

A SAS® Output Delivery System Menu for All Appetites and Applications

Chevell Parker, SAS Institute Inc., Cary, NC

ABSTRACT

This document describes how to unleash the power of various ODS destinations to generate fantastic applications and reports. This power is evident when generating spreadsheets using a variety of ODS destinations. The document describes how and when to harness the power of ODS destinations. It describes methods to generate that perfect presentation-quality output. This includes general formatting, creating styles with the addition of templates that are specifically generated for Excel output, and all those little things that lead to great-looking output.

Other menu items include the following:

- arranging output on the page using several approaches, such as ODS LAYOUT; HTMLPanel and TableEditor tagsets; the report-writing features of the DATA step; customizing tables; enhancing output
- creating HTML, which adds the ability to sort, freeze, reorder, and export data dynamically and as other dynamic features
- modifying objects using the ODS Document facility to generate the desired table of contents and bookmarks
- enhancing styles using a variety of methods

INTRODUCTION

Beginning with SAS® 7.0, generating effective, stylish, and functional reports became possible with the SAS Output Delivery System (ODS). The system began with one formatted destination, the ODS HTML destination, and it has evolved to other ODS destinations such as PDF, RTF, ExcelXP, and others.

Regardless of what applications you use or your taste in reporting styles, ODS has a menu of offerings that enable you to build that perfect application or report. This document highlights the ODS menu of destinations and practical applications that you can generate to create effective reports.

MODIFYING THE TABLE OF CONTENTS AND BOOKMARKS

The table of contents (TOC) in the HTML destination, or the bookmark list in a PDF destination, provides a navigational aide or outline to output. This navigational aide, created by ODS, points to specific tables or graphs. As the designer of the report, you must choose the appropriate description and appearance of this outline. To effectively name and organize this table of contents, you can choose from multiple items on the ODS menu, including the following:

- destination-specific items such as ODS Markup for HTML
- the new ODS PDF option in SAS® 9.2, PDFTOC=
- the ODS PROCLABEL statement, the CONTENTS= procedure option, and the DESCRIPTION= option for graphics
- the ODS Document facility, which includes the ODS DOCUMENT statement, the DOCUMENT procedure, and the SASDOC LIBNAME engine

We will start with the simplest method of modifying the table of contents, naming and removing nodes, and will build then from there.

NAMING AND REMOVING NODES

A table of contents is created with ODS HTML and the ExcelXP tagset, while a bookmark list is created with ODS PDF. Renaming items in a table of contents or in a bookmark list is easy with the ODS PROCLABEL statement. You can use this statement to rename the top-most item (the procedure name) that is created from each output object. All output objects create two or more node levels by default. The subsequent levels of this output object vary depending on the procedure that generates it. For example, the PRINT procedure creates a single sub-node, while multiple complex node levels are created by procedures such as the SHEWHART procedure. Procedures such as PRINT, REPORT, TABULATE, and FREQ include the CONTENTS= option, which enables a sub-node to be renamed. Unless you use a BY statement in PROC PRINT and PROC REPORT, the sub-node will be the second level of the output object. You can use the CONTENTS= option with null quotation marks (CONTENTS='') to remove a sub-node. Some graphics procedures, such as GCHART and GPLAT, support the DESCRIPTION= option that you can use to label sub-nodes.

Procedures that do not include the CONTENTS= option have table templates. Using these templates, you can rename a sub-node with the CONTENTS_LABEL attribute within the column attribute. In the following example, Display 1 shows both a default TOC and bookmark output created by the sample code. Note that only the last level in the TOC is a link to the output by default; however, all levels of the PDF bookmark list link to the output.

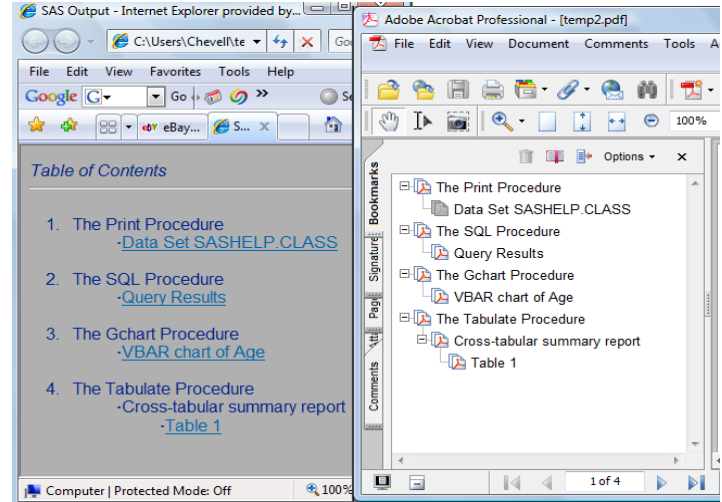
Example 1

```
ods pdf file="temp2.pdf";
proc print data=sashelp.class;
run;

proc sql;
  select * from sashelp.class;
run;
quit;

proc gchart data=sashelp.class;
  vbar age;
run;
quit;

proc tabulate data=sashelp.class;
  class age;
  var height weight;
  table age, height weight;
run;
ods pdf close;
```



Display 1. Default TOC and bookmark list

The next example uses the same procedures, and it adds the ODS PROCLABEL statement and the CONTENTS= option. The ODS PROCLABEL statement renames the first level (the procedure name) of the output object, while CONTENTS="" removes the sub-node (the second level under **The Print Procedure**) and the second and third levels under **The Tabulate Procedure**. The DESCRIPTION= option is used here to rename the sub-node under **The GCHART Procedure**.

Example 2

```
ods pdf file="temp.pdf";

ods proclabel "Detail Report";
proc print data=sashelp.class
  contents="";
run;

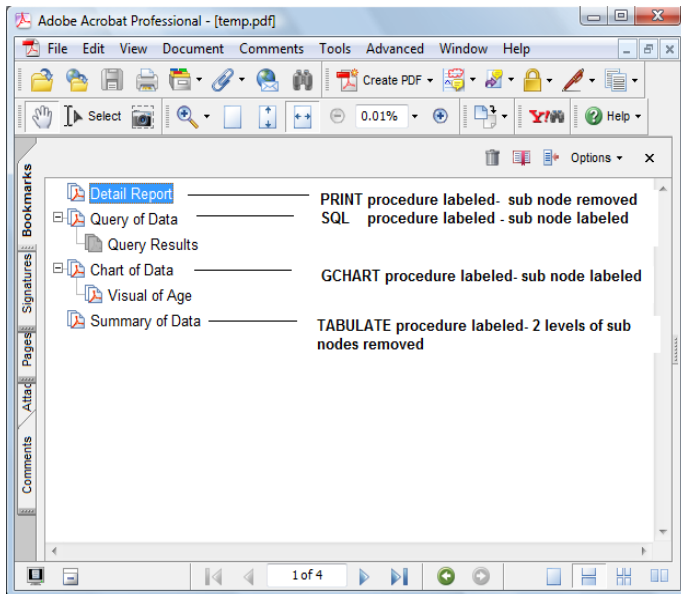
ods proclabel "Query of Data";
proc sql;
  select * from sashelp.class;
run;
quit;

ods proclabel "Chart of Data";
proc gchart data=sashelp.class;
  vbar age / description="Visual of
    Age";
```

```
run;
quit;

ods proclabel "Summary of Data";
proc tabulate data=sashelp.class
  contents="";
  class age;
  var height weight;
  table age, height weight /
  contents="";
run;

ods pdf close;
```



Display 2. Modified bookmark list

DESTINATION-SPECIFIC MODIFICATIONS

Common methods are available for restructuring a TOC in a customized format, ODS also has destination-specific methods you can use, such as ODS Markup Language for HTML or the new ODS PDF option that is available in SAS 9.2. If you are creating HTML, you can use ODS Markup Language to customize and restructure your output. You can use the Markup Language to modify the underlying HTML4 tagset that renders output for the HTML destination. The tagset consists of individual events (for example, the CONTENTS_BRANCH and the CONTENTS_LEAF events) that can be modified to restructure the individual parts of a TOC.

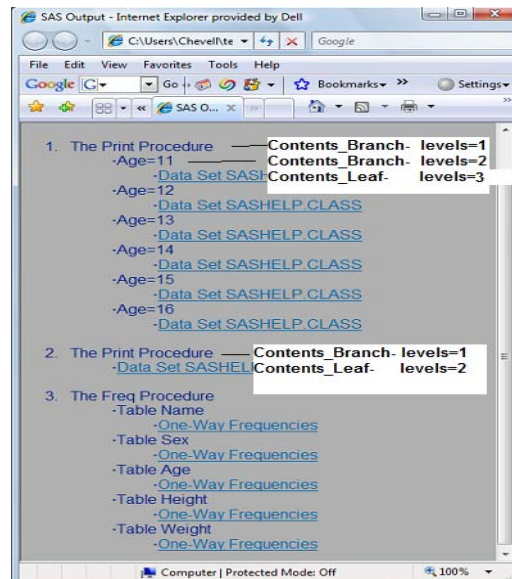
To modify the node levels that are displayed, you can use ODS Markup Language to create an option that you include in the ODS statement, as shown in the next example. This is a simple example of HTML-specific code that you can use to specify the number of levels you want to display.

