The Power of TABLE Templates and DATA _NULL_

Cynthia L. Zender, SAS® Institute, Inc., Cary, NC

ABSTRACT

Are you a DATA step programmer? Do you want to route **DATA _NULL_** output to the Output Delivery System (ODS)? Do you want to convert classic **DATA _NULL_** and **FILE PRINT** programs to take advantage of the ODS? This paper explains how to create and use a custom TABLE template with a **DATA _NULL_** program. Through the use of concrete examples, you will learn how to become a power user of custom TABLE templates and **DATA _NULL_**. Topics covered include defining a new template, defining headers and footers, using **GENERIC** columns, and performing traffic-lighting based on data cell values.

INTRODUCTION

The Output Delivery System (ODS) was a radical innovation for SAS users that presented them with a new set of skills to master. With the ability to automatically generate production-quality Web pages (HTML destination), word processing documents (RTF destination), or print-ready documents (PDF destination) from SAS output, ODS users soon learned the following basic tenets of ODS processing:

1. The three main ODS destinations are RTF, PDF, and HTML. Other ODS destinations are PRINTER, PCL, TROFF, CSV, LaTeX, XML, XHTML, and more.
2. Basic ODS invocation is a "sandwich" technique, with opening and closing statements, similar to **PROC PRINTTO** in the old days.
3. If you open a destination, you must close the same destination (or use **ODS _ALL_ CLOSE;**).
4. To get the correct output, step boundaries belong inside the ODS invocation "sandwich".
5. Most SAS procedures create **output objects**, which represent data components from the procedure bound to a TABLE template that specifies information about formats, column order, and headings to be used when the **output object** is routed to an ODS destination.
6. **PROC REPORT**, **PROC TABULATE**, and **PROC PRINT** are procedures that do not have a TABLE template because each procedure uses syntax to internally control the look of the output table. These procedures support internal statement-level **STYLE** options to change the style of ODS output in destinations that support style.
7. Not all ODS options work in all ODS destinations (also known as the "it depends on the destination" rule).
8. There are four different kinds of templates involved in ODS processing— **TABLE**, **STYLE**, **TAGSET**, and **GRAPH**. Not all templates are "active" in all jobs.
9. For almost all procedures, ODS produces output in the form of tables that consist of rows and columns.
10. There is new syntax for working with ODS destinations within a **DATA _NULL_** program. ODS does not currently operate with **DATA _NULL_** and **FILE PRINT** the way most users expect.

A BRIEF REVIEW OF DATA STEP PROGRAMMING

To understand why the tenth tenet presents such a challenge to SAS programmers, it is important to review the capabilities of the DATA step program. DATA step program functionality falls into the following broad categories:

- data set creation, which includes data transformation, data manipulation, and infile/input processing to read raw data files into SAS format
- **DATA _NULL_** and **FILE PRINT** processing for report generation

At one time, **DATA _NULL_** processing with classic **PUT** statements for report generation was the mainstay of SAS report programmers. **DATA _NULL_** programs, with the power of the **PUT** statement, enabled programmers to create either tabular or free-format output from a DATA step program. With output directed to the special **FILE PRINT** fileref, programmers could produce report output with custom headers, footers, and page breaks. Programmers could also use the power of SAS **BY** group processing, array processing, SAS functions, and conditional logic within their programs. When output was sent to the **LISTING** window, the **OUTPUT** window, or the **SYSPRINT DD** device, this output was meant to be printed with a monospace font, such as SAS Monospace, Courier, or LinePrinter. But, what happens when you use the ODS invocation statements around a classic **DATA _NULL_** program?

Consider the differences between the following examples:

Classic DATA _NULL_ and PUT	DATA _NULL_ and PUT for ODS
file print; put @15 Type @30 Amt comma8.;	file print ods; put _ods_;

Example 1. Classic versus ODS **DATA _NULL_** and **PUT**

FREE-FORMAT OUTPUT VERSUS TABULAR OUTPUT

In the **PUT** statement in the "classic" example, the **TYPE** variable is being written in print position **15** of the output page, and the **AMT** variable is being written in print position **30** of the output page. However, when you look at the entire example, the report that is being produced contains a mixture of pure tabular output, mixed with free-format output.

In comparison, in the **PUT** statement in the ODS example, classic syntax elements are not being used. The absence of print positioning is your clue that ODS does not address print positions on the output page. When you look at the output from ODS, you can see that ODS is creating a regular, rectangular table with no free-format print positioning as is found in the classic example.

Classic DATA _NULL_ and PUT	DATA _NULL_ and PUT for ODS

data _null_;
set tktdata;
retain Tot;
by dest;
file print;
if first.dest then do;
Tot = 0;
link h;
end;
put @15 Type
@30 Amt comma8.;
Tot + Amt;
if last.dest then do;
put @30 '--------';
put @30 Tot dollar8. /;
put @5 'Sign Below' ///
@5 50*'_'///
@5 'Deposit Number: _____';
put _page_;
end;
return;
h:
put @5 54*'*';
put @5 '*'
@7 'Destination: ' Dest
@58 '*' /
@5 54*'*' /;
put @15 'Type of Sale'
@30 'Amount of Sale';
return;
run;
	data _null_;
	footnote
	'Sign: _____'
	'Deposit:_____';
	set tktdata end=LastObs;
	by dest;
	if first.dest then Tot = 0;
	Tot + Amt;
	GTot + Amt;
	file print ods=(variables=
	(Dest Type Amt));
	put _ods_;
	if last.dest then do;
	dest='Subtotal';
	put @1 dest @3 Tot;
	put ' ';
	end;
	if LastObs then do;
	dest='Grand Total';
	put @1 dest @3 GTot ;
	end;
	format Amt comma8.;
	run;

