement to display the source of a template. See the *SAS 9.1.3 Output Delivery System: User's Guide, Volumes 1 and 2* (SAS Institute 2006) for details.

```

define table Base.Contents.Variables;
  notes "Contents Variables";
  dynamic name_width name_width_max label_width label_width_max;
  column Num Variable Type Len Pos Flags Format Informat Label Transcode;
  header main;

  define main;
    text "Alphabetic List of Variables and Attributes";
    space = 1;
    spill_adj;
    spill_margin;
  end;

  define Num;
    header="/#/" ;
    style=RowHeader;
    id;
  end;

  define Variable;
    header="Variable";
    width_max=name_width_max;
    width=name_width;
  end;

  define Type;
    header="Type";
  end;

  define Len;
    header="Len";
  end;

  ... content snipped ...

  col_space_max = 4;
  col_space_min = 1;
  newpage=off
  balance;
end;

```

If you run PROC CONTENTS on the Sashelp.Class data set using the following code, you'll see the table that is created using the previous template.

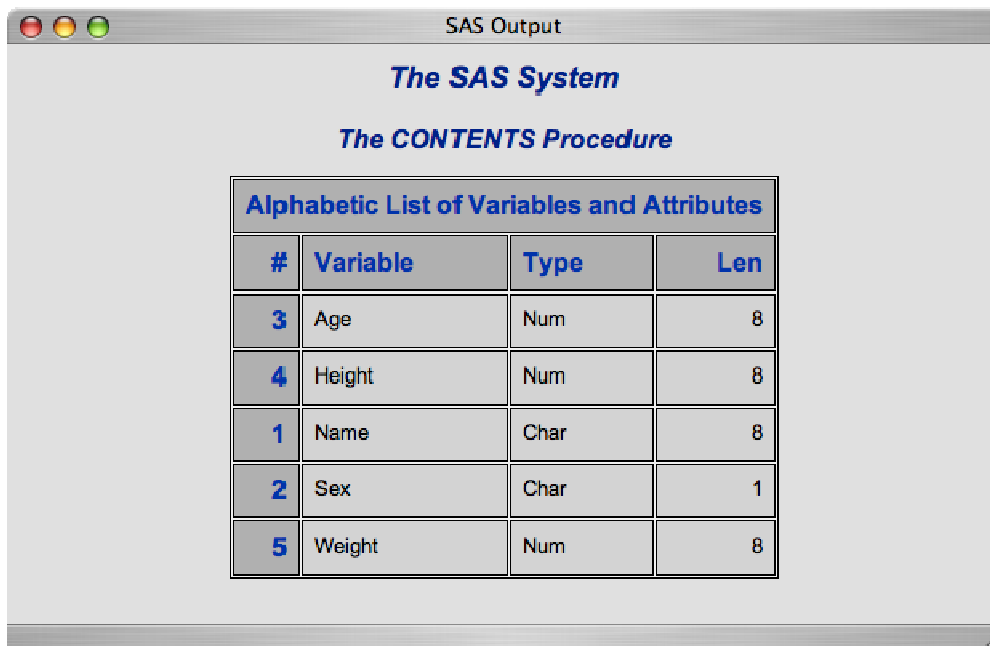
```

ods select variables;
proc contents data=sashelp.class;
run;

```

Figure 2 shows the output from this code in HTML⁵.

⁵ You can generate HTML output by wrapping your procedure code with `ods html file='filename.html';` and `ods html close;`, where *filename.html* is the name of the output file.



The screenshot shows a SAS Output window titled 'SAS Output'. Inside, the text 'The SAS System' is displayed in a large, bold, blue font. Below it, 'The CONTENTS Procedure' is displayed in a smaller, bold, blue font. The main content is a table titled 'Alphabetic List of Variables and Attributes' in a bold, blue font. The table has four columns: '#', 'Variable', 'Type', and 'Len'. The rows are sorted alphabetically by variable name: Age, Height, Name, Sex, and Weight. The 'Len' column shows the length of each variable: 8 for Age, Height, and Name; 1 for Sex; and 8 for Weight.

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

Figure 2. SAS Output in HTML Format

Although we don't expect you to completely understand the table template for this table, you should be able to make some connections between the provided code and the table it produces. For example, the table has a header with the text "Alphabetic List of Variables and Attributes." If you look at the template that created this table, you'll see that same text in the define block named main. You may also notice that there are four columns in the output table with the headers #, Variable, Type, and Len. In the table template, you'll see four define blocks with HEADER= statements containing each of those four strings.

Other things in that table template don't make sense yet, but that's OK because we're just getting started. The following sections of this paper show you how to define headers and columns to construct your own tables and how to apply data to them.

USING A TABLE TEMPLATE

Before we dive into writing table templates, it might be nice to know how to use one so you can test your templates as we are working through them. While procedures can tell the Output Delivery System (ODS) directly which table template to use and what data to use, you'll have to use the DATA step to apply a data set to a template. The code to do this isn't very long, but it is a bit quirky. Therefore, we've wrapped it up in a nice little macro shown here:

```
%macro do_table(dataset, template);
data _null_;
  set &dataset;
  file print ods=(template="&template");
  put _ods_;
run;
%mend;
```

To apply a data set to a table template, you call the macro with the data set name as the first argument and the table template name as the second argument. The following code applies a data set named MyDataSet to a table template named MyTemplate:

```
%do_table(mydataset, mytemplate);
```

THE BASICS

This section deals with the basics of defining a table, including defining its columns, headers, and footers. The information in this section is the foundation of all table templates.

DEFINING A TABLE

The code that we saw in the Template Browser was actually PROC TEMPLATE code. In order to compile that code, you would have to wrap it in a PROC TEMPLATE and a RUN statement, as follows:

```
proc template;
  ... template code ...
run;
```

You also may have noticed that there are a lot of blocks of code surrounded by DEFINE and END statements. This is the general form for defining templates of any type. Anything between the DEFINE and END statements, whether it is an attribute such as `space=1` or `style=RowHeader` or another DEFINE block, becomes a part of that template.

In addition, the first line of the table template shown previously contains two pieces of information in addition to the keyword DEFINE:

```
define table Base.Contents.Variables;
```

The first piece of information, **table**, tells PROC TEMPLATE what type of template to create. In this case, it is a table template. We also discuss header, footer, and column templates in this paper. The second piece of information, **Base.Contents.Variables**, is the name of the template. This is the name that a procedure uses when it tells ODS to render a particular table. It is also the name that you will use when you apply a data set to a table template using the macro from the first section.

The DEFINE statements within another template don't always require you to specify a template type (as you can see from the example in the "Template Safari" section), but it doesn't hurt to put it there for clarity. In these cases, the type of the template can be inferred from the enclosing template. In the case of tables, there are COLUMN, HEADER, and FOOTER statements to indicate the nested template types (as you'll see in the following sections).