Example 3

```
proc template;
  define tagset tagsets.test;
    parent=tagsets.html4;

    define event contents_branch;
      start:
        set $test $options['LEVEL'];
        break / if toc_level > $test;
        trigger do_link;
        trigger list;
      finish:
        trigger list finish;
        file=CONTENTS;
    end;

    define event contents_leaf;
      break / if toc_level > $test;
      trigger do_link;
      file = CONTENTS;
    end;
  end;
run;
```



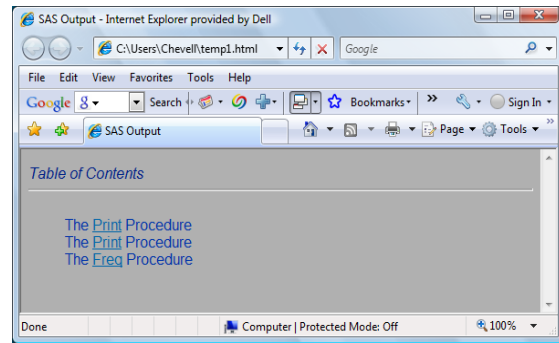
Display 3. Default output that is generated

```
ods tagsets.test file="temp.html"
contents="templ.html"
options(level="1");

proc print data=sashelp.class;
run;

proc print data=sashelp.class;
run;

proc freq data=sashelp.class;
run;
ods tagsets.test close;
```



Display 4. TOC generated using ODS Markup

In this example, you add the new option by using the a SET statement. The SET statement creates a memory variable from the LEVEL= option in the ODS statement. The memory variable that is created is used as a conditional argument along with the TOC_LEVEL automatic markup variable, which contains the current nesting levels of various items. The CONTENTS_BRANCH event represents the top node in the TOC. This node begins with the procedure name and contains one level under it. CONTENTS_BRANCH can also contain more than one level, as illustrated in Display 3. Links to the tables and graphs are controlled by the CONTENTS_LEAF event. Note that you can also modify the BREAK statement to show only a specific level.

In the next example, the PDFTOC= option is added to the SAS 9.2 ODS PDF destination. Along with the methods discussed so far, the PDFTOC= option provides new functionality and control in SAS 9.2. It enables you to specify the number of levels that you want to appear in the bookmark list when the PDF file is open. By default, the option displays all levels in the list when the PDF file is open. This gives you more control than was available in past releases.

Example 4

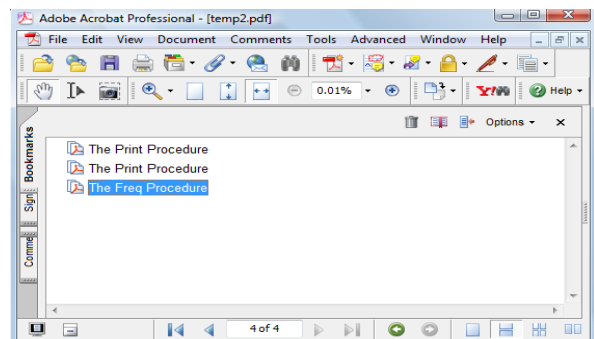
```
ods pdf file="temp2.pdf" pdftoc=1;

proc print data=sashelp.class;
run;

proc print data=sashelp.class;
run;

proc freq data=sashelp.class;
run;

ods pdf close;
```



Display 5. Bookmarks created with the PDFTOC = option

Note: This option does not enable you to restructure or rename the items in the bookmark list.

ODS DOCUMENT FACILITY

Now that your appetite has been whetted with some examples of customizing reports, you are ready to move on to total customization, which is available with the ODS Document facility. The ODS Document facility gives you complete control over your output objects. It enables you to rename, reorder, delete, hide, and import objects. In addition, SAS 9.2 now supports using the REPORT procedure with the Document facility.

The Document facility is useful for tasks such as creating a single node for each output object in the TOC or grouping together all of the PROC PRINT output. To use the Document facility, you first create an ODS document that will be generated from a procedure or a DATA step. Such a document prevents having to store a physical file. You create the document using an ODS DOCUMENT statement.

You can modify objects either interactively (via the ODS Document window within the SAS Display Manager

System) or by using the DOCUMENT procedure. To modify documents interactively, submit the ODSDOC command to display the ODS DOCUMENT window. After you modify and replay your objects in the Document window, you can use the new Document Recorder to generate the code from the actions that you performed. If you are familiar with UNIX commands, this process should be familiar.

Once the document is created, you can use PROC DOCUMENT along with the LIST statement to list the contents and the path of items within a document. Then you can modify the paths to customize the output the way you want it. The output consists of directories, reports, tables, and graphs. To simplify the TOC structure, you can reduce the directory structure in order to make a new directory and copy only the reports, tables, and graphs to this directory. Then you can replay the output.

The following example creates an ODS Document from the procedures used previously in Example 1. PROC DOCUMENT manipulates the objects in this example first by creating a new directory with the MAKE statement. Then, the COPY statement is added to copy the graph and tables from the current directory structure to the new directory \TEST that is created. The branch \TEST is then replayed to the PDF destination, thereby removing the directory structure and displaying just a single table or graph. **Note:** If you have existing graphics in the WORK.GSEG catalog, you might need to use the GREPLAY procedure to delete them before you run the following example. Otherwise, incremental versions of the image (or the GRSEG member) will be created.

Example 5

```
/* Create document. */
ods document name=test(write);

/* Sample code used previously in Example 1. */
ods document close;

proc document name=test;
  list/levels=all;
run;
quit;
```

The following output lists all levels of the document TEST before it has been manipulated:

Listing of: \Work.Test\
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\Print#1	Dir
2	\Print#1\Print#1	Report
3	\SQL#1	Dir
4	\SQL#1\SQL_Results#1	Table
5	\Gchart#1	Dir
6	\Gchart#1\Gchart1#1	Graph
7	\Tabulate#1	Dir
8	\Tabulate#1\Report#1	Dir
9	\Tabulate#1\Report#1\Table#1	Table

Example 6

```
ods pdf file="temp5.pdf";

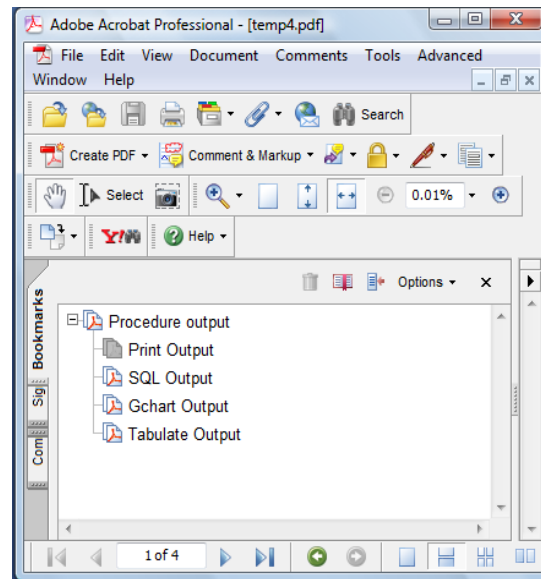
proc document name=test;
  make \test;
  dir \test#1;

  setlabel \test#1 "Procedure
                  output";
  setlabel \print#1\Print#1 "Print
                           Output";
  setlabel \SQL#1\SQL_Results#1
           "SQL Output";
  setlabel \Gchart#1\Gchart#1
           "Gchart Output";
  setlabel
      \Tabulate#1\Report#1\Table#1
      "Tabulate Output";

  copy \print#1\Print#1 to ^;
  copy \SQL#1\SQL_Results#1 to ^;
  copy \Gchart#1\Gchart#1 to ^;
  copy \Tabulate#1\Report#1\Table#1
  to ^;

replay \test#1;

run;
quit;
ods pdf close;
```



Display 6. Output created by PROC DOCUMENT

The following output lists all levels of the document TEST before manipulating the document:

Listing of: \Work.Test\test#1
Order by: Insertion
Number of levels: All

Obs	Path	Type
1	\test#1\Print#1	Report
2	\test#1\SQL_Results#1	Table
3	\test#1\Gchart#1	Graph
4	\test#1\Table#1	Table

ODS DOCUMENT MACRO

You can also use the ODS Document facility with a macro to automate the process of manipulating a table of contents. Example 7 replicates the prior example, but it uses PROC DOCUMENT along with a macro to automate the process. An output data set is created by the LEVELS statement in PROC DOCUMENT. Macro variables are created from each observation where the output is a table, report, or graph. The macro variables are then resolved within the COPY statement and the DOCUMENT procedure. The SET statement labels the items that are replayed. As in Example 6, the MAKE statement creates a new directory where only the tables and graphs are copied and are replayed to the ODS PDF destination, thereby creating a single node for each table and graph.