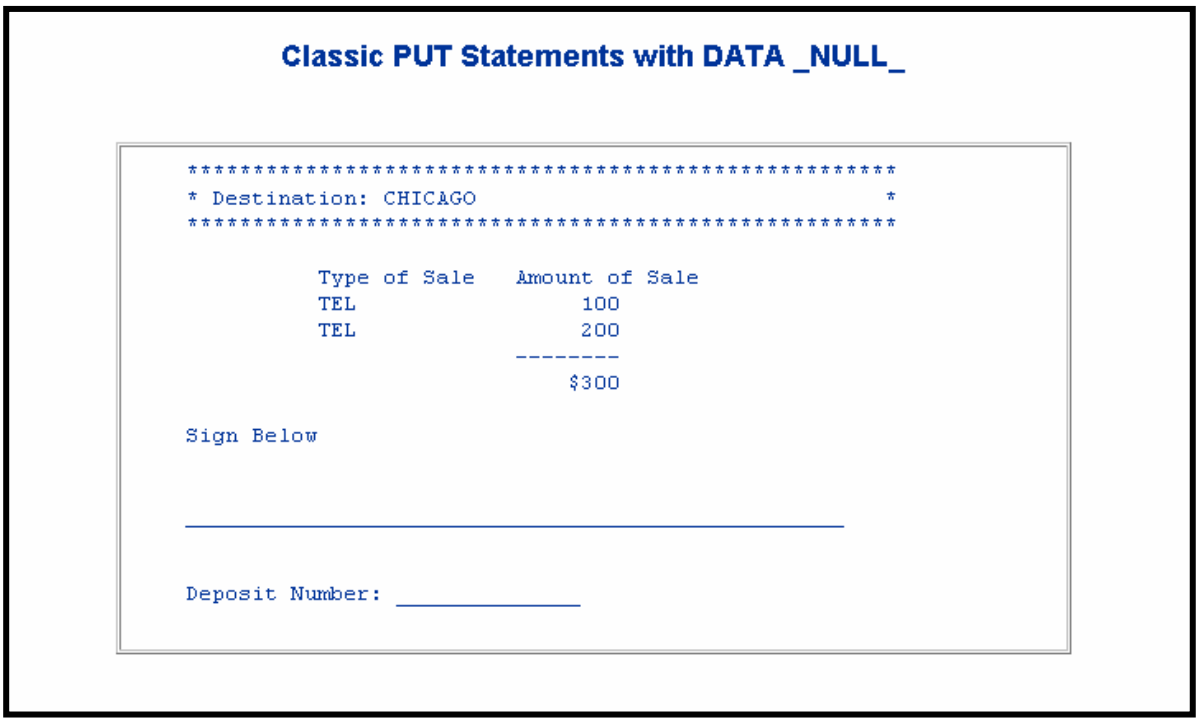
Example 2. Classic versus ODS **DATA _NULL_** and **PUT,** continued

The following is partial output from the “classic” example if routed to the **LISTING** destination:

```
                Classic PUT Statements with DATA _NULL_                    1

**********************************************************
* Destination: CHICAGO                                   *
**********************************************************

            Type of Sale    Amount of Sale
            TEL                      100
            TEL                      200
                            ------------
                                    $300

Sign Below


_____________________________________________________


Deposit Number: _________________
```

Example 3. **LISTING** Destination Output

The following is partial output from the “classic” example if it is routed to an HTML destination:



Example 4. HTML Output

As you can see from the above HTML output, the free-format output is centered inside a “batch” area in the output file. If you are satisfied with the way the HTML output looks, then all you might want to do is change the **STYLE** template to redefine the font.

However, absolute placement in print position **15** or print position **30** is difficult to guarantee when you route free-format output to non-**LISTING** destinations such as HTML, RTF, or PDF. **LISTING** destination output is produced using a monospace font. Therefore, it is possible to treat the entire print page as a grid into which you can place characters in absolute print positions. However, all other ODS output is produced using a proportional font. With a proportional font, it is impossible to predict how many characters will fit on a single line, because each letter takes a different amount of space.

Consider the differences between the following two lines, shown in a 10 pt Arial font:
iljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljiljilj
kmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmwkmw

The first line, consisting of "skinny" letters **i**, **l**, and **j**, contains 200 characters. The second line, consisting of "wide" letters **k**, **m**, and **w**, contains only 60 characters.

FILE PRINT ODS AND PUT _ODS_ STATEMENTS

When you use the **DATA _NULL_** syntax designed for ODS, you can produce tabular output in the default proportional font for the destination (which is Arial for HTML and Times Roman for RTF and PDF). This means that free-format output, such as what you produce in the **LISTING** destination, is not possible to produce at this time for RTF, PDF, and HTML destinations using **DATA _NULL_** syntax.

ODS developers are working on an interface between **DATA _NULL_** and ODS to enable the creation of free-format output. For more information about this capability, refer to *Next Generation Data _NULL_ Report Writing Using ODS OO Features* by Daniel O'Connor. You might also want to read *Using New Features in ODS to Create Master/Detail Reports* by Jack Hamilton.

In Example 2, the special syntax is evident in the ODS example. First, the fileref **FILE PRINT ODS** is used to pass variable information from the Program Data Vector (PDV) to an output table. The resulting table will have one table column for every non-temporary variable in the PDV by default, or one table column for every variable in the **variables=** sub-option. The variables that are passed are specified in the **variables=** sub-option of the **FILE PRINT ODS** statement. When the statement **PUT _ODS_** is executed, the value of each variable is written to the corresponding table column in the output table. In the example, **DEST** is column 1 in the table; **TYPE** is column 2 in the table, and **AMT** is column 3 in the table. SAS automatically writes the variable labels or names as column headers before the first observation in the output table. Then, every observation in the data set is placed in one row of the output table, unless program logic inserts other information, such as the subtotals in the example.