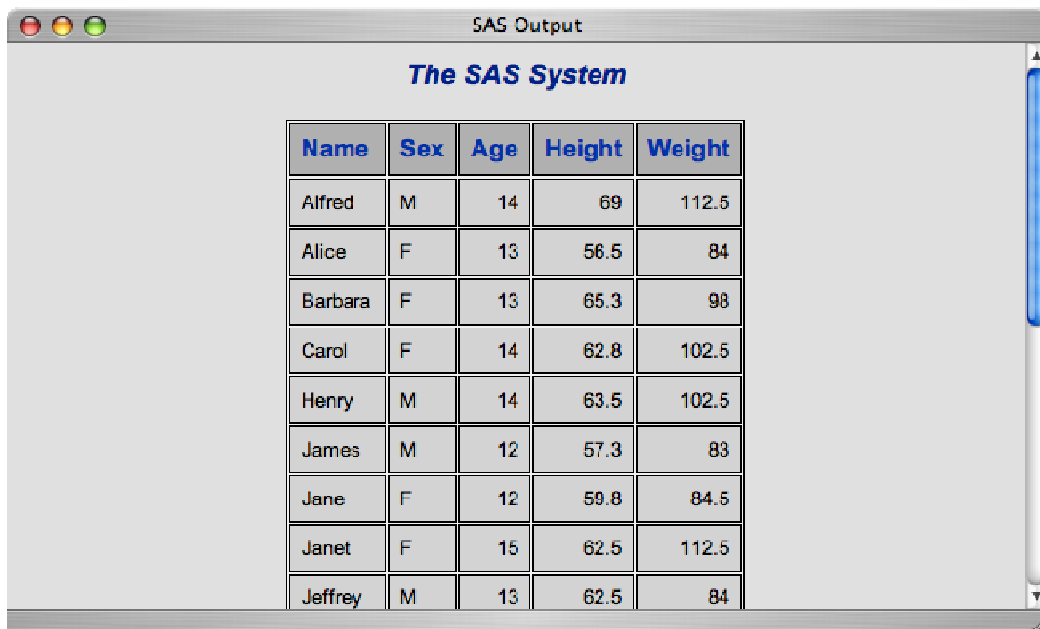
Now that we've got the foundation information out of the way, let's start building some tables! The first thing we need is a data set that we want to display. Sashelp.Class is always a nice and simple data set to start with. If you ran the PROC CONTENTS example in the "Template Safari" section, you've seen that it has five variables: two character variables, Name and Sex, and three numeric variables, Age, Height, and Weight. You can use the COLUMN statement to build a very simple table that displays those columns with the default attributes. The simplest form of the COLUMN statement lists the names of the variables that you want to display⁶. The following example shows a table definition with a simple COLUMN statement that includes all of the columns in the Sashelp.Class data set:

```
define table mytable;
  column name sex age height weight;
end;
```

If you run this definition in PROC TEMPLATE and then use the %DO_TABLE macro from the previous section, as shown here, you get the output that follows (Figure 3):

```
proc template;
  define table mytable;
    column name sex age height weight;
  end;
run;
ods html;
%do_table(sashelp.class, mytable);
ods html close;
```

⁶ If you don't specify any columns, you won't get a table.



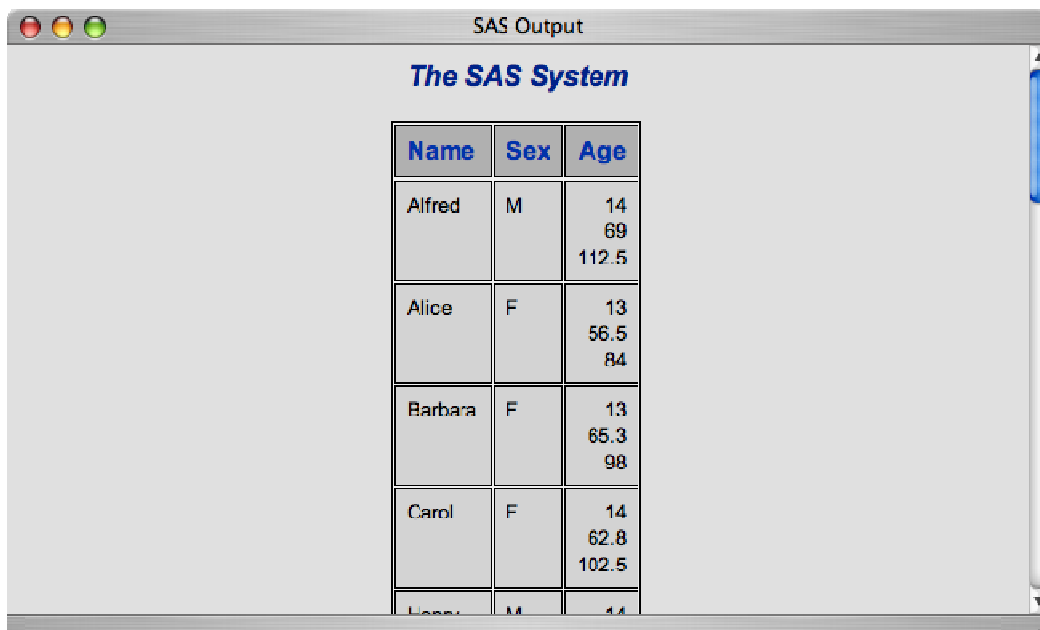
Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84

Figure 3. SAS Output Table in HTML Using the COLUMN Statement

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 column:

```
column name sex (age height weight);
```

The output is shown in Figure 4.



Name	Sex	Age
Alfred	M	14 69 112.5
Alice	F	13 56.5 84
Barbara	F	13 65.3 98
Carol	F	14 62.8 102.5
Henry	M	14 63.5 102.5
James	M	12 57.3 83
Jane	F	12 59.8 84.5
Janet	F	15 62.5 112.5
Jeffrey	M	13 62.5 84

Figure 4. SAS Output in HTML Stacking Values within Column Cells – Age Over Height Over Weight

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

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:

```
column name (sex height) * (age weight);
```

Figure 5 shows the output created by this COLUMN statement.

The SAS Output window displays a table titled "The SAS System". The table has three columns: Name, Sex, and Height. The data is as follows:

Name	Sex	Height
Alfred	M	69 112.5
Alice	F	58.5 84
Barbara	F	65.3 96
Carol	F	62.8 102.5
Henry	M	63.5 102.5
James	M	57.3 83

Figure 5. SAS Output in HTML Stacking Groups of Columns – Sex Over Age and Height Over Weight

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.

```
column name (sex height) *  
            (age weight);
```

Although the COLUMN statement determines which columns go into a table, other attributes can also affect how the table is rendered. Most of the table attributes are for advanced techniques and are beyond the scope of this paper. You can read more about them in the *SAS 9.1.3 Output Delivery System: User's Guide, Volumes 1 and 2* (SAS Institute Inc. 2006).

The next step in building tables is adding headers and footers, which is the topic of the next section.

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.