Example 7

```
ods document name=temp(write);

/* Procedures from Example 1. */
ods document close;
ods output properties=temp;
```

```

proc document name=temp;
  list / levels=all;
run;
quit;

data _null_;
  set temp end=last;
  if type in("Table","Report","Graph") then do;
    count+1;
    call symput('path' || trim(left(count)),path);
  end;
  if last then call symput('total',count);
run;

%macro test;

ods pdf file="temp.pdf";
proc document name=temp;
  make \newfolder;
  setlabel \newfolder "Procedure
  Output";

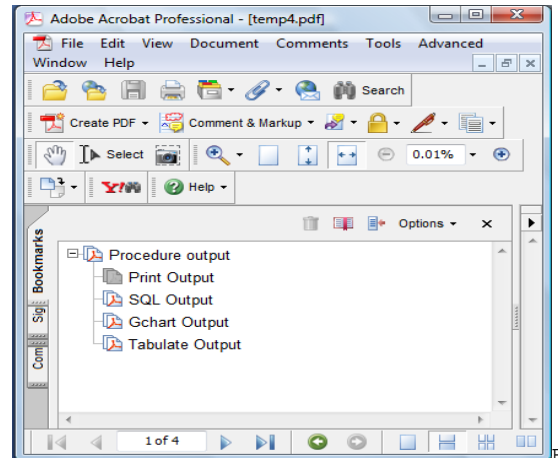
  %do i=1 %to &total;
    %let proc=%substr
    (%scan(&path&i,1,"#"),2);
    setlabel &path&i "&proc
    Output";

    copy &path&i to
    \newfolder#1;

  %end;
  replay \newfolder;
run;
quit;
ods pdf close;
%mend;

%test

```



Display 7. Output that is replayed from PROC DOCUMENT

ARRANGING AND PANELING OUTPUT

SAS 9.2 software provides more features than ever before to generate output in exactly the layout you want by placing data in a specific spot or by paneling output. In this environment, *paneling* refers to arranging tables or graphs across or down a page. Whether you want to place data in a specific spot on a page or to panel output, you can accomplish these tasks with the following items from ODS:

- the ODS LAYOUT feature, which currently works with the PDF and HTML destinations
- the report-writing features of the DATA step
- the HTMLPanel and the TableEditor tagsets for HTML, which is made possible with ODS Markup Language
- the GREPLAY procedure, which has many functions, including the ability to panel output
- The new SAS 9.2 graphics procedures (such as the SGPanel procedure)

The following section provides the syntax for each of these methods as well as a description of how to use them.

ODS LAYOUT

The ODS LAYOUT feature is supported currently only by the PDF destination. However, more destinations might support the feature in future versions of the software. ODS LAYOUT enables you to place output on a page based on the requirements that you specify. ODS LAYOUT permits two types of layouts: absolute and gridded. Both types of layouts start with an ODS LAYOUT statement, contain one or more ODS REGION statements, and end with an ODS LAYOUT END statement.

Absolute layout enables you to explicitly place items on the page. With the absolute layout method, you can match an existing format such as a form, a specialized report (for example, a credit card bill), or any other type of output that requires exact positioning on a page. Such formatting was impossible in previous versions of SAS software. Using this method, you can place the items within a defined region by specifying the height

and width as a standard number or a percentage and by defining the starting coordinates for the output with the X= and Y= options in the ODS REGION statement. If the region is not specified, the coordinates default to 0,0, which is the top left corner of the page (excluding the margins). This method can be problematic, however, in that if you do not define a large enough area, you might lose the output. If that happens, a warning appears in your log.

Gridded layout enables you to define how many columns or rows you want. With this method, ODS LAYOUT dynamically determines how large the region needs to be. Unlike absolute layout, this method enables you to span output across multiple pages.

Example 8 – Absolute Layout

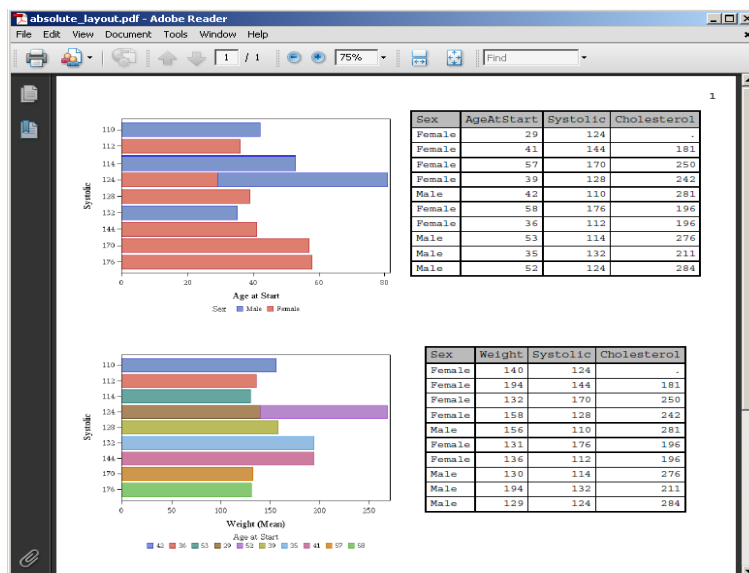
```
ods pdf file="temp.pdf";
ods layout start height=11in width=8in;
ods region width=4in height=5in x=0in y=0in;
ods graphics / height=4in height=4in;
proc sgplot data=sashelp.heart(obs=10);
  hbar systolic / response=ageatstart
  group=sex
  stat=mean;
run;

ods region width=4in height=5in x=4in y=0in;
proc print data=sashelp.heart(keep=sex systolic ageatstart cholesterol obs=10)
noobs;
run;

ods region width=4in height=5in x=0in y=3.5in;
proc sgplot data=sashelp.heart(obs=10);
  hbar systolic / response=weight
  group=ageatstart
  stat=mean;
run;

ods region width=4in height=5in x=4in y=3.5in;
proc print data=sashelp.heart(keep=sex systolic weight cholesterol obs=10)
noobs;
run;

ods layout end;
ods pdf close;
```

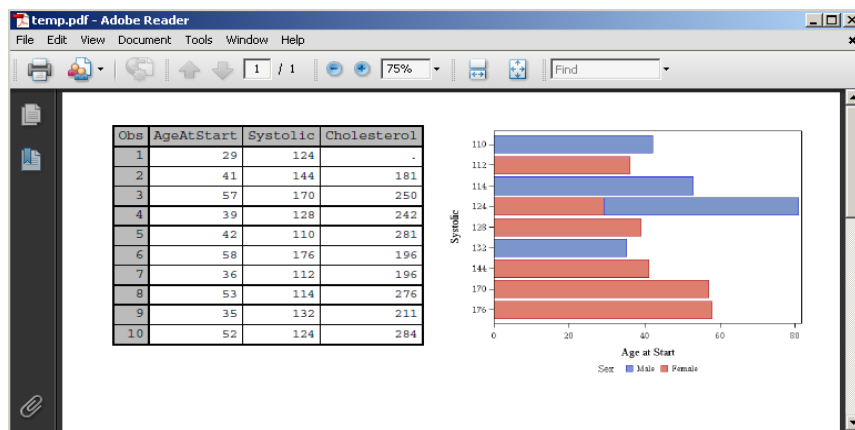


Display 8. Output generated with ODS LAYOUT (absolute layout)

Example 9 – Gridded Layout

```
title;
options nodate nonumber;
ods pdf file='c:\temp\temp.pdf';
ods layout start columns=2 ;
ods region;
proc print data=sashelp.heart(keep=systolic ageatstart cholesterol obs=10);
run;

ods region ;
ods graphics / height=3in height=3in;
proc sgplot data=sashelp.heart(obs=10);
    hbar systolic / response=ageatstart
    group=sex
    stat=mean;
run;
ods layout end;
ods pdf close;
```



Display 9. Output generated with ODS LAYOUT (gridded layout)

HTMLPANEL TAGSET

The HTMLPanel destination, or tagset, generates HTML output for the Web just as the LAYOUT feature is used to generate PDF output. This tagset enables you to panel output. Paneled output is similar to the gridded layout available with ODS LAYOUT. With the HTMLPanel tagset, you can create simple output that panels tables and graphs across a page or complex output that panels across and down a page.

Because the HTMLPanel tagset is a member of ODS Markup Language, it has defined options that determine how many tables or graphs to panel, or span, across or down a page. Manual methods of paneling are also available. For example, you can instruct the tagset where to begin and end paneling by using the EVENT= option in the ODS statement. You can also use the EVENT= option to create row-wise and column-wise panels. **Note:** To see the syntax for all options that are available with this tagset, use the option DOC="HELP".

