Paper 56

Twisty Little Passages, All Alike – ODS Templates Exposed

Chris Olinger, SAS Institute Inc., Cary, NC

Abstract

The Output Delivery System in Version 7 of the SAS® System provides a powerful feature set for customizing SAS output. This paper talks about the twisty capabilities of the TEMPLATE procedure for changing the layout of procedure output and for creating your own custom corporate styles. Covered are techniques for editing existing ODS templates, for binding templates to data sets, and for manipulating ODS styles. The concepts covered in this talk can be applied to your SAS jobs to create outstanding looking HTML (and later in Version 8, RTF, Postscript, and PDF).

Introduction

You are standing at the end of a road before a small brick building. Around you is a forest. A small stream flows out of the building and down a gully.
`> down`

If you are reading this paper then you are probably a SAS adventurer - the type of person who wants to discover and figure out how you can squeeze every last option out of the SAS programming language. This paper is inspired by the first text-parser adventure game, The Colossal Cave Adventure, written way back in the mid-seventies for the PDP-10. All I could think of when asked to write this paper was the Colossal Cave puzzle: "*You are in a maze of twisty little passages, all alike.*" It was a confusing puzzle, but once you figured out how to get around it, you were home free. This is a lot like PROC TEMPLATE.

PROC TEMPLATE is not for everybody. It has a myriad of options to control the look and feel of output produced by the SAS procedures – in fact, a few of the options are used only by one or two procedures. If you master the syntax, then it is safe to say that you are the undisputed output adventurer at your site.

This paper covers several topics:

- what a template is
- editing existing templates in the system
- creating your own table template to bind with a data set
- creating a style template

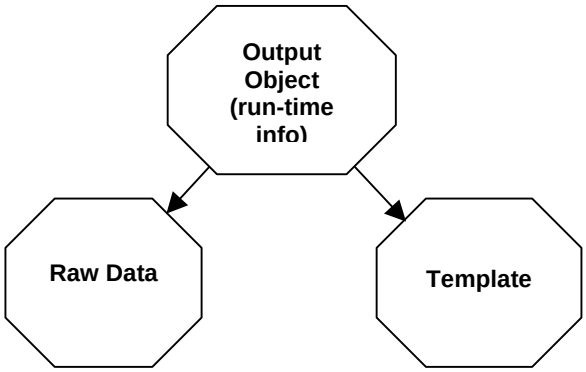
Template Overview

A little dwarf just walked around a corner, saw you, threw a little axe at you, which missed, cursed, and ran away.
`> get axe`

Most ODS output objects contain two basic components:

- a template component
- and a data component containing the raw data values that make up the piece of output

The output object also contains miscellaneous run-time information about the current running procedure. Represented visually, an output object looks like this:



The template portion of the output object is the only part of an output object that you can directly modify. Most output in the system references a template. However, some pieces of output in the system do not. Most notably, line printer oriented output (like that produced by the PLOT and CHART procedures) does not reference a template. PUT statements in a DATA _NULL_ step also do not reference a template. Other exceptions are the REPORT, TABLUATE, PRINT, and FREQ[1] procedures. The syntax for each of these procedures acts as a custom template so an ODS template is not needed.

What exactly is a template? A template is a set of SAS statements and attributes that describes how ODS should format and present raw data. You create and modify a template using PROC TEMPLATE. PROC TEMPLATE also provides statements to delete templates, list templates, and dump the SAS source

[1] PROC FREQ multi-way tables do not use a template. However, the rest of the tables do.

statements that make up a given template. You can find out more about the syntax for PROC TEMPLATE in *The Complete Guide to the SAS Output Delivery System.*[2]

A template is compiled and then stored in a file called a *template store*. A template store is a special version of an *item store*. An item store is a new SAS file that allows multiple levels of directories to reside inside, much like a PC file system stored within a single file. A template store is just an item store with some extra information thrown in so that ODS can recognize the contents as templates.

Version 7 and Version 8 enable you to create six different types of templates:

- Table
- Column
- Header
- Footer
- Tree
- Style

The Table template is a collection of attributes and statements that describe a particular table, and it is the most widely used template. The Column, Header, and Footer templates are used mostly in conjunction with the Table template. The Tree template is used for rudimentary equations and functions, and the Style template is used for controlling the overall look and feel of the entire SAS job.

To create a template, you need to run PROC TEMPLATE, specifying the DEFINE statement. This statement allocates the template and then parses the template's statements up to the corresponding END statement. When the parsing is complete, the template is written to the specified directory. For instance,

```
proc template;
   define table mytables.table1;
   … statements …
   end;
run;
```

This example creates a table template and stores it in *mytables.table1. mytables* is a directory, and *table1* is the actual template name. You can create as many subdirectories as you want by using more periods. By default, the template store that the template gets written to is the first template store that is updateable in the *template path*. The template path is controlled by the ODS PATH statement. The default template path is

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

In the example above, the template would be written to the template store SASUSER.TEMPLAT. For added flexibility, in Version 8 there is an option on the DEFINE statement to write directly to a specific template store, bypassing the current ODS template path.