Defining a header is much like defining a table. A DEFINE statement that has a type and name begins the definition, and an END statement ends 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 with the TEXT statement, as shown here:

```
define header myheader;
  text "Class Information";
end;
```

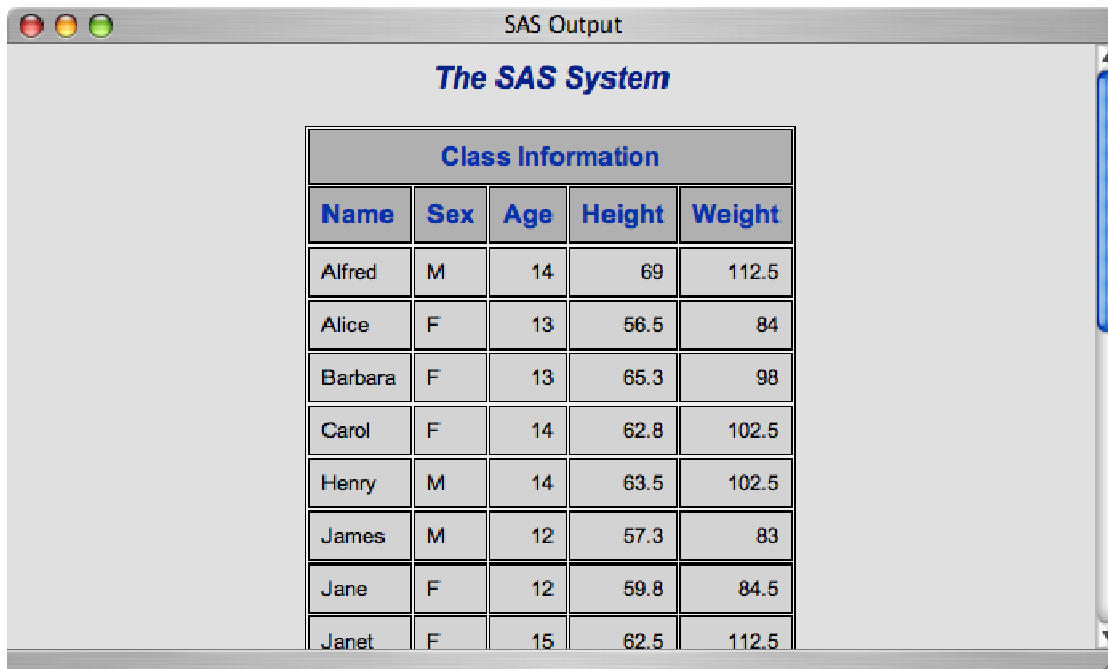
Now that we have a header definition, we need to attach it to a table. This can be done in one of two ways: 1) use the HEADER statement⁷ or 2) define the header within a table definition. The HEADER statement is much like the simple form of the COLUMN statement because you list the names of the headers 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 1⁸:

```
define header my.header;
  text "Class Information";
end;
define table mytable;
  header my.header;
  column name sex age height weight;
end;
```

Here is an example of applying a header to a table using Method 2:

```
define table mytable;
  column name sex age height weight;
  define header myheader;
    text "Class Information";
  end;
end;
```

Here is what the HTML output looks like when you apply the Class data set to this template (Figure 6).



The screenshot shows a SAS Output window with a title bar containing three colored buttons (red, yellow, green) and the text 'SAS Output'. The main content area has a title 'The SAS System' in blue. Below the title is a table with a header 'Class Information' in blue. The table has five columns: Name, Sex, Age, Height, and Weight. The data rows are as follows:

Class Information				
Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5

Figure 6. SAS Output in HTML Applying a Header

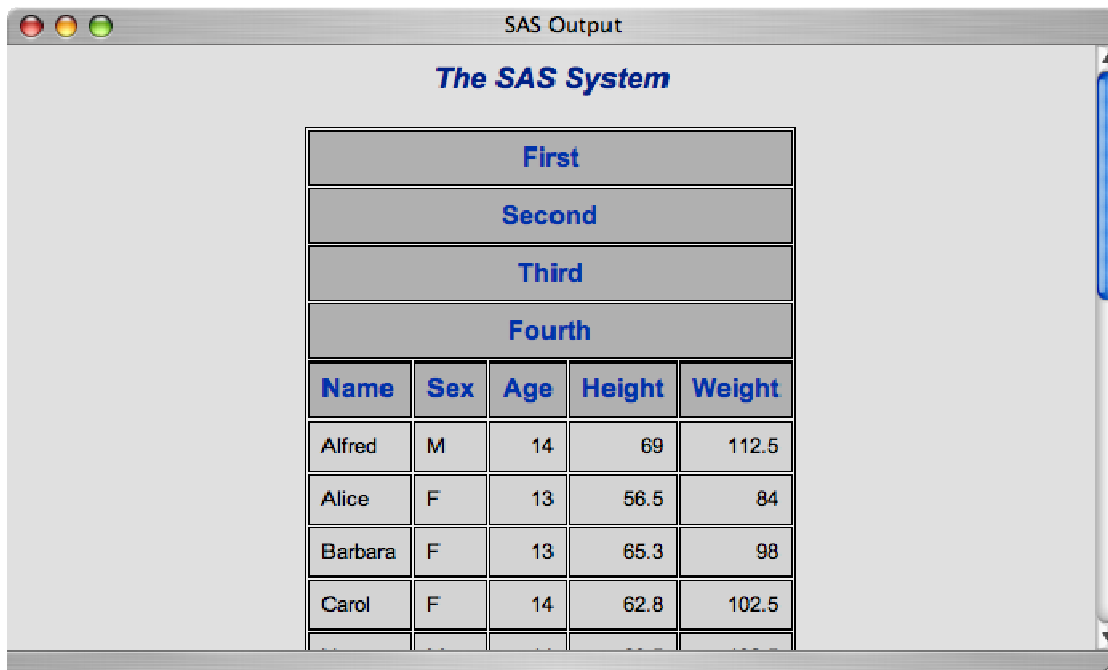
⁷ For footers, you would use the FOOTER statement.

⁸ Note that PROC TEMPLATE requires that headers that are defined outside of the table definition have more than one level (that is, they need to have at least one dot (.) in them).

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.

```
define table mytable;
  column name sex age height weight;
  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;
```

This code produces the output shown in Figure 7.

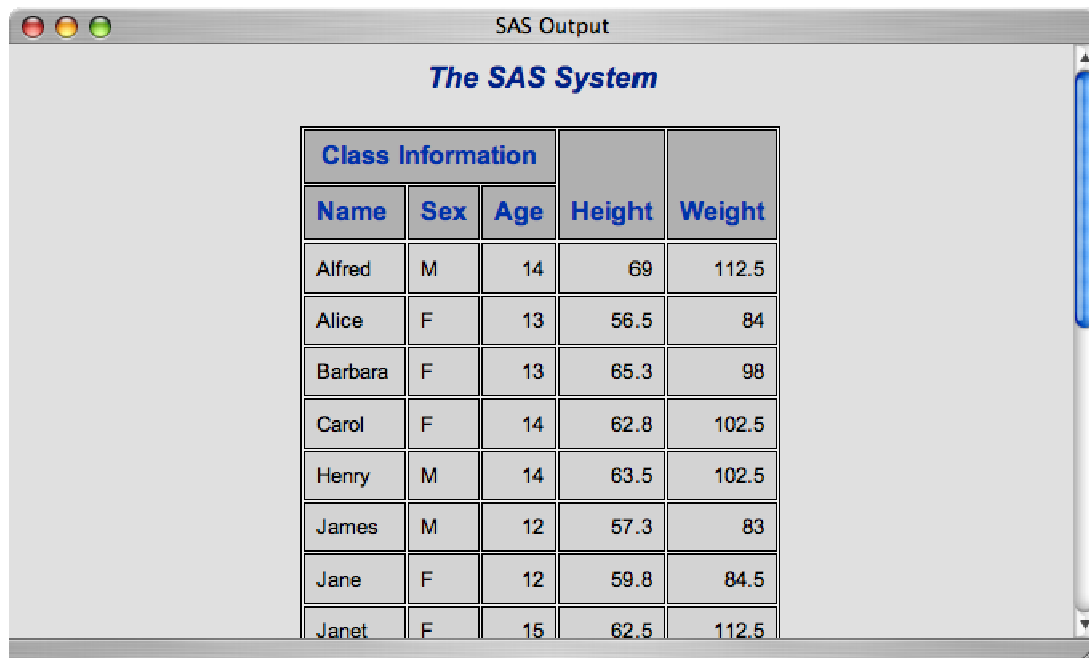


First				
Second				
Third				
Fourth				
Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5