It is possible to use column-pointer control with the **PUT** statement. However, when a statement, such as:

```
put @1 dest @3 Tot;
```

is executed, the variable values are placed in column 1 and column 3 of the output table, not in print position 1 and print position 3 (as is the case with the **LISTING** destination output).

New ODS Syntax

```
data _null_;
  footnote
    'Sign: _____________________'
    'Deposit:__________________';
  set tktdata end=LastObs;
  by dest;
  if first.dest then Tot = 0;
  Tot + Amt;
  GTot + Amt;
  file print ods=(variables=
                  (Dest Type Amt));
  put _ods_;

  if last.dest then do;
     dest='Subtotal';
     put @1 dest @3 Tot;
     put ' ';
  end;
  if LastObs then do;
     dest='Grand Total';
     put @1 dest @3 GTot ;
  end;
  format Amt comma8.;
run;
```

Default Table Template for DATA _NULL_

Dest	Type	Amt
CHICAGO	TEL	100
CHICAGO	TEL	200
Subtotal		300
GENEVA	WEB	300
GENEVA	WEB	400
Subtotal		700
LONDON	TEL	500
LONDON	TEL	600
LONDON	WEB	700
Subtotal		1,800

Example 5. Syntax and Partial Output

The DATA step program uses the table template **BASE.DATASTEP.TABLE**, which defines two **GENERIC** columns, **_numvar_** and **_charvar_**. **DEST** and **TYPE** use the **_charvar_** generic definition and **AMT** uses the **_numvar_** generic definition. The full text of this template is shown in *Appendix A: Programs and Templates*.

When you use **PROC TEMPLATE** syntax, it is possible to define a custom table template for use with a **DATA _NULL_** program. **PROC TEMPLATE** syntax for table templates looks like a variation of **PROC REPORT** syntax. After the initial **PROC TEMPLATE** statement and the **DEFINE** statement for the table, there is a **COLUMN** statement that defines the order of the table columns, and then a **DEFINE/END** block that contains column attribute definitions.

To define a custom TABLE template for the **TKTDATA** data set in the examples, you have to define a table column for each variable (**DEST**, **TYPE**, and **AMT**) in a **COLUMN** statement. Then, for every item in the **COLUMN** statement, a **DEFINE/END** block is needed to define the column attributes. For example, the **HEADER** statement specifies the **COLUMN** header. In the default TABLE template for the DATA step, the header is defined as the **_LABEL_** for the variable. In the custom TABLE template, a variable-specific header is used for every column.

In addition to the header, justification and format information can be supplied for the column in a custom TABLE template. Other frequently used column attributes are:

FORMAT=	sets the format for the column
PRINT_HEADERS=ON	specifies whether or not to print headers
BLANK_DUPS=ON	inserts blanks for repetitious or duplicate column values
JUST=	specifies column justification
HEADER=	specifies the text for the column header (can also define a header)
GENERIC=ON	specifies whether the column definition can be reused for other variables
STYLE=	specifies style element and/or style attributes to use for the column
CELLSTYLE…AS	conditionally specifies style attributes for a particular column or an entire table

Consult ODS documentation for a comprehensive list of all the possible column attributes.

Following is custom table template for the **TKTDATA** data set in the previous examples:

Custom TABLE Template

```
proc template;
  define table tables.tktdata;
  column dest type amt;
  define dest;
    header='Destination';
    just=center;
    blank_dups=on;
  end;
  define type;
    header='Ticket Agent';
    just=left;
  end;
  define amt;
    header = 'Sales Amount';
    just=right;
    format=dollar10.0;
  end;
  end;
run;

options missing = ' ';
ods pdf file='tkt_temp1.pdf';
ods rtf file='tkt_temp1.rtf';
ods html file='tkt_temp1.html'
style=sasweb;
title
 'Custom Table Template for DATA
_NULL_';
data _null_;
  set tktdata;
  by dest type;
  file print ods=
     (template='tables.tktdata');
  put _ods_;
  if first.dest and first.type
     then totsales=0;
  totsales+amt;
  if last.dest and last.type
    then do;
    dest = 'Total Sales';
    type = ' ';
    amt = totsales;
    put _ods_;
    put ' ';
  end;
run;
ods _all_ close;
```

Partial Output in HTML

Custom Table Template for DATA _NULL_

Destination	Ticket Agent	Sales Amount
CHICAGO	TEL	$100
	TEL	$200
Total Sales		$300
GENEVA	WEB	$300
	WEB	$400
Total Sales		$700
LONDON	TEL	$500
	TEL	$600
	WEB	$700
Total Sales		$1,800
PARIS	TEL	$800
	WEB	$900
Total Sales		$1,700

Example 6. Custom TABLE Template and Output

ENHANCING THE CUSTOM TABLE TEMPLATE

Through the use of the **STYLE=** statement and the **CELLSTYLE-AS** statement, you can enhance how output looks. Style elements and attributes that are specified in the table template override style information in the **STYLE** template. For example, the following are different statements to specify style information:

Statement	Changes
style=header;	column uses **HEADER** element style attributes
style=data;	column uses **DATA** element style attributes
style={background=purple foreground=white};	statement overrides background and foreground style attributes
style={font_weight=bold};	statement overrides the **font_weight** style attribute

In addition to the **STYLE=** statement, you can use the **CELLSTYLE-AS** statement to change the appearance of the output. In addition, the use of the **MVAR** statement enables you to use macro variables to make the template code more flexible.