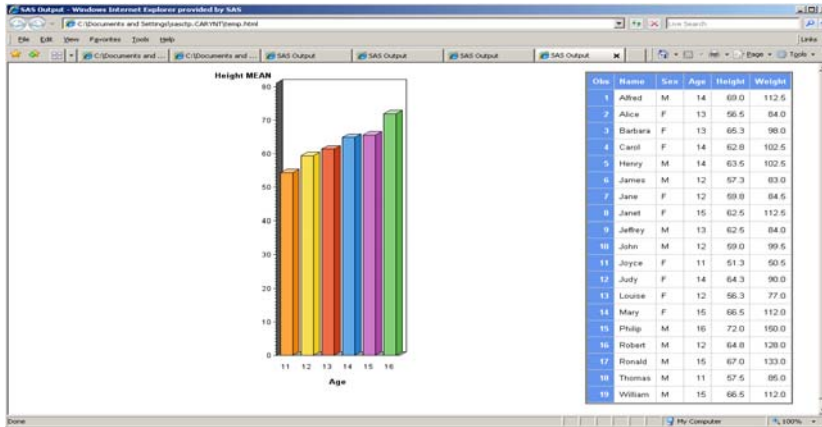
Example 10

```
ods tagsets.htmlpanel file="temp.html" style=sasweb;
ods tagsets.htmlpanel event=panel(start);

proc gchart data=sashelp.class;
    vbar3d age / discrete sumvar=height type=mean patternid=midpoint;
run;
quit;
```

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

ods tagsets.htmlpanel event=panel(finish);
ods tagsets.htmlpanel close;
```



Display 10. Output generated from the HTMLPanel tagset

TABLEEDITOR TAGSET

The TableEditor destination, or tagset, also enables simple paneling of HTML output through the use of the PANELCOLS= option in the ODS statement. With the TableEditor tagset, you can add tables and graphs side by side in addition to down a page. This tagset also many other features that you can use in conjunction with paneling functionality just by supplying options to the tagset. The TableEditor tagset inherits from the HTML4 tagset, which is used to render the HTML destination. The paneling feature of this tagset is not as robust as the HTMLPanel tagset.

This tagset also enables you to add tabs along with paneled output. This feature is useful if you want to view or print only a subset of your data. Tabs are created with the WEB_TABS= option, which works similarly to creating sheet names in Microsoft Excel. Clicking on a specific tab displays only the selected output in the browser window. All other output remains hidden until you select the appropriate tab. You can find more information about the TableEditor tagset in the section [“ODS Markup Language and the TableEditor Tagset”](#).

As shown in the following example, you indicate where you want paneling to be by using the PANELCOLS= option in the OPTIONS parameter of the ODS TAGSETS.TABLEEDITOR statement. To add tabs to your paneled output, name each piece of output, separated by quotation marks, in the WEB_TABS= option. To prevent a tab from being displayed, use the name HIDE in the OPTIONS parameter. In Example 11, the first and third procedures display the tabs, while the tabs for the second and fourth procedures are hidden.

Example 11

```
ods tagsets.tableeditor file="templ.html"
options(panelcols="2"
        web_tabs="Graph/Summary,Hide,Detail,Hide"
        pageheight="500") style=styles.sasweb;

proc gchart data=sashelp.shoes;
  vbar3d region / sumvar=sales type=sum patternid=midpoint;
  title "Sales by Region";
run;
quit;

proc means data=sashelp.shoes;
  class region;
  var sales;run;
```

```
ods tagsets.tableeditor options(panelcols="2");

proc gplot data=sashelp.shoes;
    plot region*sales;
run;
quit;

proc print data=sashelp.shoes;
    title "Sales Detail by Region";
run;
quit;
ods tagsets.tableeditor close;
```



Display 11. Paneled output generated with the TableEditor tagset

REPORT WRITING AND THE DATA STEP

The object-oriented DATA step is currently a preproduction feature in SAS 9.2. This feature provides a way to generate reports that are even more customized than what you could generate in previous versions of SAS and ODS. This feature also works with all destinations except the listing destination. The object-oriented DATA step enables you to explicitly build your output one cell at a time, which gives you full control of the layout. Because you are building one cell (the lowest level) at a time, the DATA step syntax can be verbose.

To begin creating output with the object-oriented DATA step, you first have to declare a DATA step object with a DCL statement, as shown here:

```
dcl odsout obj();
```

After you declare the object, you can apply methods to it that define the object's behavior. For example, methods such as IMAGE(), TITLE(), and PAGE() instantiate the object and instruct it to add images, titles, and explicit page breaks. Some of the more popular methods are those used to generate a table, for example, TABLE_START(), ROW_START(), and CELL_START(). These methods, respectively, start a table, add a row, and add a cell. The ROW_END() and TABLE_END() methods end a table, as it is closed, in the same order it was started. If you are familiar with HTML syntax, building tables with these object-oriented DATA step methods should sound familiar because both tools use the same concept of the table structure. The object-oriented DATA step methods can also contain

- arguments that define what text or variables to display
- various options, including the OVERRIDES option, which enables you to pass style attributes

The following example uses the FORMAT.CELL() method and an override to specify the various style attributes (OVERRIDES=, JUST=, and FONT_SIZE=).

```
obj.format_cell(text:"This is an example", overrides:"just=center font_size=18pt");
```

The object-oriented DATA step also enables you to specify the layout of the output, including both absolute and gridded layouts. When you create a layout, you need to specify one or more regions for the layout. Example 12 uses the GRIDDED() and REGION() methods to create a gridded layout with two columns. The first region contains a table with columns and rows that span the table, and the second region imports an image.

Note: This example shows only a segment of the full code that generates the output described above. To download the full object-oriented DATA step, see ["References."](#)

Example 12

```
data one;
  input cash_asset inventory_asset accountr_asset return_liab retain_liab;
  net_income=sum(cash_asset,inventory_asset,accountr_asset)-
    sum(return_liab,retain_liab);
  total_assets=sum(cash_asset,inventory_asset,accountr_asset);
  total_liab=sum(return_liab,retain_liab);
  balance=total_liab+net_income;
  cards;
20000 50000 10000 30000 30000
;
run;

options nonumber nodate nocenter;
ods listing close;
ods pdf file='c:\temp\test10.pdf' notoc;
ods escapechar='^';
title;

data _null_;
  set one;
  if _n_=1 then do;
    dcl odsout obj();
  end;
  obj.title(text:"^S={just=c} Balance Sheet For XYZ company",
    overrides:"just=c");
  obj.layout_gridded(columns:2);
  obj.region();
  obj.table_start(overrides: "just=1 ");

  obj.row_start();
  obj.format_cell(text: "^S={Font_size=18pt} Trial Balance Sheet",
    column_span:3,overrides:"just=c background=skyblue");
  obj.row_end();

  obj.row_start();
  obj.format_cell(text: "Item",overrides:"font_size=14pt");
  obj.format_cell(text: "Assets",overrides:"background=yellow");
  obj.format_cell(text: "Liabilities",overrides:"background=red");
  obj.row_end();

  obj.row_start();
  obj.format_cell(text: "Cash");
  obj.format_cell(text:put(cash_asset,dollar.),
    overrides:"background=yellow just=r");
  obj.format_cell(text:"$0",row_span:2,
    overrides:"just=r background=orange vjust=m");
  obj.row_end();

  . . . more method statements. . .

  obj.row_start();
  obj.format_cell(text:"Company has been certified as viable for
    2008",column_span:3,overrides:"just=c
    background=yellow");
  obj.row_end();
  obj.table_end();

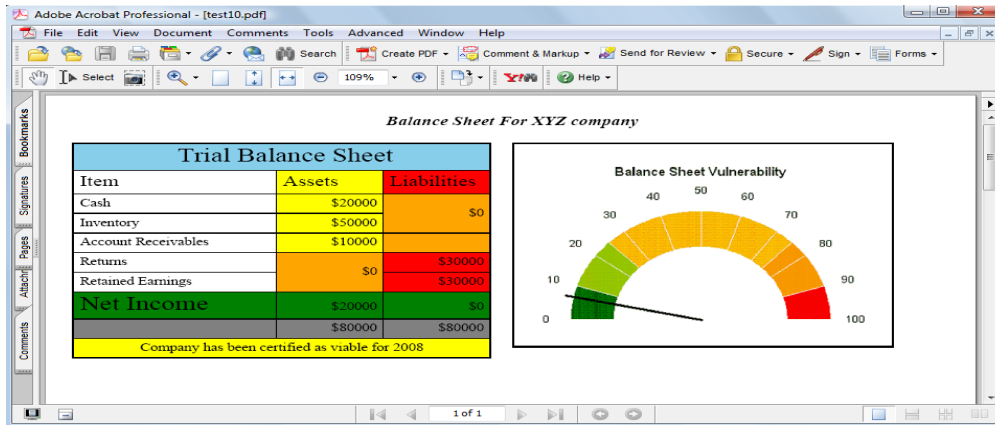
  obj.region();
  obj.table_start();
  obj.row_start();
```