As an example, consider the following table template from the STANDARD procedure:

```
proc template;

define table Base.Standard;
   column name mean std n label;
   define name;
     header = 'Name';
   end;
   define mean;
     header = 'Mean';
     format = D12.;
   end;
   define std;
     header=’/Standard/Deviation';
     format = D12.;
   end;
   define n;
     header = 'N';
     format = best.;
   end;
   define label;
     header = 'Label';
   end;
   required_space = 3;
   byline;
   wrap;
end;

run;
```

This template defines five columns. The COLUMN statement specifies the ordering of the columns. Each column has a definition declared via an embedded DEFINE/END statement block. Each column has a header and a format specified. Additionally, there are three table attributes defined, REQUIRED_SPACE, BYLINE, and WRAP.

[2] The documentation for PROC TEMPLATE was not finalized in time for the Version 7 ship date. The documentation for PROC TEMPLATE is now posted at the BASE/SAS web site: http://www.sas.com/rnd/base/index-early-access.html).

Modifying a System Template

A huge, green, fierce dragon bars the way! The dragon is sprawled out on a Persian rug!
`> kill dragon`

With your bare hands?
`> yes`

Congratulations! You have just vanquished a dragon with your bare hands! (Unbelievable, isn't it?)

Modifying a template is not hard. Once you know the secret it's easy. As an example, you are going to take the OverAllAnova table from GLM procedure and reformat it so that you can fit it inside this two-column paper format.

The following code generates a data set and produces an OverAllAnova table in RTF[3] form.

```
data a;
  do a = 1 to 4;
    do b = 1 to 4;
      do c = 1 to 4;
        n = int(6*ranuni(1));
        do i = 1 to n;
           y = a + b + rannor(1);
           output; end;
        end;
      end;
    end;

ods rtf file="glm.rtf";
ods select OverallANOVA;

proc glm data=a;
  class a b c;
  model y=a|b|c / ss1 ss2 ss3 ss4;
quit;

ods rtf close;
```

This code produces the following table. Unfortunately, the table is too wide for this paper format, so after selecting table AutoFormat in Word, you get the following table (your supervising editor is very displeased):

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	**56**	**492.5623493**	**8.79575562**	**10.47**	**<.0001**
Error	**105**	**88.17570 08**	**0.83976 86**		
Corrected Total	**161**	**580.7380501**			

There are just too many columns to fit in this space. What you would like to do is change how the columns are presented in the table so that it comes out correct the first time.

The first step in fixing this output is to discover the name of the template being used. The names of some of the procedure's output objects and templates are documented in the procedure guides, but an easier way to get the information is to use the ODS TRACE OUTPUT command. The TRACE OUTPUT command tells ODS to print a message to the SAS log each time a piece of output is created. The trace record contains the name of the output and the template that is associated with it.

```
ods trace output;
```

When you run the example again with TRACE OUTPUT turned on, you get the following in the SAS log:

```
Output Added:
-------------
Name:     OverallANOVA
Label:    Overall ANOVA
Template: stat.GLM.OverallANOVA
Path:     GLM.ANOVA.y.OverallANOVA
-------------
```

Here, notice that the template for the OverAllAnova table is Stat.Glm.OverAllAnova. Remember, the periods indicate directories, so the OverAllAnova template lives in the GLM directory, beneath the Stat directory, in a template store somewhere. The actual template store that the template is contained in does not matter, as long as it exists in the concatenation path.

[3] RTF is an experimental output destination in Version 7 and Version 8. Use at your own risk.

There are two ways that you can get the SAS source of this template. The first is to use the SOURCE statement in PROC TEMPLATE.

```
proc template;
   source Stat.GLM.OverallAnova;
run;
```

This statement dumps the SAS source to the log. Or, you can dump the source directly to a file so you can edit it.

```
proc template;
   source Stat.GLM.OverallAnova /
      file="overall.sas";
run;
```

You can also view the template in DMS mode by selecting View->Results. Right-Mouse-Button (RMB) down on "Results" and then select "Templates". This will bring up an explorer window on the templates in the system. Double click SASHELP.TMPLMST, select Stat, select GLM, select OverallAnova. You can RMB down on OverallAnova and select "Edit" if you prefer. This enables you to view the text of the template in edit mode rather than browse mode. After you are done editing, you can select Run->Submit to run the code.