Figure 7. SAS Output in HTML Applying Multiple Headers

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 (Figure 8 shows the output):

```
define table mytable;
  column name sex age height weight;
  define header myheader;
    text "Class Information";
    start = name;
    end = age;
  end;
end;
```

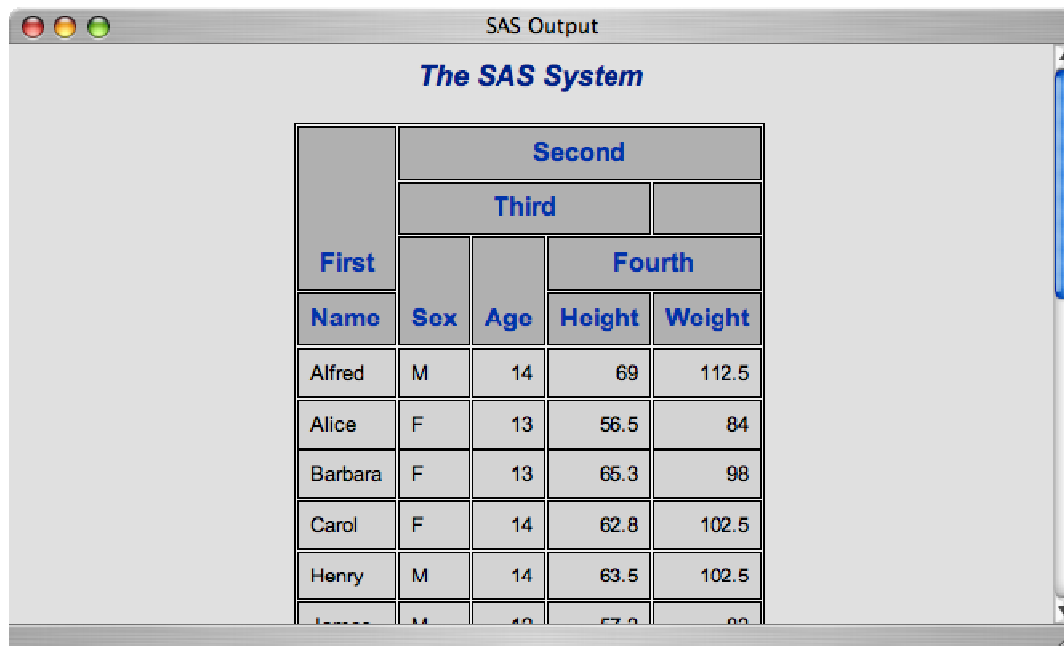
Class Information			Height	Weight
Name	Sex	Age		
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5

Figure 8. SAS Output in HTML Showing Header Span Variations

Now let's combine stacking with spanning. This is kind of like playing the game Tetris. 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.

```
define table mytable;
  column name sex age height weight;
  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;
end;
```

Figure 9 shows the output.



The screenshot shows a window titled "SAS Output" containing a table titled "The SAS System". The table has a complex header structure using colspan and rowspan attributes. The first column is labeled "First" and contains names. The second column is labeled "Second" and contains "Sex". The third column is labeled "Third" and contains "Age". The fourth and fifth columns are grouped under a label "Fourth" and contain "Height" and "Weight" respectively. The data rows show names, sex, age, height, and weight for several individuals.

First	Second			
	Third			
	Sex	Age	Fourth	
			Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	14	67.0	99

Figure 9. SAS Output in HTML Showing Header Span Variations

There are various other attributes that you can use on headers. You can read more about them in the *SAS 9.1.3 Output Delivery System: User's Guide, Volumes 1 and 2* (SAS Institute Inc. 2006). The more common attributes are described here:

JUST=

specifies the horizontal alignment of the header content. The value can be LEFT, RIGHT, or CENTER.

SPLIT=

specifies a character that will be treated as a line break. For example, if you set SPLIT='*' in your header, you will get a line break wherever a * appears in the header text.

VJUST=

specifies the vertical alignment of the header content. The value can be TOP, CENTER, or BOTTOM.

Almost all of the discussion in this section deals with headers; we didn't say much about footers. Footers are identical to headers. The only difference is that they are located at the bottom of the table rather than at the top. Figure 10 shows the output of the last header example with all of the headers changed to footers.

The screenshot shows a window titled "SAS Output" containing an HTML table. The table has five columns. The first seven rows contain data for individuals: Louise (F, 12, 56.3, 77), Mary (F, 15, 66.5, 112), Philip (M, 16, 72, 150), Robert (M, 12, 64.8, 128), Ronald (M, 15, 67, 133), Thomas (M, 11, 57.5, 85), and William (M, 15, 66.5, 112). The last row is a footer with four cells: "First" (rowspan=3), "Second" (colspan=2), "Third" (colspan=2), and "Fourth" (colspan=2). The "Second" and "Fourth" cells are highlighted in blue.

Louise	F	12	56.3	77
Mary	F	15	66.5	112
Philip	M	16	72	150
Robert	M	12	64.8	128
Ronald	M	15	67	133
Thomas	M	11	57.5	85
William	M	15	66.5	112
First	Second		Third	
			Fourth	

Figure 10. SAS Output in HTML Showing Footer Span Variations

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

We have discussed columns once already when we went over the COLUMN statement of the table definition. Although this 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:

- Use the COLUMN statement to create columns based on the data columns in a data set⁹
- 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 original table definition example:

```
define table mytable;
  column name sex age height weight;
end;
```

The following table template is equivalent but isn't nearly as convenient:

```
define table mytable;
  define column name;
end;
define column sex;
end;
define column age;
end;
define column height;
```

⁹ This is the method described in the section on table definitions.

```

end;
define column weight;
end;
end;

```

Using a hybrid approach, we 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:

```

define table mytable;
  column name sex age height weight;
  define height;
  end;
  define weight;
  end;
end;

```

Note that when you use both a COLUMN statement and column definitions, you don't need to specify the template type in the column definition. PROC TEMPLATE already knows that it is a column because of the information in the COLUMN statement.¹⁰ You can still include the template type for clarity if you want to.