Enhancements to style elements and attributes can be made in a TABLE template. In the following example, the desired changes are:

- Define a header that spans the whole table and use a macro variable in the header text.
- Change the style of the **DEST** column so that it has the same style attributes as the column headers.
- Change the style of the **Total Sales** row.
- Use traffic-lighting techniques to change the background of the **AMT** column.
- Define a footer that spans the whole table and use a macro variable in the footer text.

Enhancements to Style Elements and Attributes	Highlighted Code Achieves Desired Changes

`ods path work.temptemp(update)`
`  sasuser.templat(update)`
`  sashelp.tmplmst(read);`
`proc template;`
`define table tables.tktstyle;`
`  column dest type amt;`	

`  mvar wkday grandtot;`
`  header tabhdr;`
`  define tabhdr;`
`  start=Dest;`
`  end = Amt;`
	Declare the macro variables to be used.
	Define a header that spans the whole table.

`  text 'One Hour of Sales on: ' wkday;` `  end;`	Use the **wkday** macro variable.

`  footer tabftr;`
`  define tabftr;`
`  text 'All Locations: ' grandtot;`
`  end;`
	Define a footer that spans the whole table and use the **grandtot** macro variable.

`  define dest;`
`  header='Destination';`
`  just=center;`
`  blank_dups=on;`	

`  style=header;` `  end;`	Change the style of the **DEST** column so that it has the same attributes as the column headers.

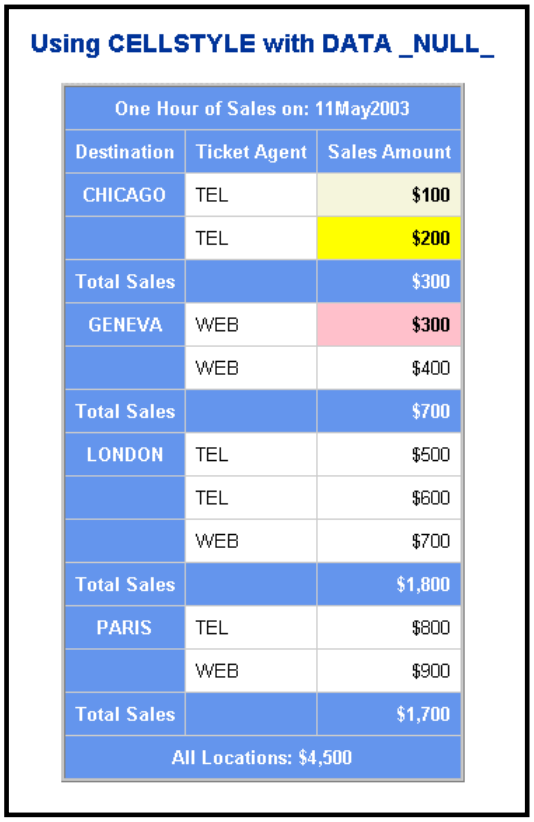
`  define type;`
`  header='Ticket Agent';`
`  just=left;`	

`  cellstyle`
`  dest = 'Total Sales' as header,`
`  1 as data;`
`  end;`
	Change the style of the **Total Sales** row.

`  define amt;`
`  header = 'Sales Amount';`
`  just=right;`
`  format=dollar10.0;`	

`  cellstyle`
`  dest = 'Total Sales' as header,`
`  _val_ eq 100 as`
`{background=beige font_weight=bold},`
	Change the style of the **Total Sales** row. Use traffic-lighting techniques to change the background of the **AMT** column.

```
      _val_ eq 200 as {background=yellow
font_weight=bold},
      _val_ eq 300 as
{background=pink font_weight=bold},
      1 as data;
  end;
end;
run;


** set macro var for date;
%let wkday = 11May2003;

** capture grand total in macro variable;
proc sql noprint;
select put(sum(amt),dollar8.) into
:grandtot
from work.tktdata;
quit;

** strip leading and trailing blanks;
%let grandtot = &grandtot;

ods html file='tkt_cellstyle.html'
         style=sasweb;
title 'Using CELLSTYLE with DATA _NULL_';
data _null_;
  set tktdata ;
  by dest type;
  file print
ods=(template='tables.tktstyle');
  put _ods_;
  grandtot + amt;
  if first.dest and first.type
     then totsales=0;
  totsales+amt;
  if last.dest and last.type then do;
    dest = 'Total Sales';
    type = ' ';
    amt = totsales;
    put _ods_;
  end;
run;
ods html close;
```



Using CELLSTYLE with DATA _NULL_

One Hour of Sales on: 11May2003		
Destination	Ticket Agent	Sales Amount
CHICAGO	TEL	$100
	TEL	$200
Total Sales		$300
GENEVA	WEB	$300
	WEB	$400
Total Sales		$700
LONDON	TEL	$500
	TEL	$600
	WEB	$700
Total Sales		$1,800
PARIS	TEL	$800
	WEB	$900
Total Sales		$1,700
All Locations: $4,500		

Example 7. Output from Custom TABLE Template

The production of the table header and table footer depends on the use of the **MVAR** statement, which enables you to specify macro variables in the custom TABLE template. As long as the macro variables exist when the **DATA _NULL_** program executes, the resolved variable values are available for use by the template. In Example 6, note the use of the **%LET** statement to set the value of **&wkday** and the use of **PROC SQL** to set the value of **&grandtot**. The **MVAR** statement enables you to reference the macro variable in the TABLE template without the use of an ampersand in front of the macro variable name. If you use an ampersand in the TABLE template, the macro variable reference is resolved when the template is compiled. The absence of an ampersand for the macro variable, coupled with the **MVAR** statement, delays resolution of the macro variable until the template is executed, not when the template is compiled. This is desirable because you may want to use a TABLE template multiple times and send different values for the macro variables each time, which can only happen if resolution of the macro variables is delayed until template execution time. Inside the **DEFINE/END** block for the **DEST** column, the statement **style=header;** tells the template that the **HEADER** style element from the **STYLE** template should be used for the entire column.