The actual code of the OverallANOVA template is as follows:

```
define table Stat.GLM.Overallanova;
   notes 'Over-all ANOVA';
   parent = Stat.GLM.ANOVA;
   top_space = 0;
   double_space;
end;
```

There is not much to this template. It turns out that this template is *parented* to another template. The PARENT= option declares that the OverallANOVA template will inherit all the values of Stat.GLM.ANOVA, overriding a select few attributes that apply only to the ODS LISTING destination. Again, using the SOURCE statement, you get the following SAS source for Stat.GLM.ANOVA:

```
define table Stat.GLM.ANOVA;
   notes 'Parent for GLM ANOVA tables';
   column Source DF SS MS FValue ProbF;
   define Source;
      parent=Stat.GLM.Source;
   end;
   define DF;
      parent=Stat.GLM.DF;
   end;
   define SS;
      parent=Stat.GLM.SS;
   end;
   define MS;
      parent=Stat.GLM.MS;
   end;
   define FValue;
      parent=Stat.GLM.FValue;
   end;
   define ProbF;
      parent=Stat.GLM.ProbF;
   end;
end;
```

For PROC GLM, all ANOVA tables share this common parent that defines the column order and specifies the basic column definitions (which are also parented to common column definitions). There are two ways that you can change this source code to get the results that you want. The first example hand copies the values from the common parent into Stat.GLM.OverallANOVA, overriding what you need. The template is stored in the first updateable template store in the concatenation path.

```
proc template;

define table Stat.GLM.OverallANOVA;
 parent=stat.glm.anova;
 column source df (ss ms) (fvalue probf);
 define source;
   parent=Common.ANOVA.source;
   format=$16.;
 end;
 define df;
   parent=Common.ANOVA.df;
   format=4.0;
 end;
 define ss;
   parent=Common.ANOVA.ss;
   header="#SS#MS";
   format=7.2;
 end;
 define ms;
   parent=Common.ANOVA.ms;
   format=7.2;
 end;
```

```
 define fvalue;
   parent=Common.ANOVA.fvalue;
   header="#F#p>F";
 format=7.2; end;
 define probf;
   parent=Common.ANOVA.probf;
   format=pvalue5.2;
 end;
end;

run;

ods rtf file="glm.rtf";

ods select overallANOVA;
proc glm data=a;
  class a b c;
  model y = a|b|c / ss1 ss2 ss3 ss4;
run;

ods rtf close;

proc template;
  delete stat.glm.overallAnova;
run;
```

The second example uses the EDIT statement to explicitly copy the template, modifying only the values that you want to change. The template is stored in the first updateable template store in the concatenation path.

```
proc template;

edit Stat.GLM.OverallANOVA;
 column source df (ss ms) (fvalue probf);

 edit source;
   format=$16.;
 end;
 edit df;
   format=4.0;
 end;
 edit ms;
   format=7.2;
 end;
 edit probf;
   format=pvalue5.2;
 end;
 edit ss;
   header="#SS#MS";
   format=7.2;
 end;
 edit fvalue;
   header="#F#p>F";
   format=7.2;
 end;
end;

run;

ods rtf file="glm.rtf";

ods select overallANOVA;
proc glm data=a;
   class a b c;
   model y = a|b|c / ss1 ss2 ss3 ss4;
   run;

ods rtf close;

proc template;
   delete stat.glm.overallAnova;
run;
```

Both of these examples use the *column stacking* feature of table templates.

```
Column source df (ss ms) (fvalue probf);
```

When you enclose a group of columns within a set of parentheses, ODS stacks the columns in the group on top of each other. Each field in the grouped column is formatted according to the real column format. The header for the column comes from the first column in the group.

Notice also that you are using new formats to change the relative widths of the columns and that the headers for the two stacked columns contain a '#' as the first character. The '#' is a *split* character that will force a split in the header.

Lastly, notice the DELETE statement at the end of both examples. The DELETE statement removes the template that we just created so that it will not interfere with subsequent runs of PROC GLM.

In either example, the new template produces the following table (after a table AutoFormat in Word):

Source	**DF**	**SS** **MS**	**F** **p>F**
Model	56	492.56 8.80	10.47 <.01
Error	105	88.18 0.84	
Corrected Total	161	580.74	

Binding a Template to a SAS Data Set

You are at a complex junction. A low hands and knees passage from the north joins a higher crawl from the east to make a walking passage going west. There is also a large room above. The air is damp here.
> north

If you have a table that you are producing with PUT statements in a DATA _NULL_ step, you can instead bind a template to the DATA step to produce the table in a more regular fashion. There are some unique features of the table template that can be leveraged to produce a nice looking tabular report. However, most of the features of the table template are redundant with the REPORT procedure, so we are not suggesting that you run out and convert all of your PROC REPORT code. Use the procedure that best suits your needs.

The following example shows how to bind a template to a DATA step. For more information on how to use the DATA step to interface with ODS, please refer to Heffner (1998) and the documentation.

```
data _null_;
   set mydata;
   file print
   ods=(template='mytemplates.template');
   put _ods_;
run;
```

The *file print* statement takes a template name that the DATA step uses to render its output. The template column names should match the variable names in the data set. The *put _ods_* statement moves the data from the data set to the underlying ODS data component.

The next three examples produce a budget analysis report in HTML. The report consists of a table or set of tables, one for each quarter of the year, that show how well various departments did with respect to their stated budget goals. The input data set is documented fully in **Figure 1** in the appendix.