```

        obj.format_cell(text:"",overrides:"preimage="c:\balance_sheet5.gif"");
        obj.row_end();
        obj.table_end();
        obj.layout_end();
run;

ods pdf close;

```



Display 12. Object generated with a DATA step

GENERATING EFFECTIVE WEB APPLICATIONS

ODS has quite a few features on its menu that help you generate effective Web applications. Practically everyone has access to a Web browser, so a Web application is a logical choice for your reports. The ODS HTML destination is the backbone of the system when it comes to generating Web applications. The HTML destination is great for creating links from discrete parts on a page that enable you to drill down to summary or detail information. You can also enhance and control your Web output in a variety of ways, such as the following:

- using ODS Markup Language and tagsets
- creating dynamic output with products as SAS/IntrNet[®] software
- creating stored processes, which are rendered by client software such as SAS[®] Enterprise Guide[®], the SAS[®] Stored Processes Web application, or other SAS client software

DRILL-DOWN TECHNIQUES

Drill-down techniques enable you to start at a high level in a table or image and use hyperlinks to navigate to information on the same page, to deeper levels of information within the same application, or to information in other applications or files. For example, you might link to an Excel spreadsheet (or a specific cell within the sheet), to bookmarks or to a page in a PDF or an RTF file, or to a specific slide in a PowerPoint application. This technique is very effective for analyzing data further.

Various methods for creating hyperlinks are available within SAS: using formats, creating anchor tags or links within a DATA step, using the TEMPLATE procedure, using styles, and so on. In addition, if you are linking from a Web page to a PDF file, Adobe Acrobat also supports commands that you can use in the link syntax to determine what zoom mode, page mode, view, and other properties are in effect when a PDF file is open.

This next example illustrates how you can use PROC REPORT to create hyperlinks. To create a drill-down hyperlink with PROC REPORT, you use a CALL DEFINE statement along with the URL attribute and a compute block. These components enable you to create correct links, regardless of the destination that you are using. As shown in Example 13, the CALL DEFINE statement is created within the COMPUTE statement block. When you specify the URL access method in the CALL DEFINE statement, PROC REPORT uses the path you specify to construct the link for the cell. This method eliminates having to know destination-specific tagging to create the link. You can also modify the specified path so that the link changes based on the contents of the cell.

Example 13

```
/* Create a sample Excel worksheet. */
ods tagsets.excelxp file="c:\temp\excel.xls"
options(sheet_interval="bygroup"
        sheet_label=" "
        embedded_titles="yes");

proc print data=sashelp.prdsale noobs;
  by country;
  title "Report for the #byval(region)";
run;

ods tagsets.excelxp close;
ods html file="temp.html" style=sasweb;

proc report data=sashelp.prdsale nowd;
  col country region predict actual division prodtype product ;
  title "Linking to Excel";

  compute country;
    tmp_region="c:\temp\excel.xls#" || trim(left(country)) || "!A1";
    call define(_col_, 'URL', tmp_region);
  endcomp;
run;

ods html close;
```

You can also generate drill-down links with graphics procedures (for example, the GCHART procedure) that support the HTML= option. In such a procedure, you create a variable that contains the path to which you want to link. Then, you specify the variable in the HTML= option of an action statement (in Example 14, the VBAR statement). Graphics procedures, such as PROC GCHART, build the appropriate destination-compatible links. Then, you can click on these links to drill down to other information, tables, or graphs.

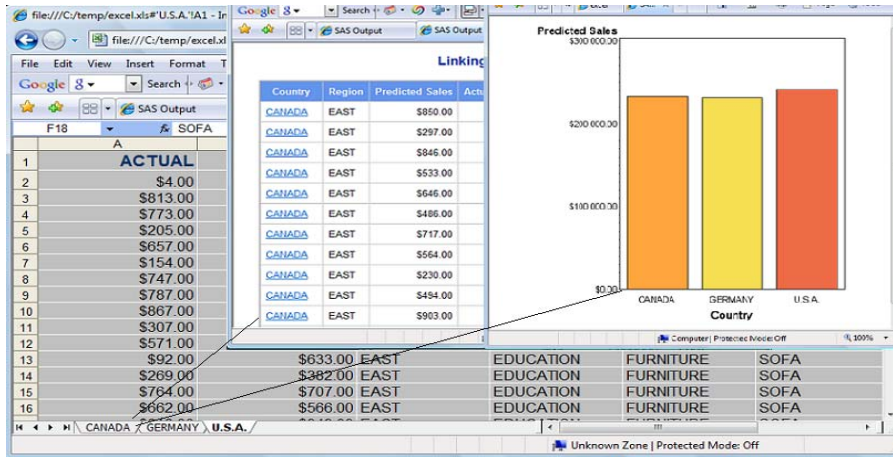
Note: The ability to drill down to a specific sheet from HTML is limited to files addressed using the Http:// addressing of the file.

Example 14

```
data one;
  set sashelp.prdsale;
  linkvar="href="c:\temp\excel.xls#" || trim(left(country)) || "!A1"";
run;

goptions hsize=5in vsize=5in;
ods html file="temp.html" style=sasweb;

proc gchart data=one;
  vbar country / sumvar=predict patternid=midpoint type=sum html=linkvar;
run;
quit;
ods _all_ close;
```



Display 13. Drilling down in a table and graphics in an Excel worksheet

ODS MARKUP LANGUAGE AND THE TABLEEDITOR TAGSET

ODS Markup Language and tagsets are powerful tools that give you complete control over output that is generated when you create Web applications. These tools give you control over the tags that are generated using the ODS Markup Language. For example, with ODS Markup Language you can rename, reorder, and remove tagging. A

powerful feature of ODS Markup Language (beginning with SAS 9.1.3) is the ability to add options. Adding options enables you to trigger various pieces of functionality, regardless of the procedure that is being executed.

In particular, the TableEditor tagset enables you to greatly enhance a Web application with functions such as a dynamic sorting of data by clicking on the column headings, filtering data, freezing headers and row header, adding images and JavaScript, and alternating colors. For details about the TableEditor tagset, see "Creating a Data Grid Like VB.NET" (support.sas.com/rnd/base/ods/odsmarkup/tableeditor/index.html). **Note:** The TableEditor tagset is not installed with SAS, but you can download it from the Web page mentioned here or by using the URL access method in a FILENAME statement, as follows:

```
filename temp url
    'http://support.sas.com/rnd/base/ods/odsmarkup/tableeditor/tableeditor.tpl';
%include temp;
```

Example 15 uses options with the TableEditor tagset to add sorting and filtering on a column, a banner, alternate background colors, and frozen headers to a Web application.

Example 15

```
ods tagsets.tableeditor file="temp.html" style=sasweb
options(sort="yes" doc="help" sort_underline="yes"
      sort_arrow_color="Red" frozen_headers="yes"
      banner_color_even="pink" highlight_color="cyan"
      autofilter="yes" autofilter_width="4em"
      image_path="c:\sas.gif"
      image_just="right" print_dialog="yes");

proc print data=sashelp.retail;
  format date mmddyy8.;
  title "Data Grid Created with the TableEditor";
run;
ods tagsets.tableeditor close;
```


Data Grid Created with the TableEditor

Obs	SALES	DATE	YEAR	MONTH	DAY
1	\$220	01/01/80	1980	1	1
2	\$257	04/01/80	1980	4	1
3	\$258	07/01/80	1980	7	1
4	\$295	10/01/80	1980	10	1
5	\$247	01/01/81	1981	1	1
6	\$292	04/01/81	1981	4	1
7	\$286	07/01/81	1981	7	1
8	\$323	10/01/81	1981	10	1
9	\$284	01/01/82	1982	1	1

Display 14. Output from the TableEditor tagset

You can also use the options with the TableEditor tagset to export output to Excel and to modify the Excel file that is generated. You can also directly create a native Excel file. Options are available that enable you to modify orientation, add filters, freeze headers, change the zoom, add pivot tables, provide graphics, save files with various formats, and so on. When you specify any of the Excel options, an **Export** button is added to the HTML file. The only time this is not the case is when you specify either the AUTO_EXCEL option (which automatically generates the Excel file when the HTML is loaded in the browser) or the AUTO_FORMAT_SELECT (which exports the file when a style is selected). If you do not need the HTML file, it can be created as a temporary file. You can change the text of the **Export** button by specifying new text in the BUTTON_TEXT= option.

Example 16

```
ods tagsets.tableeditor file="temp.html"
options(excel_autofilter="yes"
       excel_zoom="90"
       excel_scale="90"
       excel_frozen_headers="yes"
       excel_orientation="landscape"
       sheet_name="Retail,Class,Citimon"
       excel_table_move="1,2,3"
       auto_format_select="yes");

proc print data=sashelp.retail;
  title "Export to Excel by Selecting
    Styles";
run;
proc print data=sashelp.class;
run;
proc print data=sashelp.citimon;
run;
ods tagsets.tableeditor close;
```

Export to Excel by Selecting Styles

Display 15. Selecting Excel styles and adding Excel Formats

USING ODS WITH STORED-PROCESS APPLICATIONS

The custom tagset can also be used with dynamic applications such as a stored process. To execute a stored process, you start with the %SSTPBEGIN macro, which initializes ODS and generates output to the client. As shown in the following example, you control ODS options in the process by using reserved global macro variables. These variables control everything from the style to the ODS destination, and they must be specified before the %STPBEGIN macro. The process is ended with the %STPEND macro. When you are using a stored process, the default destination is HTM.

Example 17

```
%let _ODSDEST=tagsets.tableeditor;
%let _ODSSTYLE=SASWEB;
%let _ODSOPTIONS=RS=NONE;
%let _GOPT_DEVICE=activex;

* processbody;

%stpbegin;

proc print data=sashelp.class;
run;