The **CELLSTYLE-AS** statement for **TYPE** satisfies two objectives:

- If the value of the **DEST** column is **Total Sales**, then the **TYPE** column inherits **HEADER** style attributes.
- Otherwise, the **TYPE** column inherits **DATA** style attributes.

```
define type;
  header='Ticket Agent';
  just=left;
  cellstyle dest = 'Total Sales' as header,
                1 as data;
end;
```

The **AMT** column has two objectives satisfied by the **CELLSTYLE-AS** statement:

- If the value of the **DEST** column is **Total Sales**, then the **AMT** column inherits **HEADER** style attributes. The expression that is evaluated in the **CELLSTYLE-AS** statement can be any valid **WHERE** statement. The special column reference **_VAL_** refers to the value in the current row for the column that is being defined. You are not limited to only testing the current column value. As seen in the following example, you can base one column's style characteristics on the values of another column in the template.

```
define amt;
  header = 'Sales Amount';
  just=right;
  format=dollar10.0;
  cellstyle dest = 'Total Sales' as header,
     _val_ eq 100 as {background=beige font_weight=bold},
     _val_ eq 200 as {background=yellow font_weight=bold},
     _val_ eq 300 as {background=pink font_weight=bold},
       1 as data;
end;
```

- If the current cell value is **100**, the background color is changed to **beige**; if the current cell value is **200**, the background color is changed to **yellow**; if the current cell value is **300**, the background color is changed to **pink**. All three cell values have a **FONT_WEIGHT** of **bold** and a background color change. Because the test for **DEST='Total Sales'** is performed first, the style attributes for the **Total Sales** row are set first. In Example 7, even though the **Chicago** total is **$300**, the cell uses **HEADER** attributes, rather than the background color change of pink, because the conditions in the **CELLSTYLE-AS** statement are executed in the order that they are encountered. Once the current cell value meets a condition in the **CELLSTYLE-AS** statement, no further conditions are tested.

MAKING THE CUSTOM TABLE TEMPLATE MORE FLEXIBLE

Further modifications can make the TABLE template more flexible. By changing the **DEFINE/END** block for the **AMT** column to use macro variables that are specified in the **NMVAR** statement, low, medium, and high values for the conditions can be specified. The only changes needed to the previous code are the following:

```
nmvar botamt midamt topamt;
define amt;
  header = 'Sales Amount';
  just=right;
  format=dollar10.0;
  cellstyle dest = 'Total Sales' as header,
          _val_ eq botamt as {background=#ccccff font_weight=bold},
          _val_ eq midamt as {background=#ccffcc font_weight=bold},
          _val_ eq topamt as {background=#99ccff font_weight=bold},
             1 as data;
end;
```

Rather than having hard-coded values in the TABLE template, the values for comparison are set before the TABLE template is executed. Values can be set with **%LET**, **CALL SYMPUT**, or **PROC SQL**. Values have to be specified in an **NMVAR** statement because **_VAL_** for the **AMT c**olumn is a numeric variable. When the values are retrieved from the macro symbol table, they need to be treated as numbers by the TABLE template. In the example, the **DATA _NULL_** program is run once. The colors shown in the **CELLSTYLE-AS** statement are specified as RGB hexadecimal values— **#ccccff** is lavender, **#ccffcc** is mint green, and **#99ccff** is light blue. Even though the colors stay the same, when the **DATA _NULL_** program is run twice, different cells are highlighted with these colors. Before the first run of the program, you specify one set of values for these macro variables; before the second run of the program, you specify a different set of values for these macro variables.

First Run of the DATA _NULL_ Program	Second Run of the DATA _NULL_ Program

`%let botamt = 100;`
`%let midamt = 200;`
`%let topamt = 300;`
	`%let botamt = 500;`
	`%let midamt = 600;`
	`%let topamt = 700;`

One Hour of Sales on: 11May2003

Destination	Ticket Agent	Sales Amount
CHICAGO	TEL	$100
	TEL	$200
Total Sales		$300
GENEVA	WEB	$300

LONDON	TEL	$500
	TEL	$600
	WEB	$700
Total Sales		$1,800

Example 8. Program Runs and Resulting Output

In the previous example, the text **CHICAGO** has a black foreground, the text **GENEVA** has a blue foreground, and the text **LONDON** has a yellow foreground. These changes were accomplished by traffic-lighting, but with a user-defined format for the **FOREGROUND** attribute instead of a **CELLSTYLE-AS** statement. The full text of this program—**DATA_NULL_DEMO4.SAS**—is shown in *Appendix A: Programs and Templates*.

After the format **$destfmt** has been defined, the style attribute for the **DEST** column requires the following highlighted change:

```
define dest;
  define header desthdr;
    text 'Destination';
    just=left;
    style=header{font_size=12pt};
  end;
  header=desthdr;
  just=center;
  blank_dups=on;
  style=header{foreground=$destfmt.};
end;
```

In the previous code, notice the **HEADER** definition block for **DESTHDR** inside the **DEFINE/END** block for the **DEST** column, which is another way that you can define a header for a particular column. This way enables you to specify text and style attributes for the header that are separate from justification or other style characteristics of the column. Once the header has been defined, then you only need to point to it in the **HEADER=** statement for the template to use the newly defined header.