The first example is a simple table. It uses the TEST statement to test out the template. The TEST statement binds the last template created to the specified data set.

```
data budget;
   input QTR @8 Dept $10. @22
         Account $8. Budget Actual;
   attrib Actual format=dollar11.2
          Budget format=dollar11.2
          Dept   format=$10.
          QTR    format=best4.
          Diff   format=dollar11.2;
   diff = actual - budget;
   datalines;
...
;
proc sort data = budget;
   by qtr dept;
run;

ods listing close;
ods html file="budget.htm";
proc template;
   define table mytables.budget;
      column qtr dept account
             budget actual diff;
      use_name;
   end;
   test data=budget;
run;
ods html close;
```

The ODS LISTING statement closes the SAS listing. The ODS HTML statement opens an HTML file called budget.htm. This example creates a template called *mytables.budget.* The template will be stored in the first updateable template store in the template path.

The template contains a column statement that defines the columns of the table. The names of the columns must match the variable names of the input data set! If they do not match (and you do not use the column *dataname=* attribute) then you will not get any output. Only the template columns that are paired with data are printed.

The *use_name* attribute tells the template to use the name of the variable in the data component as a header for the column if there isn't a header already defined in the template and there isn't a label defined on the data component column.

The resultant output can be seen in **Figure 2** in the Appendix.

You have just created a table that looks like a PROC PRINT listing. You can take this table a step further by using the BLANK_DUPS and CLASSLEVEL attributes. These two options blank out duplicate values in vertically adjacent column cells (the MEANS procedure achieves this effect in the same manner).

```
ods html file="budget2.htm";

proc template;

  define table mytables.budget2;
      column qtr dept account budget
             actual diff;
      classlevels;
      use_name;
      define qtr;
         blank_dups;
      end;
      define dept;
         blank_dups;
      end;
   end;

   test data=budget;
run;

ods html close;
```

The output from this example can be seen in **Figure 3** in the Appendix.

Notice that the column statement defines six columns. However, only two columns, *qtr* and *dept*, actually have definitions specified. Any column that does not have a definition is assigned default values for any attributes that are needed by ODS.

To take this report one step further you can

1. add a spanning header that references the current date and time using a macro variable reference
2. split the table into multiple tables, by quarter, and add a spanning header that links to the next quarter's table
3. add an extra column using the COMPUTE statement
4. add traffic lighting to the report using CELLSTYLE[4] to help track where you had problems.

The following code produces the table in **Figure 4** in the Appendix:

[4] CELLSTYLE and COMPUTE do not work together very well in Version 7, so you will not be able to run this program as specified in Version 7. The bug(s) have been fixed in Version 8.

```
proc template;

  define table mytables.budget3;
    column dept account budget actual
           (pct diff);
    header h1 h2;
    mvar sysdate systime qnum;

    classlevels use_name;

    define h1;
      text "Quarter: " qnum;
      style = header
        { prehtml=symget("href")
          posthtml='</a>'
        };
    end;

    define h2;
      text "Report Created: " sysdate
           " " systime;
      just = l;
      start=dept;
      end=actual;
    end;

    define dept;
      header="Department";
      blank_dups;
    end;

    define pct;
      header="#% Diff#$ Diff";
      compute as (diff / budget);
      format=percent8.2;
      cellstyle
        _val_ >= .50 as data
          { background=red
            flyover="Get a new job!"
          },
        _val_ >= .25 as data
          { background=light red },
        _val_ >= .10 as data
          { background=very light red },
        _val_ <= -.50 as data
          { background=black
            foreground=yellow
            flyover="Great job!"
          },
        _val_ <= -.25 as data
          { background=blue
            foreground=white
          },
        _val_ <= -.10 as data
          { background=very light blue },
        1 as data;
    end;
  end;

run;
```

```
%macro quarters;
 %do qnum=1 %to 4;
  %let next=%eval(&qnum + 1);
  %if &next=5 %then %let next=1;
  %let href=<a href="quarter&next..htm">;
  data _null_;
    set budget(where=(qtr=&qnum));
    file print
      ods=(template='mytables.budget3');
    put _ods_;
  run;
 %end;
%mend;

ods html file="quarter1.htm"
         newfile=table style=d3d;
%quarters;
ods html close;
```

Looking at the code, the first thing that you notice is that a new PCT column is added to the column statement and that PCT is stacked on top of DIFF. PCT is a *computed* column. A computed column is a column that modifies or creates its value with the COMPUTE statement. The DEFINE/END block for PCT specifies that the value for the column is `(diff / budget)`. You can also use DATA step functions in the COMPUTE statement for more complex calculations. The COMPUTE statement uses the WHERE clause processor to evaluate its expressions, so anything that you can do in a WHERE statement you can do in a COMPUTE statement as well.

There are also two new headers in the table. H1 is defined as the string `"Quarter: " QNUM.` QNUM is declared on the MVAR statement. The MVAR statement declares variables that are macro references to be resolved when the template is used. Do not confuse this with actually using a macro variable! If you specify &QNUM then the macro variable is resolved at template compile time. Using a macro variable with the MVAR statement enables you to change the value of the variable each time the template is used. This same trick is used on the header H2, where you define a header containing the system date and the system time that the table was created. If you used "&sysdate" and "&systime" you would wind up with a header that specified the date and time that the template was created.