%stpend;
```

For more details on stored processes, see “SAS Stored Processes Web Application” (support.sas.com/rnd/itech/doc9/dev_guide/stprocess/stpwebapp.html).

CREATING AND ENHANCING STYLES

Creating styles have never been easier than it is with SAS 9.2. Numerous software enhancements make this process much easier, and ODS has capabilities to accommodate every style. You create styles, primarily, with PROC TEMPLATE, cascading style sheets (CSS), and inline formatting. PROC TEMPLATE creates a style template that is a member of an item store. Templates are specific to SAS and work across all ODS destinations, which enables formatting. Cascading style sheets, which are simply text files, are an industry standard for formatting markup output. Inline formatting, which became a production feature in SAS 9.2, can also be used to format text or data locally.

SAS® 9.2 TEMPLATE PROCEDURE

Style inheritance in PROC TEMPLATE has been enhanced in SAS 9.2. The inheritance cycle has been modified to add the following templates:

- Base.Template.Style, which is the parent of all style definitions and does not have a parent itself
- Base.Template.Table, which is the parent for all tables

Modifying this template provides the functionality to alternate row and column colors. In addition, the REPLACE statement has been deprecated in favor of the DEFINE STYLE and the CLASS statements. This enhancement reduces the length of code that is required to modify a template.

In the following SAS 9.1 example, the REPLACE statement specifies the complete list of indirect style references within the style element GraphFonts that is required to modify the indirect style reference GraphTitleFont.

Example 18 (SAS 9.1)

```
proc template;
  define style Styles.NewStyle;
    parent=styles.default;

    replace GraphFonts "Fonts used in graph styles" /
      'GraphDataFont'=('Arial',8pt)
      'GraphValueFont'=('Arial',10pt)
      'GraphLabelFont'=('Arial',12pt,Bold)
      'GraphFootnoteFont'=('Arial',12pt,Bold)
      'GraphTitleFont'=('Arial',14pt,Bold);
  end;
run;
```

However, with the SAS 9.2 DEFINE STYLE and the CLASS statements, only the indirect style is required, as shown here.

Example 19 (SAS 9.2)

```

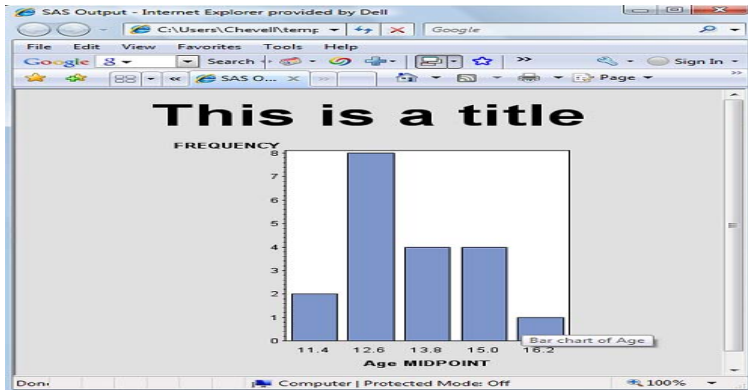
proc template;
  define style Styles.NewStyle;
    parent=styles.default;
    class GraphFonts /
      'GraphTitleFont'=('Arial',14pt,Bold);
    end;
run;

ods html file="temp.html" style=Styles.NewStyle;

. . .PROC GCHART code. . .

ods html close;

```



Display 16. Title font modification using the new CLASS statement

The CLASS statement creates liked-named style elements; therefore, all three of the following statements do the same thing:

- `class GraphFonts;`
- `style GraphFont from GraphFontss;`
- `style Graphfonts from _self_;`

CASCADING STYLE SHEETS

Cascading style sheets (CSS) are created by default with some destinations such as the HTML destination (beginning in SAS®9). CSS files can be either external or embedded. However, the HTML destination creates an embedded CSS file by default. Prior versions of the HTML destination that used HTML 3.2 used tagging to provide the formatting. You can also apply your own existing CSS file by using the `STYLESHEET=` option with the `URL=` suboption. **Warning:** Omitting the `URL=` sub-option overwrites the existing CSS file.

You can also specify multiple CSS files within the `URL=` suboption, but they must be separated by commas. If your file path contains spaces, as shown in Example 20, you must include `%20` in place of the space in the file path. Otherwise, ODS will recognize what follows a blank space as the beginning of a new CSS file. Removing the `URL=` sub-option enables you to create a CSS file based on the style.

Example 20

```

ods html file="temp.html" stylesheet=(url="c:\documents%20settings\file.css");

proc print data=sashelp.class;
run;

ods html close;

```

READING CUSTOM CSS FILES

In a perfect world, your CSS files would match the CSS files in SAS software, and ODS could use the same style-sheet rules. However, that is not the case. If you have a custom CSS file that does not follow the naming conventions

ODS expects, you must specify what ODS should do with the various class or ID selectors in your file. If the class selectors are not what ODS expects, the application simply ignores them. Therefore, you must modify your CSS file so that its naming conventions correspond to what ODS expects, or you need to associate ID or class selectors in the file to the corresponding style elements within PROC TEMPLATE. The style attributes that are used to associate the CLASS and ID within PROC TEMPLATE are the HTMLCLASS= and the HTMLID= attributes. These attributes enable you to use a class or an ID from your custom CSS file, which does not contain the same naming convention as what ODS expects. The CLASS= HTML attribute generated in a file that is created by a SAS[®]9 ODS HTML statement consists of multiple classes. The first class represents the justification, and the second class represents the presentation.

In the following example, the left column illustrates a CSS style-sheet rule with various style properties that you might want for the cell value of an HTML file. The right column illustrates a TEMPLATE procedure that uses the style from the CSS file.

Example 21

Custom CSS file

```
.Cell_Value
{
  font-family: Arial, Helvetica, sans-ser
  font-size: small;
  font-weight: normal;
  font-style: normal;
  color: #000000;
  background-color: #D3D3D3;
}
```

TEMPLATE Procedure with HTMLCLASS= Option

```
proc template;
  define style styles.test;
    parent=styles.default;
    style data from data /
      htmlclass="cell_value" ;
  end;
run;
```

Before you use the HTMLCLASS= option, the tag that is used in the HTML file is `<td class=" " data=value>`. After you use the HTMLCLASS= option, the tag that is used in the HTML file is `<td class=" " cell_value=value>`.

THE CSSSTYLE= OPTION

SAS 9.2 introduces the CSSSTYLE=ODS option, which enables you to format output that is generated for the HTML, RTF, and PDF destinations. To format the output, you must provide a CSS file path as an argument. The class selector in the CSS file must follow the conventions that ODS expects. This option is also useful in software such as Base SAS[®] and SAS Enterprise Guide. For example, you can use modified styles that are generated with the Style Manager in SAS Enterprise Guide.

Example 22

```
ods pdf file="temp.pdf" cssstyle="c:\your-file.css";

proc print data=sashelp.class;
run;

ods pdf close;
```

IMPORTING CSS FILES

Another major enhancement in SAS 9.2 is the ability to import a CSS file into a template definition. Before SAS 9.2, if you needed a template definition that could be used with the RTF and PDF destinations, you had to manually convert the CSS style into the appropriate template definitions. With SAS 9.2, PROC TEMPLATE has an IMPORT statement that imports a CSS file and creates a template style definition from it. This template definition can then be used by all ODS destinations that support styles. The class selectors must match the naming conventions that ODS expects in the CSS file.

In the following example, the left column illustrates a CSS file that is imported by the TEMPLATE procedure in the left column. The templates in the procedure use classes from the parent template, with the exception of those classes (.Data, Header, and Rowheader) that are imported from the CSS file.

Example 23

CSS file

```
.Data, .Header, .Rowheader {  
    background-color: white;  
    color:blue;  
    font-size: small;  
}
```

Template Procedure

```
proc template;  
    define style styles.Newstyle;  
        parent=styles.default;  
        import "StyleSheet.css";  
    end;  
run;  
  