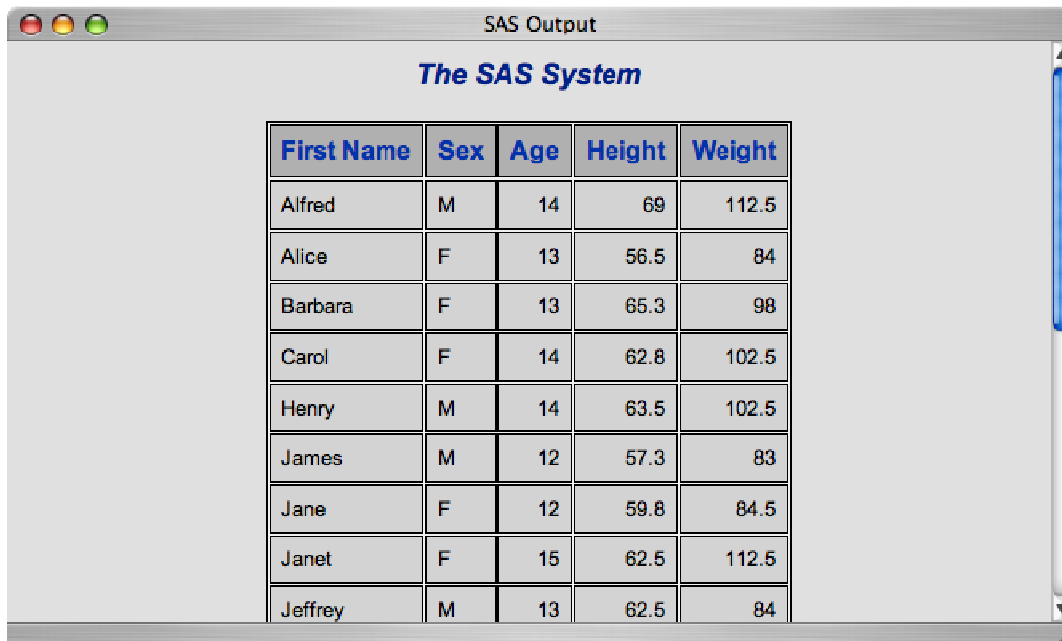
All of the previous column templates were kind of nonsensical because they didn't do anything other than the default behavior anyway. Let's say that we wanted to change the column header for the column Name to First Name. The simplest way would be to add a column definition for that column and set the HEADER= attribute.

```

define table mytable;
  column name sex age height weight;
  define name;
    header = 'First Name';
  end;
end;

```

The resulting output is shown in Figure 11.



First Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84

Figure 11. SAS Output in HTML Showing the Updated Column Header First Name

¹⁰ This is also true of header and footer definitions.

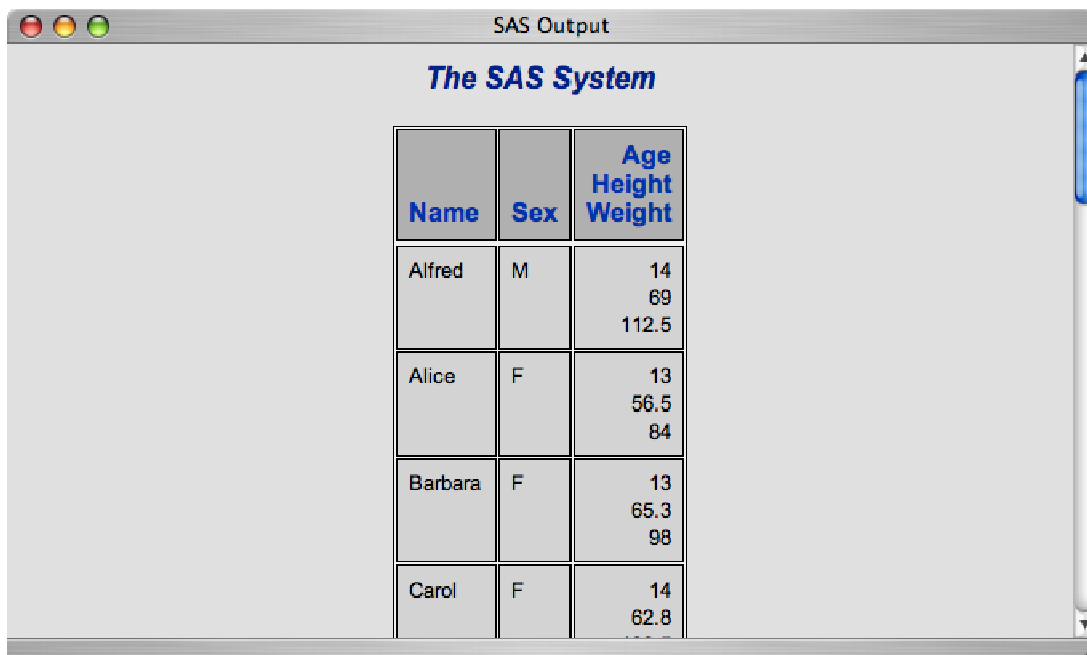
While this works for simple headers, you may want to use a full header definition instead of just a string. This form allows you to add styles and other visual effects to headers as we will see later. In the case of column headers, you must use the HEADER= attribute to associate a header definition with 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:

```
define table mytable;
  column name sex age height weight;
  define name;
    define header nameheader;
      text 'First Name';
    end;
    header = nameheader;
  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'll also use this opportunity to show off the SPLIT= header attribute described in the previous section.

```
define table mytable;
  column name sex (age height weight);
  define age;
    define header ageheightweight;
      text 'Age*Height*Weight';
      split = '*';
    end;
    header = ageheightweight;
  end;
end;
```

Figure 12 shows the output for this example.



The SAS Output window displays a table titled "The SAS System". The table has three columns: Name, Sex, and Age Height Weight. The data is as follows:

Name	Sex	Age Height Weight
Alfred	M	14 69 112.5
Alice	F	13 56.5 84
Barbara	F	13 65.3 98
Carol	F	14 62.8 100

Figure 12. SAS Output in HTML Showing Stacked Header and Column Information

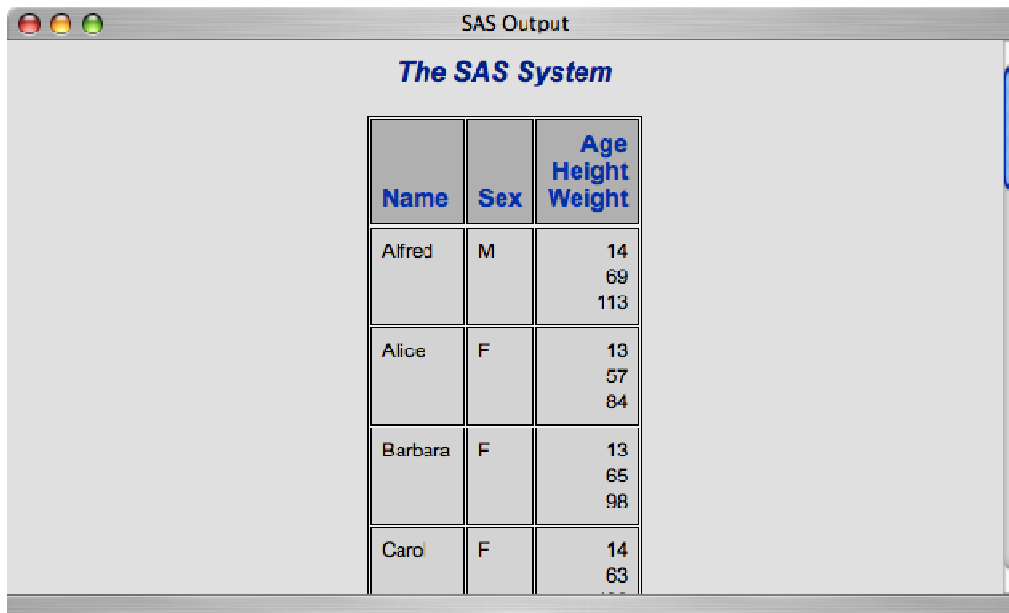
Another common thing to do with a column template is to change the data format. This is done using the FORMAT= attribute. The value is simply a SAS format. Let's continue to use the definition from the last example to try out the FORMAT= attribute. We want all of the numbers in our Age/Height/Weight column to line up. We can do this by formatting all of the numbers the same way. We'll simply make them all integers, as follows:

```

define table mytable;
  column name sex (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;

```

Figure 12 shows the output.



Name	Sex	Age Height Weight
Alfred	M	14 69 113
Alice	F	13 57 84
Barbara	F	13 65 98
Carol	F	14 63 100

Figure 13. SAS Output in HTML Showing Stacked Header and Column Information and Data Formats

Column attributes can do many other things. Here is a list of some of the more common attributes. Read the SAS 9.1.3 *Output Delivery System: User's Guide, Volumes 1 and 2* (SAS Institute Inc. 2006) for a complete list of column attributes.

BLANK_DUPS

specifies that only the first occurrence of consecutive same values should appear in a column.

JUST=

specifies the horizontal alignment of the cell content. The value can be LEFT, RIGHT, or CENTER.

PRINT_HEADERS

specifies whether or not to print the column header.

VJUST=

specifies the vertical alignment of the cell content. The value can be TOP, CENTER, or BOTTOM.

COMPUTED COLUMNS

Although you do need a data set to render a table template, you don't 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 statement that determines whether a column is associated with a data column or an expression is the COMPUTE-AS statement. This statement takes a standard WHERE expression as its argument. This 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 where-expr;
```

Since we've 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:

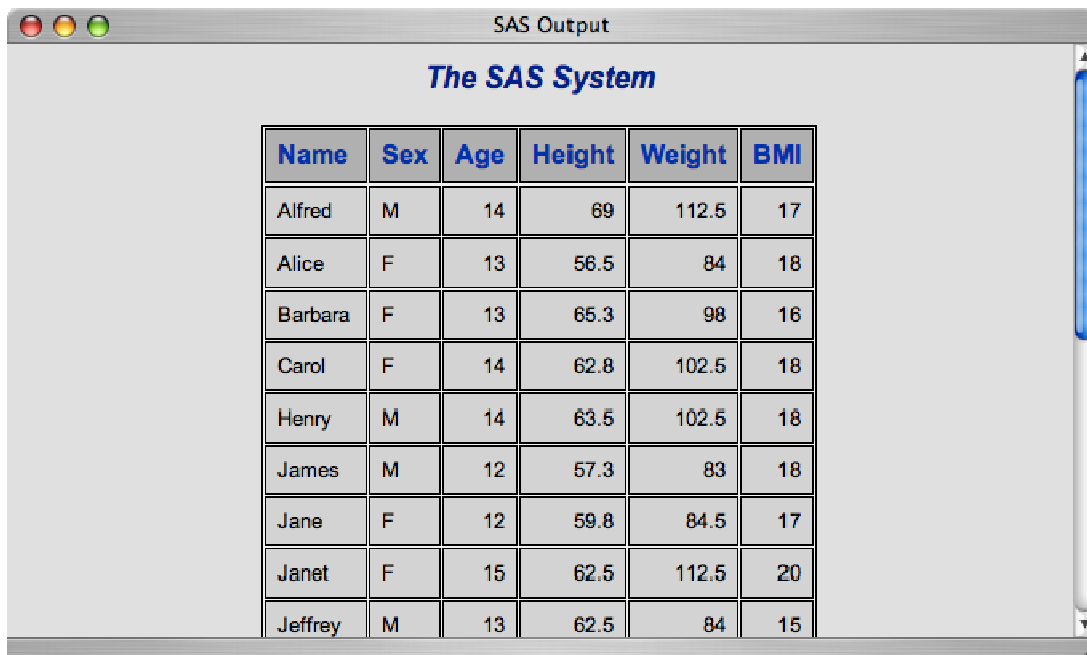
$$BMI = \frac{weight \cdot 703}{height^2}$$

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:

```
define table mytable;
  column name sex age height weight bmi;
  define bmi;
    compute as (weight * 703) / (height * height);
    header = 'BMI';
    format = 2.;
  end;
end;
```

Figure 14 shows the resulting output.



The SAS System

Name	Sex	Age	Height	Weight	BMI
Alfred	M	14	69	112.5	17
Alice	F	13	56.5	84	18
Barbara	F	13	65.3	98	16
Carol	F	14	62.8	102.5	18
Henry	M	14	63.5	102.5	18
James	M	12	57.3	83	18
Jane	F	12	59.8	84.5	17
Janet	F	15	62.5	112.5	20
Jeffrey	M	13	62.5	84	15

Figure 14. SAS Output in HTML Showing the Additional BMI Column

Although we clearly can't discuss every aspect of table, header, and column templates in a paper of this size, you should now have a good idea of how to create some useful tables using your own data sets. We aren't finished just yet, though. There are still some slightly more advanced techniques that apply to all template types that we'd like to cover, and we still haven't talked about changing the templates that SAS procedures use. These topics are all covered in the remainder of this paper.

ADVANCED TECHNIQUES

Once you are comfortable with creating table, header, and column templates as described in the previous sections, you may want to try out some more advanced techniques to make your tables really stand out from the crowd. The following sections discuss how to add styles to your tables, how to modify (or completely replace) table templates used by SAS procedures, and how to use inheritance so that you can re-use code between templates.

ADDING STYLE

Adding styles to parts of a table can help to increase the readability and the speed of comprehension of data in a table. Table templates allow you to add style in two ways. The first way is by adding *static styles*. These are styles that are applied to the table regardless of the content of the table. The second type is called *conditional styles*. These styles do change based on the data applied to the table template.¹¹ Let's start with static styles since they are simpler. We can then use what we learn with them to do conditional styling.