USING GENERIC COLUMNS IN A TABLE TEMPLATE

In the default TABLE template used for **DATA _NULL_** programs, only two **GENERIC** columns are defined in the template, **_numvar_** and **_charvar_**. Another way to make a template more flexible is to create a custom TABLE template that uses **GENERIC** columns. For example, the following data uses a different structure than the previous data:

```
data moretkt;
length Dest $16 Type1-Type3 tot $45 Amt1-Amt3 8;
input DEST $ TYPE1 $ AMT1 TYPE2 $ AMT2 TYPE3 $ AMT3 TOT $ TOTAMT;
return;
cards;
CHICAGO    TEL  100 TEL  200  WEB  300 TOT  600
GENEVA     TEL  400 WEB  500  WEB  600 TOT 1500
LONDON     TEL  700 TEL  800  WEB  900 TOT 2400
PARIS      TEL 1000 WEB 1100  WEB 1200 TOT 3300
;
run;
```

In the data, there is only one row for every destination and multiple **TYPE** variables and **AMT** variables for every row. Using **GENERIC** columns, you can create two different reports. Generic Example 1 shows the **TYPE** variables under the **DEST** column, and Generic Example 2 shows the **TYPE** and **AMT** columns repeated on each row.

Generic Example 1

One Hour of Sales on: 12May2003	
Destination and Type	Sales Amount
CHICAGO	
-----TEL	$100
-----TEL	$200
-----WEB	$300
-----TOT	$600
GENEVA	
-----TEL	$400
-----WEB	$500
-----WEB	$600
-----TOT	$1,500

Generic Example 2

Destination	Type 1	Amt 1	Type 2	Amt 2	Type 3	Amt 3	Total Amount
CHICAGO	TEL	$100	TEL	$200	WEB	$300	$600
GENEVA	TEL	$400	WEB	$500	WEB	$600	$1,500
LONDON	TEL	$700	TEL	$800	WEB	$900	$2,400
PARIS	TEL	$1,000	WEB	$1,100	WEB	$1,200	$3,300
						All Locations:	$7,800

Example 9. Output in Two Different Reports

In the TABLE template, the key to success for either example is the use of **GENERIC** columns. The default TABLE template uses two generic columns, **_numvar_** and **_charvar_**, and the TABLE templates that produced the output in Example 9 use **GENERIC** columns. Consider the difference in the two **COLUMN** statements in the **PROC TEMPLATE** definition in the following template examples:

Generic Template 1

```
proc template;
  define table tables.genex1;
    mvar wkday grandtot;
    column (dest type) (amt);
    header tabhdr;
    define tabhdr;
      start=Dest;
      end = Amt;
      text 'One Hour of Sales on: '
wkday;
    end;
    footer tabftr;
    define tabftr;
      text 'All Locations: ' grandtot;
    end;
    define dest;
```

Generic Template 2

```
proc template;
  define table tables.genex2;
    mvar grandtot;
    column dest type amt;
    footer tabftr;
    define tabftr;
      text 'All Locations: ' grandtot;
      just=right;
    end;
    define dest;
      define header desthdr;
         text 'Destination';
         just=left;
         style=header;
       end;
       header=desthdr;
```

```
      define header desthdr;
         text 'Destination and Type';
         just=left;
         style=header;
       end;
       header=desthdr;
       just=left;
       generic=on;
       style=header{cellwidth=.75in};
    end;
    define type;
      just=left;
      generic=on;
      style=header{cellwidth=.75in};
    end;
    define amt;
      generic=on;
      header = 'Sales Amount';
      format=dollar10.0;
      style=data{just=right};
    end;
  end;
run;
```

```
       just=left;
       style=header{cellwidth=.75in};
    end;
    define type;
      header=_label_;
      just=left;
      generic=on;
      style=data{cellwidth=.75in};
    end;
    define amt;
      generic=on;
      header = _label_;
      format=dollar10.0;
      style=data{just=right};
    end;
    order_data;
  end;
run;
```

In Generic Template 1, the parentheses in the **COLUMN** statement indicate that data values are stacked within one table cell.

In Generic Template 2, the **COLUMN** statement specifies two generic columns, which are reused for the **TYPE** and **AMT** variables in the data set. In addition, the **AMT** column is reused for the **TOTAMT** variable from the data set.

In Generic Template 1, an alternate method of specifying a header for a particular column is used. Notice how the **DEFINE/END** block for **DESTHDR** is contained within the **DEFINE/END** block for the **DEST** column, and how style attributes for the **DESTHDR** header are specified inside the separate **DEFINE/END** block.

When these two different templates are used, the method of invoking them from within the **DATA _NULL_** program is similar. What makes the output look different are the differences in the two TABLE templates.