ods html file="temp.html"  
style=styles.Newstyle;  
proc print data=sashelp.class;run;  
ods html close;
```

MICROSOFT OFFICE APPLICATIONS

There are a variety of techniques available that enable you to integrate SAS output into Microsoft Office applications. These techniques involve the use of several tools and applications, including ODS, dynamic data exchange (DDE), the EXPORT procedure, SAS Enterprise Guide, SAS® Add-in to Microsoft Office, the XML Engine, the RTF destination, and so on. With all of these available methods, how do you know which one to use? The answer depends on your application requirements, what versions of SAS and Microsoft applications you have, and what kind of output you need. For example, if you want presentation-quality output with Excel that includes formatting such as *traffic-lighting* (highlighting data based on value) and hyperlinks, you can use tools such as the ODS ExcelXP or the ODS MSOffice2K destinations. However, if you need a file that is similar to the appearance of default Excel file or a very small file, then the CSV destination is probably a better choice. This section concentrates specifically on ODS methods for generating presentation-quality output.

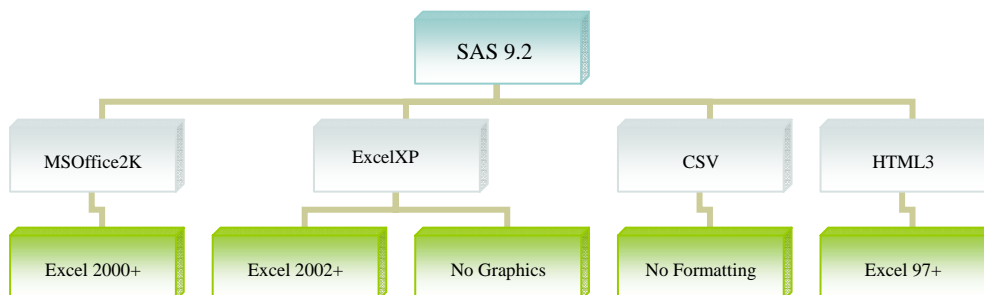


Figure 1. ODS destinations that are used to generate output for Microsoft Excel

THE ODS EXCELXP DESTINATION

In SAS 9.1 or later, the ExcelXP destination is the preferred method for generating Excel output. This destination is powerful, flexible, easy to use, and works on all operating systems. It follows the SpreadsheetML format, and it is the first non-native format for Excel to support multiple worksheets per workbook. ExcelXP uses ODS Markup Language to emulate the SpreadsheetML format. ODS Markup Language enables a tagset request to be transformed into the XML Spreadsheet format, which can be read by Excel 2002 or later. Since SAS 9.1.3, ODS Markup Language has had the ability to add options. The ExcelXP destination takes advantage of this feature, with more than 50 options in its tagset.

The ExcelXP options enable you to modify most of the more commonly used features in Excel, including adding frozen headers and row headers, generating filters, fully customizing the page setup, and so on. The 2003 SpreadsheetML specification has some limitations. Those limitations include the inability to generate graphics, outlines, and macros, among other things. For most other items, there are options that can be used to drive this behavior.

The following example uses some common display options that generate presentation-quality output. (To see a full list of options and syntax, use the DOC="HELP" option. The list of options and syntax will appear in your log.) This example embeds the title within the worksheet, adds frozen headers and rows, adds filters, specifies a column width and automatic cell height adjustment, modifies the default sheet names so that it specifies BY groups, removes the sheet label, and suppresses the BY line.

Example 24

```
ods tagsets.excelxp file="temp8.xls" options(doc="help" embedded_titles="yes"
                                             autofilter="1-3" frozen_headers="3"
                                             frozen_rowheaders="1"
                                             absolute_column_width="8.5,11,7,9,8,8"
                                             autofit_height="yes"
                                             sheet_interval="bygroup"
                                             sheet_label=" "
                                             suppress_bylines="yes") style=normal;

proc print data=sashelp.shoes noobs;
  title "Detail of Region #byval(region)";
  by region;
run;

ods tagsets.excelxp close;
```

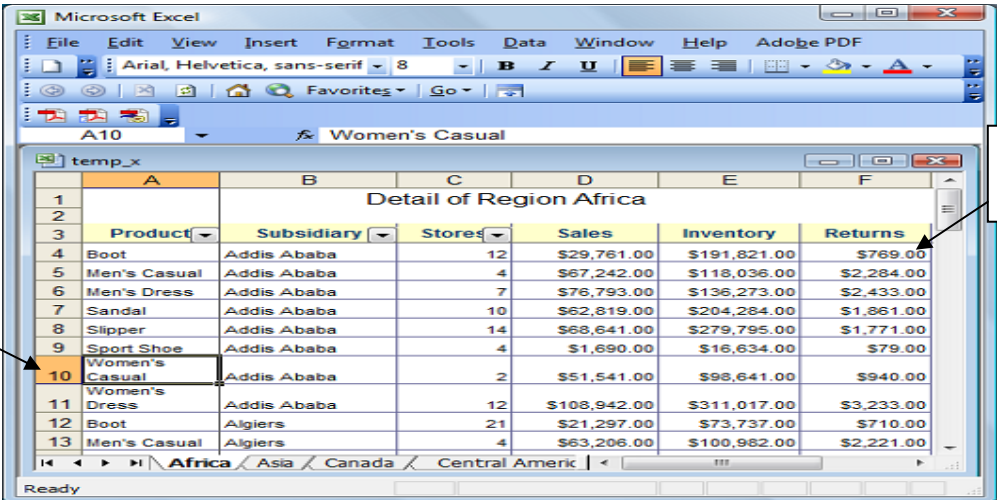


Figure 2 shows a screenshot of Microsoft Excel displaying the output of the SAS code. The spreadsheet is titled "temp_x" and contains a table with columns A through F. The first three rows are frozen headers. The first column (A) contains product names, some of which are wrapped. The second column (B) contains subsidiary names. The third column (C) contains store counts. The fourth column (D) contains sales figures. The fifth column (E) contains inventory figures. The sixth column (F) contains returns figures. The table is titled "Detail of Region Africa".

Product	Subsidiary	Stores	Sales	Inventory	Returns
Boot	Addis Ababa	12	\$29,761.00	\$191,821.00	\$769.00
Men's Casual	Addis Ababa	4	\$67,242.00	\$118,036.00	\$2,284.00
Men's Dress	Addis Ababa	7	\$76,793.00	\$136,273.00	\$2,433.00
Sandal	Addis Ababa	10	\$62,819.00	\$204,284.00	\$1,861.00
Slipper	Addis Ababa	14	\$68,641.00	\$279,795.00	\$1,771.00
Sport Shoe	Addis Ababa	4	\$1,690.00	\$16,634.00	\$79.00
Women's Casual	Addis Ababa	2	\$51,541.00	\$98,641.00	\$940.00
Women's Dress	Addis Ababa	12	\$108,942.00	\$311,017.00	\$3,233.00
Boot	Algiers	21	\$21,297.00	\$73,737.00	\$710.00
Men's Casual	Algiers	4	\$63,206.00	\$100,982.00	\$2,221.00

Figure 2. Using ExcelXP options to generate output

CELL FORMATTING

When you are exporting data to Excel, you need to be aware of how your data is handled. For example, if you export data to Excel and no Excel number format is applied, Excel applies the General format, by default. The General format tries to determine the best way to format a number. But the General format does not always provide the best format, so it is best to apply a specific format when you want numbers to appear a certain way.

In addition, a number of common problems occur when the General format is applied:

- Leading and trailing zeros are truncated.
- Numbers with a length greater than 11 are placed in scientific notation.
- The thousands separator (a comma) is deleted when you use the ExcelXP tagset to export to Excel.
- Strings that contain dashes might be incorrectly considered to be date values.
- A negative dollar format (for example, \$-3,333.00) is treated as text because it does not match the Excel currency format.

EXCEL NUMBER FORMATS

To apply Excel number formats using the ExcelXP destination, you use the TAGATTR= attribute along with the parameter FORMAT: and the Excel format that you want to use. The XML Spreadsheet format has three data types: String, Number, and DateTime. You can modify these same data types for a cell by using the TAGATTR= option along with the TYPE: parameter. To format a value, you either select an Excel number format or create your own custom formats. You can solve each of the problems mentioned previously by specifying one of the Excel number formats.

Specifying a custom Excel format is similar to specifying the PICTURE format in the FORMAT procedure. The

PICTURE format uses the numbers 0 and 9 as digit selectors, while Excel uses the pound sign (#) and the number 0 as digit selectors. For example, the `#,##0` number format instructs Excel to use commas. The pound sign is a placeholder for the comma, and it instructs Excel not to include digits if no value is provided. If you want to retain decimal places, you can create the custom format `#,##0.00`, which guarantees that two decimal places will be added to the end of the value. If the decimal places do not exist, the format specifies zeros. You can specify a text format by using either the format name `TEXT` or the at-sign (`@`). When you specify a text format, the value cannot be modified. If you specify the value **string** for the TYPE: parameter in the TAGATTR= attribute, your data will be of type String.

The following example uses a combination of text and custom formats to maintain desired formatting.

Example 25

```
data one;
  input leading_z trailing_z long_n $22. long_s $14. comma_f dollar_f;
  format leading_z z6. comma_f comma10. dollar_f dollar10.;
cards;
000001 1234.50 1234567891011121456789 1234567891012 1234567 -1234567
;
run;

ods tagsets.excelxp file="temp11.xls"
options(absolute_column_width="6,8,14,16,4,7,7" embedded_titles="yes"
sheet_interval="None") style=normal;

proc print data=one;
  title "Values prior to Excel formats applied";
run;

proc print data=one;
  title "Values after the Excel formats applied";
  var leading_z / style(data)={tagattr="format:@"};
  var trailing_z / style(data)={tagattr="format:#####.#0"};
  var long_n / style(data)={tagattr="format:@"};
  var long_s / style(data)={tagattr="format:#,###"};
  var comma_f / style(data)={tagattr="format:#,###"};
  var dollar_f / style(data)={tagattr="format:$#,###;$-#,###"};
  format dollar_f;
run;

ods tagsets.excelxp close;
```

Obs	leading_z	trailing_z	long_n	long_s	comma_f	dollar_f
1	000001	1234.50	1234567891011121456789	1,234,567,891,012	1,234,567	\$-1,234,567

Display 17. Using Excel formats to customize output

In the following example, the number formats consist of up to four parts, which are separated by semicolons:

`$#,##0;[Red]$,##0;###0 ;@`

Positive Negative Zero Text

The first number format is for positive values; the second format is for negative values; the third format is for zero values; and the fourth format is for text.

At times, you might want to format positive and negative numbers separately, which requires two parts to the number format, as shown here:

[Blue][>=1000]\$#,##0;[Red][<0]\$#,##0;\$#0;[white]@

Positive Negative Zero Text

The first format adds conditions using number formats. In this example, the conditions are blue for positive numbers greater than or equal to 1000, red for negative values, and white for text values. Example 26 illustrates how you code these conditions.

Example 26