Also, note the STYLE attribute on H1. The style for the header is defined as `style = header { prehtml = symget("href") posthtml = '</a>' }.` What this is saying is that the style for the header is the system HEADER element, but the PREHTML and POSTHTML style attributes are being overridden. PREHTML writes the text associated with the attribute before the contents of the HTML cell. POSTHTML writes the text associated with the attribute after the contents of the HTML cell. This code effectively enables you to turn the header into an HTML link to another table. PREHTML is being set to a macro reference, symget("href"), that will be resolved when the template is used. SYMGET is a little different than using the MVAR statement, but the results are the same. Use SYMGET when defining style attributes; use MVAR when defining table templates.

If there is no STYLE attribute for a header or column then the following defaults are used,

Cell Type	***Normal***	***Preformatted***
Column	Data	DataFixed
Column Header	Header	HeaderFixed
Footer	Footer	FooterFixed
Header	Header	HeaderFixed
Table	Table	

The last thing to notice about this template is the CELLSTYLE-AS statement on the PCT column. CELLSTYLE-AS sets the style of the data cell based on the value of the cell. _VAL_ is a special tag that references the current value of the column. You can string multiple expressions together with commas to form an if/then/else clause. The first expression that evaluates to TRUE specifies the style for the cell. The trailing `1 as data;` is a catch-all that sets the style of the column cell to the system default DATA if all of the previous expressions fail. Unlike with the COMPUTE statement, the CELLSTYLE-AS statement does not let you reference other columns in the expression. This is a limitation that will be dealt with in later releases of the software. However, you can reference DATA step functions.

The tables are created using a macro, %quarter, that sets the HREF and QNUM macro variables in a loop (1-4). The H1 header links to the next quarter's table. The fourth quarter's header links to the first quarter's table. The NEWFILE=TABLE attribute on the HTML statement generates a new HTML file for each table that is created. The HTML output destination automatically increments the file number.

Understanding ODS Styles

You are in a maze of twisty little passages, all alike.
```
> north
```

You are in a maze of twisty little passages, all alike.
```
> west
```

You are in a maze of twisty little passages, all alike.
```
> !@#$
```

Watch it! [5]

An ODS style is a collection of presentation attributes that apply to an output destination. These attributes include foreground and background color and fonts. Kelley and McNeill (1999) have this to say about styles:

"*ODS styles govern the overall look and feel of Version 7 SAS procedure output. Styles determine the colors, fonts, graphic images, and other visual aspects in effect when output is generated. SAS delivers several predefined styles with Version 7, and you can customize them or create new ones.*

Exactly one style is in effect at any time during a SAS session. It applies to all procedure output during the period it is in effect. Such output has a uniform look and feel – regardless of the procedure. Row headers have the same background color, data cells have the same font, and so on. Styles apply to all of the ODS output formats that allow colors, proportional fonts, and so forth.

ODS organizes a style as a named collection of style elements*. Specifically,*

- *Style names tell ODS which one of the available styles to apply to the output. If you do not specify a style name, ODS selects the predefined style named* Default*. The selected style remains in effect until you select another one or terminate the SAS session. You can rename or delete styles. (Be warned that if you rename or delete a predefined style, you could render the SAS session inoperable.)*

- *Style elements govern the look and feel of a particular part of the output, such as a PROC REPORT data cell or a PROC TABULATE row header. Style elements have names (*Data, RowHeader*) that are fixed by ODS; style elements cannot be renamed or deleted.*

- *ODS organizes style elements as a collection of name-value pairs called* style attributes*. Each attribute determines a particular aspect of the style element, such as the row header background color (background=green) or data cell font size (font_size=4). Style attribute names are fixed by ODS. The domain of style attribute values depends upon the attribute*."

[5] The game understands certain four letter words that cannot be printed here. Use your imagination.

STYLE Statement

More specifically, a style is a collection of STYLE and REPLACE statements. Each statement defines a style element by listing its style attributes. For instance, a very simple style that contains three style elements is listed below:

```
proc template;
  define style styles.MyFirstStyle;
    style header /
      foreground=black
      background=white
      font=(Arial,3,Italic)
      ;
    style footer /
      foreground=black
      background=white
      font=(Arial,3,Italic)
      ;
    style data /
      foreground=black
      background=pink
      font=(Arial,4)
      ;
  end;
run;
```

This style defines the elements HEADER, FOOTER, and DATA. Any output that uses this style is limited to the elements that have been defined. General SAS output uses a large array of style elements. For a complete list of style elements, refer to *The Complete Guide to the SAS Output Delivery System*. However, this style is enough to handle simple PROC PRINT output.