Generic DATA _NULL_ 1

```
data _null_;
  set moretkt ;
  blnkamt = .;
  indstr = '-----';
  type1 = trim(indstr)||trim(type1);
  type2 = trim(indstr)||trim(type2);
  type3 = trim(indstr)||trim(type3);
  tot = trim(indstr)||trim(tot);
  file print
ods=(template='tables.genex1'
    columns=(Dest=dest(generic=on)
             amt=blnkamt(generic=on)
             type=type1(generic=on)
             amt=amt1(generic=on)
             type=type2(generic=on)
             amt=amt2(generic=on)
             type=type3(generic=on)
             amt=amt3(generic=on)
             type=tot(generic=on)
             amt=totamt(generic=on)
            ));
  put _ods_;
run;
```

Generic DATA _NULL_ 2

```
data _null_;
  set moretkt ;
  file print ods=(template='tables.genex2'
    columns=(Dest
             type=type1(generic=on)
             amt=amt1(generic=on)
             type=type2(generic=on)
             amt=amt2(generic=on)
             type=type3(generic=on)
             amt=amt3(generic=on)
             amt=totamt(generic=on)
            ));
  put _ods_;
  label type1 = "Type 1"
        amt1 = "Amt 1"
        type2 = "Type 2"
        amt2 = "Amt 2"
        type3 = "Type 3"
        amt3 = "Amt 3"
        totamt = "Total Amount";
run;
```

Output from **Generic DATA _NULL_ 1** is stacked within one table cell because of the parentheses in the **COLUMN** statement. A blank amount needs to be associated with the **DEST** column so that the other numbers line up correctly.

No stacking occurs in **Generic DATA _NULL_ 2**. The **TYPE** and **AMT** columns are repeated as many times as necessary. In **Generic DATA _NULL_ 2**, **TOTAMT** is passed to the template without the corresponding **TOT** column. With one template, **GENERIC** columns stack data set information vertically. With the other template, columns are repeated horizontally. The full text of **DATA_NULL_DEMO5.SAS** is shown in *Appendix A: Programs and Templates*.

CONCLUSION

If you need to generate tabular output and route the output to ODS destinations, the TABLE template syntax, coupled with **DATA _NULL_** programming, provides you with the power to generate your own custom tables. You can use macro variables in the template or change a cell's style conditionally, either with the **CELLSTYLE-AS** statement or the **STYLE=** statement. In addition, through user-defined formats, the TABLE template, and other features of TABLE template syntax (such as the ability to compute columns), the SAS programmer has many ways to make use of his/her **DATA _NULL_** programming skills in the world of ODS.

APPENDIX A: PROGRAMS AND TEMPLATES

BASE.DATASTEP.TABLE TEMPLATE

```
proc template;
   define table Base.Datastep.Table;
      notes "Default DATA Step Table Definition";
      column _numvar_ _charvar_;
      define _numvar_;
         define header n_hdr;
            text _label_;
            just = c;
         end;
         header = n_hdr;
         generic;
      end;
      define _charvar_;
         define header c_hdr;
            text _label_;
            just = c;
         end;
         header = c_hdr;
         generic;
      end;
      justify;
      order_data;
   end;
run;
```

DATA_NULL_DEMO4.SAS PROGRAM

```
options nodate number pageno=1 ls=70;
ods path work.temptemp(update)
         sasuser.templat(update)
         sashelp.tmplmst(read);

** make format for foreground color of dest;
proc format;
  value $destfmt 'CHICAGO' = 'black'
                 'LONDON' = 'yellow'
                 'PARIS' = 'green'
                 'GENEVA' = 'blue'
                 'Total Sales' = 'white';
run;
```

```
proc template;
  define table tables.tktfmt;
  column dest type amt;
  mvar wkday grandtot;
  nmvar botamt midamt topamt;
  header tabhdr;
  define tabhdr;
    start=Dest;
    end = Amt;
    text 'One Hour of Sales on: ' wkday;
  end;
  footer tabftr;
  define tabftr;
    text 'All Locations: ' grandtot;
  end;
  define dest;
    define header desthdr;
      text 'Destination';
      just=left;
      style=header{font_size=12pt};
    end;
    header=desthdr;
    just=center;
    blank_dups=on;
    style=header{foreground=$destfmt.};
  end;
  define type;
    header='Ticket Agent';
    just=left;
    cellstyle dest = 'Total Sales' as header,
                1 as data;
  end;
  define amt;
    header = 'Sales Amount';
    just=right;
    format=dollar10.0;
    cellstyle dest = 'Total Sales' as header,
            _val_ eq botamt as {background=#ccccff font_weight=bold},
            _val_ eq midamt as {background=#ccffcc font_weight=bold},
            _val_ eq topamt as {background=#99ccff font_weight=bold},
               1 as data;
  end;
  end;
run;

**FIRST PROGRAM INVOCATION;
** set macro var for date and amount values;
%let wkday = 11May2003;
%let botamt = 100;
%let midamt = 200;
%let topamt = 300;

** capture grand total in macro variable;
proc sql noprint;
select put(sum(amt),dollar8.) into :grandtot
from work.tktdata;
quit;

** strip leading and trailing blanks from macro var and show value in log;
%let grandtot = &grandtot;
```

```
%put grandtot is: &grandtot;

ods html file='tkt_macro1.html' style=sasweb;
title 'Using CELLSTYLE with DATA _NULL_';
data _null_;
  set tktdata ;
  by dest type;
  file print ods=(template='tables.tktfmt');
  put _ods_;
  grandtot + amt;
  if first.dest and first.type then totsales=0;
  totsales+amt;
  if last.dest and last.type then do;
    dest = 'Total Sales';
    type = ' ';
    amt = totsales;
    put _ods_;
  end;
run;
ods html close;
title;

*** Second Invocation with different values for the macro variables;
%let botamt = 500;
%let midamt = 600;
%let topamt = 700;

ods html file='tkt_macro2.html' style=sasweb;
title 'Changing Macro Values';
data _null_;
  set tktdata ;
  by dest type;
  file print ods=(template='tables.tktfmt');
  put _ods_;
  grandtot + amt;
  if first.dest and first.type then totsales=0;
  totsales+amt;
  if last.dest and last.type then do;
    dest = 'Total Sales';
    type = ' ';
    amt = totsales;
    put _ods_;
  end;
run;
ods html close;
```