Static styles can be applied to table, header, footer, and column templates in the same way: the `STYLE=` attribute. To fully understand the `STYLE=` attribute, you would have to understand PROC TEMPLATE style templates as well. Instead, we'll just apply style overrides, which are simpler and more commonly used for this purpose anyway. The general form of the `STYLE=` attribute is as follows.

```
style = { style-attributes };
```

where style-attributes are the style attributes to be applied to the rendered output. This includes things like fonts, colors, and borders. Here is an example of a `STYLE=` attribute that sets the font to bold, the background color to red, and the foreground color to white:

```
style = { font_weight=bold background=red foreground=white };
```

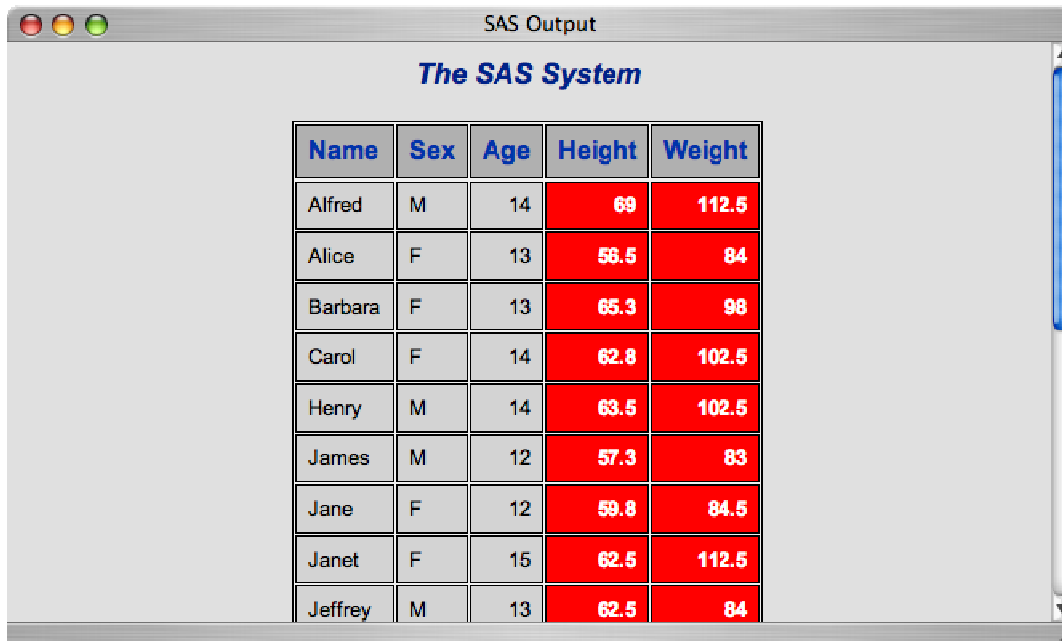
The style attributes that can be used in table templates are described in the *SAS 9.1.3 Output Delivery System: User's Guide, Volumes 1 and 2* (SAS Institute Inc. 2006) in the section on style definitions. We'll use a mere handful of the more common style attributes in the examples here.

As mentioned previously, the `STYLE=` attribute can be used on table, header, footer, or column templates. In the following code, we've put the `STYLE=` attribute on the Height and Weight columns:

```
define table mytable;
  column name sex age height weight;
  define height;
    style = { font_weight=bold background=red foreground=white };
  end;
  define weight;
    style = { font_weight=bold background=red foreground=white };
  end;
end;
```

Figure 15 shows the output using static styles.

¹¹ This technique is sometimes called *trafficlighting*.



The SAS System

Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	98
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84

Figure 15. SAS Output in HTML Applying Style to Height and Weight Data

Conditional styles use the style override syntax as well, but they require a condition to be applied. This condition is a WHERE expression, just like the expression in the COMPUTE-AS statement in column templates. Conditional styles can be applied on a table or column basis, and they are set using the CELLSTYLE-AS statement. The general form of the CELLSTYLE-AS statement is as follows:

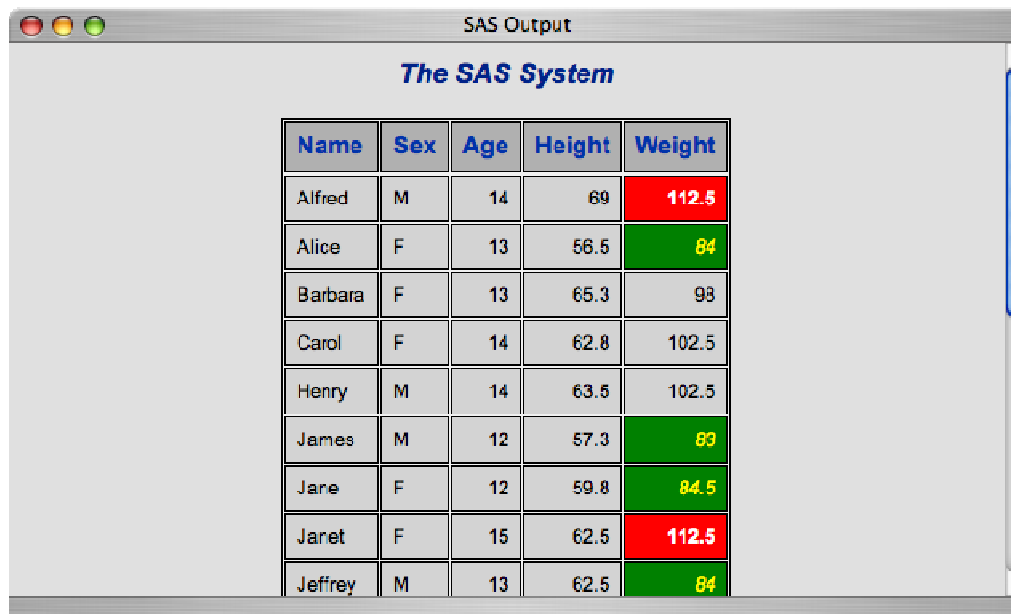
```
cellstyle where-expr-1 as { style-attributes },
      where-expr-2 as { style-attributes },
      ...
      where-expr-n as { style-attributes };
```

As you can see, you can have various expressions with different style attributes for each case. The CELLSTYLE-AS statement is executed for each cell in the table. Each expression is tested in order until one of the expressions returns a result that is interpreted as true. If the CELLSTYLE-AS statement finds one, the style attributes associated with that expression are applied and execution ends.

Let's use a CELLSTYLE-AS statement to color-code the Weight column in our SasHELP.CLASS table. Any weight that is greater than or equal to 110 will be made bold with a red background and white foreground. Any weight that is less than or equal to 90 will be italic with a green background and yellow foreground. The code to do this is shown here:

```
define table mytable;
  column name sex age height weight;
  define weight;
    cellstyle weight >= 110 as { font_weight=bold
                                background=red
                                foreground=white },
    weight <= 90 as { font_style=italic
                     background=green
                     foreground=yellow };
  end;
end;
```