Each element in this example has a foreground and background color. Style colors can be any valid SAS/GRAPH® color or a #RRGGBB value. Fonts are specified as a tuple: (*FontNameList, Size, Attributes*). *FontNameLis*t is the name of the font(s) that you want to use. If there is more than one (or if a font name contains embedded spaces), then enclose the font list in quotes and separate the font names with commas. The first font in the list that is recognized by the output destination is used. *Attributes* is a list of modifiers like *Italic* and **Bold.**

There are approximately sixty-five different style attributes that can be specified on a style element. For a complete list, refer to the documentation.

If you look closely at this style, you will notice that the Header and Footer elements are exactly the same. You

can abstract common information to another element using *style element inheritance.* Style element inheritance is achieved using the FROM keyword on the STYLE statement. For instance,

```
proc template;
  define style styles.MyFirstStyle;
    style cell /
      foreground=black
      background=white
      font=(Arial,3,Italic)
      ;
    style header from cell
      ;
    style footer from cell
      ;
    style data from cell /
      background=pink
      font=(Arial,4)
      ;
  end;
run;
```

Here you add a new abstract element named Cell. The style elements Header, Footer, and Data inherit from Cell. The Data element overrides background color and font. Using style element inheritance enables you to build one style element based on some other style element.

You can further abstract this style using *style references.* Often, it is desirable to use a reference to a common value so that all attributes with the same value can be easily changed. For instance,

```
proc template;
  define style styles.MyFirstStyle;
    style ref /
      "cellfg"   = black
      "cellbg"   = white
      "databg"   = pink
      "cellfont" = (Arial,3,Italic)
      "datafont" = (Arial,4)
      ;
    style cell /
      foreground=ref("cellfg")
      background=ref("cellbg")
      font=ref("cellfont")
      ;
    style header from cell;
    style footer from cell;
    style data from cell /
      background=ref("databg")
      font=ref("datafont")
      ;
  end;
run;
```

Notice how the style element REF is composed of attributes whose labels are enclosed in quotes. Any attribute name enclosed in quotes is called *a user-defined attribute*. These attributes can be used by other attributes to group common elements. If you want to change all occurrences of the color black to the color blue, all you would need to do is change the reference color instead of searching through your style for all instances of the color black.

All references are resolved the first time a style is used. If a reference cannot be resolved, then a warning is printed to the log and the value of the attribute remains unset. You can use the special reference value SYMGET to reference a macro variable directly:

```
proc template;
...
style cell / background=symget("bg");
...
run;

%let bg=red;

... sascode ...
```

The macro reference is resolved *each time the style element is used* so the value can change between uses. This is a very powerful technique, but it can be costly performance-wise due to the potential number of times the macro variable may be queried.

In Version 8 there is also the special value, _UNDEF_. If an attribute is marked _UNDEF_ then the value of the style attribute is considered not set, even if an inherited parent has declared the value specifically.

```
style data from cell /
  background=_undef_;
```

REPLACE Statement

The style in the previous examples is a good example of how to build a standalone style. But what if you want to keep your master style the way it is *and* change it subtly? For instance, if you wanted to keep the structure of the master style (for example, the inheritance path, the style element names), and if you wanted to change just the colors, how would you go about it? There are two ways. The brute-force method would be to copy the source using the SOURCE statement in PROC TEMPLATE, change the values that you wanted to change, and then recompile it using another name. This method works great for small styles but isn't so wonderful for larger ones. The second, more elegant, solution is to use *style inheritance*. Style inheritance is a little different than style element inheritance using FROM. Style inheritance uses the PARENT= attribute.

```
proc template;
  define style styles.MySecondStyle;
    parent=style.MyFirstStyle;
  end;
run;
```

The above code creates a new style called Styles.MySecondStyle that is an exact copy of your first style. If you wanted to change the Data element to add FLYOVER text, you would just override the element definition.

```
proc template;
  define style styles.MySecondStyle;
    parent=styles.MyFirstStyle;
    style data from data /
      flyover="Data Cell"
    ;
  end;
run;
```

Notice here that Data inherits FROM Data. The parent Data is contained in the parent style, styles.MyFirstStyle. It is illegal to inherit FROM an element of the same name when the element does not exist in a parent inheritance layer. Using FROM in this context appends or overrides any attributes specified in the parent element. Conversely, if you wanted to override the Data style element in its entirety, then you would leave the FROM clause off.

You have successfully added the FLYOVER attribute to the Data element. But what happens when you want to change the Cell element such that the changes to Cell cascade down to both Header and Footer? Changing the Cell element with the STYLE statement *will only affect the Cell element* ! For instance,

```
proc template;
  define style styles.MySecondStyle;
    parent=styles.MyFirstStyle;
    style cell from cell /
      flyover="Cell"
    ;
  end;
run;
```

You have modified Cell, but SAS output does not use Cell for actual rendering, it uses Header and Footer which inherit from Cell. Cell is an abstract definition. What you really want is for the new values in Cell to trickle down to Header and Footer. The way to achieve this effect is by using REPLACE.

```
proc template;
  define style styles.MySecondStyle;
    parent=styles.MyFirstStyle;
    replace cell /
      foreground=ref("cellfg")
      background=ref("cellbg")
      font=ref("cellfont")
      flyover="Cell"
    ;
  end;
run;
```

REPLACE causes the element definition *to appear as if it were replaced in the parent style*. Note that the element definition is replaced in its entirety! This includes any FROM that is specified.[6] Because of this restriction, you must copy any values from the parent that you want kept as part of the element. When the Cell element is replaced in the parent, all elements inheriting from Cell will pick up the new definition.