DATA_NULL_DEMO5.SAS PROGRAM - GENERIC COLUMN EXAMPLE 1

```
options nodate number pageno=1 ls=70;
ods listing close;

data moretkt;
length Dest $16 Type1-Type3 tot $45 Amt1-Amt3 8;
input DEST $ TYPE1 $ AMT1 TYPE2 $ AMT2
            TYPE3 $ AMT3 TOT $ TOTAMT;
return;
cards;
CHICAGO   TEL  100 TEL  200  WEB  300 TOT  600
GENEVA    TEL  400 WEB  500  WEB  600 TOT 1500
LONDON    TEL  700 TEL  800  WEB  900 TOT 2400
PARIS     TEL 1000 WEB 1100  WEB 1200 TOT 3300
;
run;
```

```
ods path work.temptemp(update)
         sasuser.templat(update)
         sashelp.tmplmst(read);

proc template;
  define table tables.genex1;
    mvar wkday grandtot;
    column (dest type) (amt);
    header tabhdr;
    define tabhdr;
      start=Dest;
      end = Amt;
      text 'One Hour of Sales on: ' wkday;
    end;
    footer tabftr;
    define tabftr;
      text 'All Locations: ' grandtot;
    end;
    define dest;
      define header desthdr;
         text 'Destination and Type';
         just=left;
         style=header;
       end;
       header=desthdr;
       just=left;
       generic=on;
       style=header{cellwidth=.75in};
    end;
    define type;
      just=left;
      generic=on;
      style=header{cellwidth=.75in};
    end;
    define amt;
      generic=on;
      header = 'Sales Amount';
      format=dollar10.0;
      style=data{just=right};
    end;
  end;
run;

** set macro var for date and amount values;
%let wkday = 12May2003;

** capture grand total in macro variable;
proc sql noprint;
select put(sum(totamt),dollar8.) into :grandtot from work.moretkt;
quit;

** strip leading and trailing blanks from macro var;
%let grandtot = &grandtot;

options missing = ' ';
ods html file='genex1.html' style=sasweb;
title '#1: Using generic columns with DATA _NULL_';
data _null_;
  set moretkt ;
  blnkamt = .;
  indstr = '-----';
  type1 = trim(indstr)||trim(type1);
  type2 = trim(indstr)||trim(type2);
  type3 = trim(indstr)||trim(type3);
  tot = trim(indstr)||trim(tot);
  file print ods=(template='tables.genex1'
```

```
                    columns=(Dest=dest(generic=on)
                           amt=blnkamt(generic=on)
                           type=type1(generic=on)
                           amt=amt1(generic=on)
                           type=type2(generic=on)
                           amt=amt2(generic=on)
                           type=type3(generic=on)
                           amt=amt3(generic=on)
                           type=tot(generic=on)
                           amt=totamt(generic=on)
                     ));
  put _ods_;
run;
ods html close;
```

GENERIC COLUMN EXAMPLE 2

```
proc template;
  define table tables.genex2;
    mvar grandtot;
    column dest type amt;
    footer tabftr;
    define tabftr;
      text 'All Locations: ' grandtot;
      just=right;
    end;
    define dest;
      define header desthdr;
         text 'Destination';
         just=left;
         style=header;
       end;
       header=desthdr;
       just=left;
       style=header{cellwidth=.75in};
    end;
    define type;
      header=_label_;
      just=left;
      generic=on;
      style=data{cellwidth=.75in};
    end;
    define amt;
      generic=on;
      header = _label_;
      format=dollar10.0;
      style=data{just=right};
    end;
    order_data;
  end;
run;

** set macro var for date and amount values;
%let wkday = 12May2003;

** capture grand total in macro variable;
proc sql noprint;
select put(sum(totamt),dollar8.) into :grandtot from work.moretkt;
quit;

** strip leading and trailing blanks from macro var;
%let grandtot = &grandtot;

options missing = ' ';
ods html file='genex2.html' style=sasweb;
title '#2: Using generic columns with DATA _NULL_';
data _null_;
```

```
  set moretkt ;
  file print ods=(template='tables.genex2'
                  columns=(Dest
                         type=type1(generic=on)
                         amt=amt1(generic=on)
                         type=type2(generic=on)
                         amt=amt2(generic=on)
                         type=type3(generic=on)
                         amt=amt3(generic=on)
                         amt=totamt(generic=on)
                   ));
  put _ods_;
  label type1 = "Type 1"
        amt1 = "Amt 1"    type2 = "Type 2"
        amt2 = "Amt 2"    type3 = "Type 3"
        amt3 = "Amt 3"    totamt = "Total Amount";
run;
ods html close;
```

REFERENCES

Hamilton, Jack. 2004. “Using New Features in ODS to Create Master/Detail Reports.” *Proceedings of the Twenty-ninth Annual SAS Users Group International Conference.* Cary, NC: SAS Institute Inc.

O’Connor, Daniel. 2003. “Next Generation Data _NULL_ Report Writing Using ODS OO Features”. *Proceedings of the Twenty-eighth Annual SAS Users Group International Conference.* Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The author would like to thank Michele Ensor, Linda Jolley, Jane Stroupe, Bari Lawhorn, Chevell Parker, Sandy McNeill, and David Kelley for reviewing this paper and making suggestions to improve it.

CONTACT INFORMATION

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

Cynthia Zender
SAS Institute, Inc.
Denver Regional Office

Work Phone: 303-290-9112 ext 1738
Email: Cynthia.Zender@sas.com