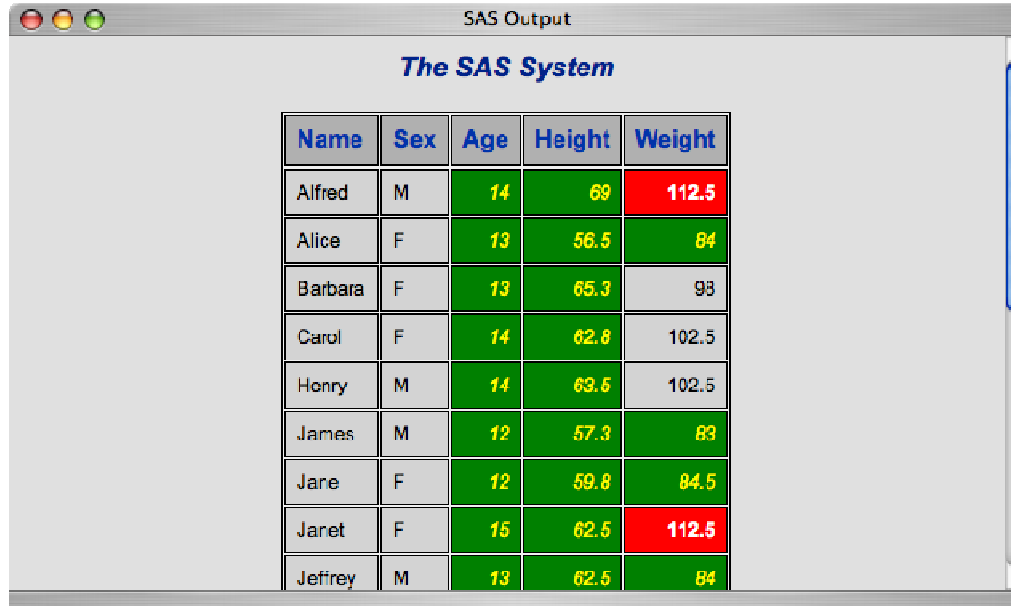
Here is the output (Figure 16).



Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	93
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Jared	F	15	62.5	112.5
Jeffrey	M	13	62.5	84

Figure 16. SAS Output in HTML Applying Color-Coding to Weight Data

There are a couple of things to note about the CELLSTYLE-AS statement. First, the expressions in this CELLSTYLE-AS statement use the variable name weight. If you want to use the value of the data column that is associated with the column that the CELLSTYLE-AS statement is being executed on, you can use the keyword `_VAL_`. This feature comes in handy when the CELLSTYLE-AS statement is in the table definition and applies to multiple columns. Figure 17 shows the output from the same CELLSTYLE-AS statement applied to the table rather than the column and with `_VAL_` used instead of weight.



Name	Sex	Age	Height	Weight
Alfred	M	14	69	112.5
Alice	F	13	56.5	84
Barbara	F	13	65.3	93
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83
Jane	F	12	59.8	84.5
Jared	F	15	62.5	112.5
Jeffrey	M	13	62.5	84

Figure 17. SAS Output in HTML Applying Color-Coding to Data in Multiple Columns

This output isn't very interesting since the columns Age and Height always fall into the `_VAL_ <= 90` category, but you get the idea.

One last tip about the CELLSTYLE-AS statement refers to situations where none of the expressions is true. If you want a case at the end that will always be applied when none of the other expressions evaluates to true, you can simply set the final expression to 1, which is true by definition.

```
cellstyle weight >= 110 as { ... },
           weight <= 90 as { ... },
           1           as { ... };
```

MODIFYING SAS PROCEDURE TABLES

Modifying the table template associated with a SAS procedure is no more difficult than writing your own templates. When a procedure prints a table, it hands the data to ODS and tells ODS which table template to use to render it. Therefore, in order to get ODS to use a different version of a table template than the procedure is asking for, you just need to make sure that ODS finds your table template before it finds the SAS-supplied table template. This is actually easier than it sounds. First, figure out what the name of the table template is. To do this, submit the following code:

```
ods trace on;
```

From now on, whenever ODS prints a table, it will also print information to the log that tells you about that table. Each table will print out something like the following:

```
Output Added:
-----
Name:      Variables
Label:     Variables
Template:   Base.Contents.Variables
Path:      Contents.DataSet.Variables
-----
```

This output was generated by the third table when running PROC CONTENTS on Sashelp.Class. As you can see, the template for this table is named Base.Contents.Variables. Coincidentally, this is the table template that we first looked at in this paper.

There are a couple of different ways that we can go about modifying this template. One way is to get the source of the template using the Template Browser as described at the beginning of this paper, edit it, and submit it. Another way is to use the PROC TEMPLATE EDIT statement. In either case, the new table template is written to the first writable template store in the ODS path.¹² In most cases, this is Sasuser.Templat.

The EDIT statement is just like the DEFINE statement except that it makes a copy of the definition being edited and modifies only the things that you change within the scope of the EDIT. Here is a simple modification to the Base.Contents.Variables table that removes the column that contains the variable number:

```
edit Base.Contents.Variables;
    column Variable Type Len Pos Flags Format Informat Label Transcode;
end;
```

If you run PROC CONTENTS on the data set again, you'll see that the first column of the Variables table is now gone. To get it back, just edit the table back to the way it was originally, or delete the template using the following code:

```
proc template;
    delete Base.Contents.Variables;
run;
```

¹² See the SAS 9.1.3 *Output Delivery System: User's Guide, Volumes 1 and 2* (SAS Institute Inc. 2006) for more information on the ODS PATH statement.

INHERITANCE

Inheritance is probably the most complex feature of table templates. It lets you re-use pieces of templates in other templates. When we introduced column templates earlier in this paper, we talked about setting formats on columns and came up with a template like this:

```
define table mytable;
  column name sex (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;
```

The point of using the FORMAT statements was to make sure that all of those columns used the same format. Although this simple example did work, it is error-prone. What if we wanted to change the format to something else? We would have to locate all of the occurrences of the FORMAT= attribute and make sure they were all correct. We can use inheritance to remedy this by putting the format information in one template and then inheriting from that template, as follows:

```
define column my.formatcolumn;
  format = 5.1;
end;
define table mytable;
  column name sex (age height weight);
  define age;
    define header ageheightweight;
      text 'Age*Height*Weight';
      split = '*';
    end;
    header = ageheightweight;
    parent = my.formatcolumn;
  end;
  define height;
    parent = my.formatcolumn;
  end;
  define weight;
    parent = my.formatcolumn;
  end;
end;
```

As you can see in this code example, we defined a new column called My.FormatColumn, which now contains the format that we want applied to the Age, Height, and Weight columns. Then, the Age, Height, and Weight columns inherit from this column using the PARENT= attribute. The value of the PARENT= attribute is the name of the template that you want to inherit attributes from. The PARENT= attribute exists for tables, headers, footers, and columns. A template can only inherit from the same template type.

The example we show here is a very simple example of what inheritance can do. The SAS-supplied templates use inheritance extensively, so if you plan on modifying procedure-produced tables a lot, you should read more about inheritance in the *SAS 9.1.3 Output Delivery System: User's Guide, Volumes 1 and 2* (SAS Institute Inc. 2006).

CONCLUSION

We hope that you enjoyed your foray into PROC TEMPLATE tables. You've now seen how to locate the SAS-supplied templates using the Template Browser. You have been introduced to defining table, header, footer, and column templates. And in the last section, you saw more advanced techniques such as trafficlighting, inheritance, and how to modify table templates used by SAS procedures. The only thing left is for you to go out and try these techniques on your own data. As Anton Chekhov, Russian playwright, once said,

"Knowledge is of no value unless you put it into practice."

RESOURCES

SAS Institute Inc. 2006. *SAS 9.1.3 Output Delivery System: User's Guide, Volumes 1 and 2*. Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/base_ods_9268.pdf.

CONTACT INFORMATION

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

Kevin D. Smith
SAS Institute Inc.
100 SAS Campus Drive
Cary, NC 27513
E-mail: Kevin.Smith@sas.com
Web: www.sas.com

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

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