Currently, there is no way to override part of or append to a style element that you want to replace - you must replace the element completely. Under investigating is an OVERRIDE statement that would act as STYLE does but affect only the parent style element.

A note of warning: using REPLACE can sometimes result in confusing warning messages. Most often these warning messages are the result of replacing a style element that was being used by another style element in the parent style, but forgetting to include all of the attributes that the parent style needs. If the parent style can't find a color, for instance, it will complain. Practice caution when using REPLACE.

Styles.Default

Now that you have mastered styles, take a look at Styles.Default and Styles.D3D.

```
proc template;
  source styles.default;
  source styles.D3D;
run;
```

Styles.Default is the master style in the system. The elements defined by this style are used by all of the procedures that generate ODS output. To create a new style, you should PARENT= off of Styles.Default and modify the elements that give you the desired effect. As

[6] In fact, you can use REPLACE to change the inheritance path of a style element. Just change the FROM value and you have modified the parent style element to inherit from a different element.

an example, consider Styles.D3D. This style mostly overrides colors, but it also does some replacing of system style elements as well.

A good example of producing a corporate style can be found at: http://www.sas.com/rnd/base/topics/style-template/style.html. This paper was written by the infamous Paul Kent and shows you how to take an existing background image (your company's logo perhaps) and extrapolate it into a full-blown style. The best way to learn about styles is to try one so you are encouraged to check this example out. You can take a look at some of the output that uses this style in **Figure 5** in the Appendix.

Conclusion

There is a loud explosion, and a twenty-foot hole appears in the far wall, burying the dwarves in the rubble. You march through the hole and find yourself in the main office, where a cheering band of friendly elves carry the conquering adventurer off into the sunset.

PROC TEMPLATE is a powerful tool for manipulating SAS output. With it, you can

- modify any procedure output that uses a template
- create stand alone reports by binding a template to a data step
- create custom styles for jazzing up your presentations and reports
- achieve output enlightenment

We look forward to hearing from you at future conferences.

XYZZY.

Contact Information

The SAS Institute elves that bring you ODS are very excited about the potential of these new capabilities of the SAS System. You can send electronic mail to ods@sas.com with your comments, or if you prefer, to beg for a map. You can also contact the bearded pirate directly at sascro@sas.com

We will be adding more information about ODS capabilities to the Research and Development section of SAS Institute's Web Page. See http://www.sas.com/rnd/base/

References

Kelley, D. W. and McNeill S. L. (1999), "Getting Stylish with Version 7 Base Reporting," in the *Proceedings for the Twenty-Fourth Annual SAS Users Group International Conference,* Cary, NC: SAS Institute Inc.

Heffner, W. F. (1998), "ODS: The Data Step Knows," in the *Proceedings for the Twenty-Third Annual SAS Users Group International Conference,* Cary, NC: SAS Institute Inc.

If you are interested in playing the Colossal Cave Adventure, see http://www-tjw.stanford.edu/adventure/. General information about the game can be found at http://people.delphi.com/rickadams/adventure/index.html