```
data one;
    input widget $ sales;
    cards;
Wii 250
Nintendo 450
Barbie 1000
Ken -100
Train 0
Other .A
;
run;

ods tagsets.excelxp file="tempd.xls";

proc print data=one;
    var widget;
    var sales /
    style(data)={tagattr="format:[Blue][>=1000]$#,##0;[Red][<0]$#,##0;$#0;[white]@"};
run;

ods tagsets.excelxp close;
```

Obs	widget	sales
1	Wii	\$250
2	Nintendo	\$450
3	Barbie	\$1,000
4	Ken	\$100
5	Train	\$0
6	Other	\$A

Display 18. Output generated with conditional formats

THE MSOFFICE2K DESTINATION

The MSOffice2K destination, which became available in SAS®9, generates HTML, by default. This destination replaced the often-used HTML destination, which was used in SAS 8.2 to generate output for Excel. The format is easy to use and is flexible; that is, you can create styles with both PROC TEMPLATE and CSS files. In addition, it allows graphics and tables, and it works on all operating systems.

The HTML that is generated with the MSOffice2K destination works well with the Microsoft products that can handle HTML, such as Excel 2000 or later or the SAS Add-In for Microsoft Office. Unlike the ExcelXP destination, MSOffice2K enables you to include images or graphics. It also generates presentation-quality output. The HTML format that is generated with the SAS Add-in to Microsoft Office uses this destination to generate HTML. Microsoft also has numerous style-sheet properties, which are specific to Microsoft, that enable you to customize style

information in a worksheet. You can use these Microsoft style-sheet properties in conjunction with the other methods for specifying styles.

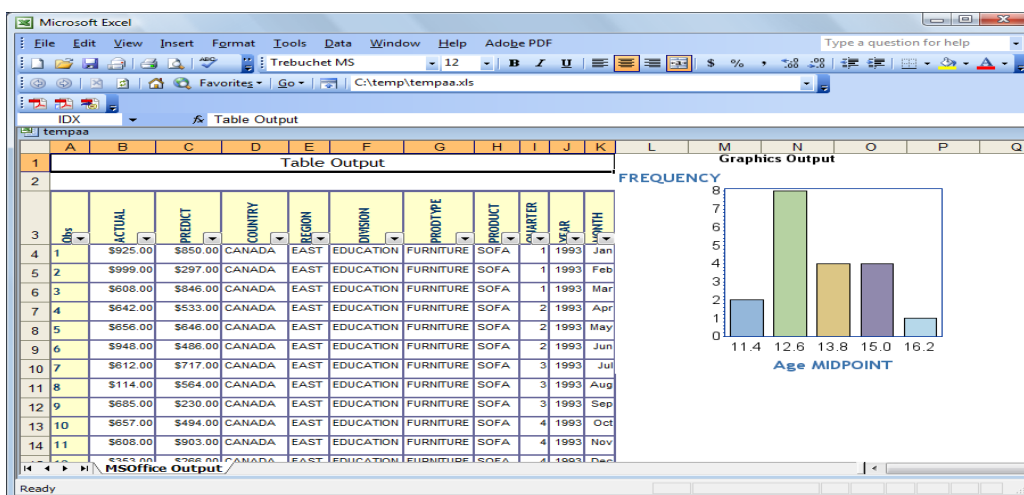
Msoffice2K does not have options created specifically for it as is the case with the ExcelXP destination. However, because this destination is a member of the ODS Markup family, you can add options. Specifying an option in an ODS statement modifies the underlying CSS or XML and produces the desired formatting. The new destination can perform many of the same functions as the ExcelXP destinations. To download the enhanced Msoffice2K tagset, see <http://ftp.sas.com/techsup/download/blind/sgf34.zip>. After you download the tagset, use the DOC="HELP" to see a full list of options that you can use.

Example 27

```
options hsize=3in vsize=3in;
ods tagsets.msoffice2k_x path="c:\temp" file="temp.xls"
    options(rotate_headers="90" rotate_headers="90" panelcols="2"
        autofilter="yes" sheet_name="Msoffice Output" doc="help")
        style=normal;
proc print data=sashelp.prdsale;
    title "Table Output";
run;

proc gchart data=sashelp.class;
    vbar age / patternid=midpoint;
    title h=10pt "Graphics Output";
run;
quit;

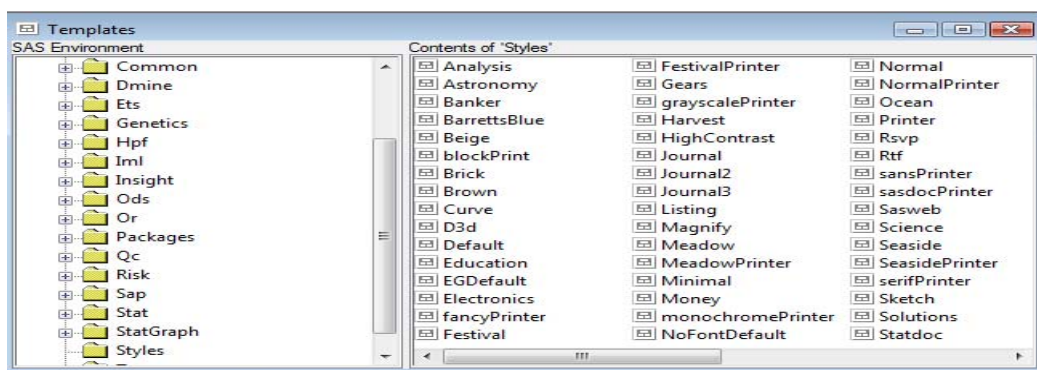
ods _all_ close;
```



Display 19. Output generated with the Msoffice2K destination

STYLES

For output that is generated with the ExcelXP and the Msoffice2K destination, the default style for SAS is applied if no other style is specified. The default style displays a gray background for the cell and header values. You can modify this style by selecting one of the other styles that is shipped with the SAS® System. To get a list of the available styles, submit PROC TEMPLATE with the LIST STYLES statement. You can also view the styles from the TEMPLATE window by specifying ODSTEMP on the command line. This list displays all styles shipped with the SAS software; however, not all the styles in this list present well in Excel for a variety of reasons. For example, some of the fonts in certain styles do not present well in Excel. In other cases, colors do not display correctly. Before Excel 2007, Excel supported a 56-color palette. With Excel 2007, that number has been increased to over 2 million colors. Some of the styles that look better in Excel are specified in ftp.sas.com/techsup/download/base/sgf34.zip along with other downloadable style templates. Refer to this download to see how these styles look in Excel.



Display 20. SAS styles that work well in Excel

CONCLUSION

Using the ODS menu of destinations and features, you can generate customized output for all applications and reporting needs. Whether you are generating a customized table of contents, arranging output, creating Web applications, enhancing styles, or exporting data to Microsoft Office applications, ODS has more than enough features to satisfy any appetite.

REFERENCES

Parker, Chevell. 2007. "Creating a Data Grid Like VB.NET" (includes the TableEditor tagset download). Cary, NC: SAS Institute Inc. Available at support.sas.com/rnd/base/ods/odsmarkup/tableeditor/index.html.

Parker, Chevell. 2009. "Object-Oriented DATA Step" (full program download). Cary, NC: SAS Institute Inc. Available at ftp.sas.com/techsup/download/blind/sgf34.zip.

RECOMMENDED READING

O'Connor, Daniel. 2003. "Next Generation Data _NULL_ Report Writing Using ODS OO Features". Cary, NC: SAS Institute Inc. Available at www2.sas.com/proceedings/sugi28/022-28.pdf.

Parker, Chevell. 2008. "Creating That Perfect Data Grid Using the SAS® Output Delivery System". Cary, NC: SAS Institute Inc. Available at www2.sas.com/proceedings/forum2008/258-2008.pdf.

SAS Institute Inc. 2008 "ODS MARKUP Resources". Cary, NC: SAS Institute Inc. Available at support.sas.com/rnd/base/ods/odsmarkup/.

SAS Institute Inc. 2009. "Base SAS: SAS Notes and Concepts for ODS". Cary, NC: SAS Institute Inc. Available at support.sas.com/rnd/base/ods/templateFAQ/Template_csstyle.html.

ACKNOWLEDGEMENTS

I would like to thank Bari Lawhorn and Susan Berry for contributions to this paper. I would also like to thank the entire ODS development team for continuing to create great software.

CONTACT INFORMATION

Chevell Parker
SAS Institute
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8008
Fax: (919) 531-9449
E-mail: support@sas.com
Web: support.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.