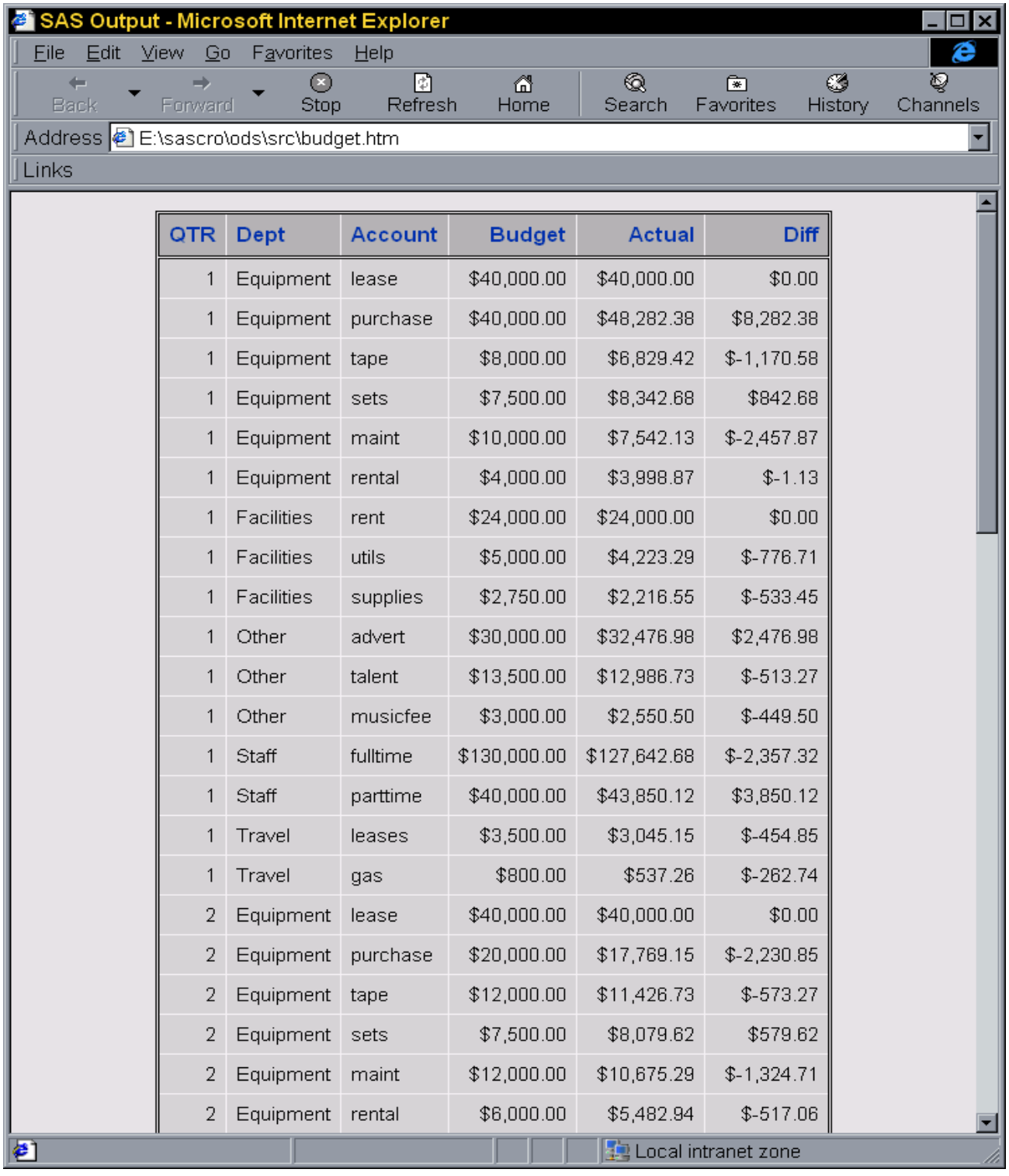
Appendix

Figure 1:

```
data budget;
   input QTR @8 Dept $10. @22 Account $8. Budget Actual;
   attrib Actual format=dollar11.2
          Budget format=dollar11.2
          Dept   format=$10.
          QTR    format=best4.
          Diff format=dollar11.2;
diff = actual - budget;
datalines;
 1     Staff         fulltime       130000.00       127642.68
 2     Staff         fulltime       165000.00       166345.75
 1     Staff         parttime        40000.00        43850.12
 2     Staff         parttime        60000.00        56018.96
 1     Equipment     lease           40000.00        40000.00
 2     Equipment     lease           40000.00        40000.00
 1     Equipment     purchase        40000.00        48282.38
 2     Equipment     purchase        20000.00        17769.15
 1     Equipment     tape             8000.00         6829.42
 2     Equipment     tape            12000.00        11426.73
 1     Equipment     sets             7500.00         8342.68
 2     Equipment     sets             7500.00         8079.62
 1     Equipment     maint           10000.00         7542.13
 3     Staff         fulltime       130000.00       127642.68
 4     Staff         fulltime       165000.00       166345.75
 3     Staff         parttime        40000.00        43850.12
 4     Staff         parttime        60000.00        56018.96
 3     Equipment     lease           40000.00        40000.00
 4     Equipment     lease           40000.00        40000.00
 3     Equipment     purchase        40000.00        48282.38
 4     Equipment     purchase        40000.00        17769.15
 3     Equipment     tape             8000.00         6829.42
 4     Equipment     tape            12000.00        11426.73
 3     Equipment     sets             7500.00         8342.68
 4     Equipment     sets             7500.00         8079.62
 3     Equipment     maint           10000.00         7542.13
 4     Equipment     maint           12000.00        10675.29
 3     Equipment     rental           4000.00         3998.87
 4     Equipment     rental           6000.00         5482.94
 3     Facilities    rent            24000.00        24000.00
 4     Facilities    rent            24000.00        24000.00
 3     Facilities    utils            5000.00         4223.29
 2     Equipment     maint           12000.00        10675.29
 1     Equipment     rental           4000.00         3998.87
 2     Equipment     rental           6000.00         5482.94
 1     Facilities    rent            24000.00        24000.00
 2     Facilities    rent            24000.00        24000.00
 1     Facilities    utils            5000.00         4223.29
 2     Facilities    utils            3500.00         3444.81
 1     Facilities    supplies         2750.00         2216.55
 2     Facilities    supplies         2750.00         2742.48
 1     Travel        leases           3500.00         3045.15
 2     Travel        leases           4500.00         3889.65
 1     Travel        gas               800.00          537.26
 2     Travel        gas              1200.00          984.93
```

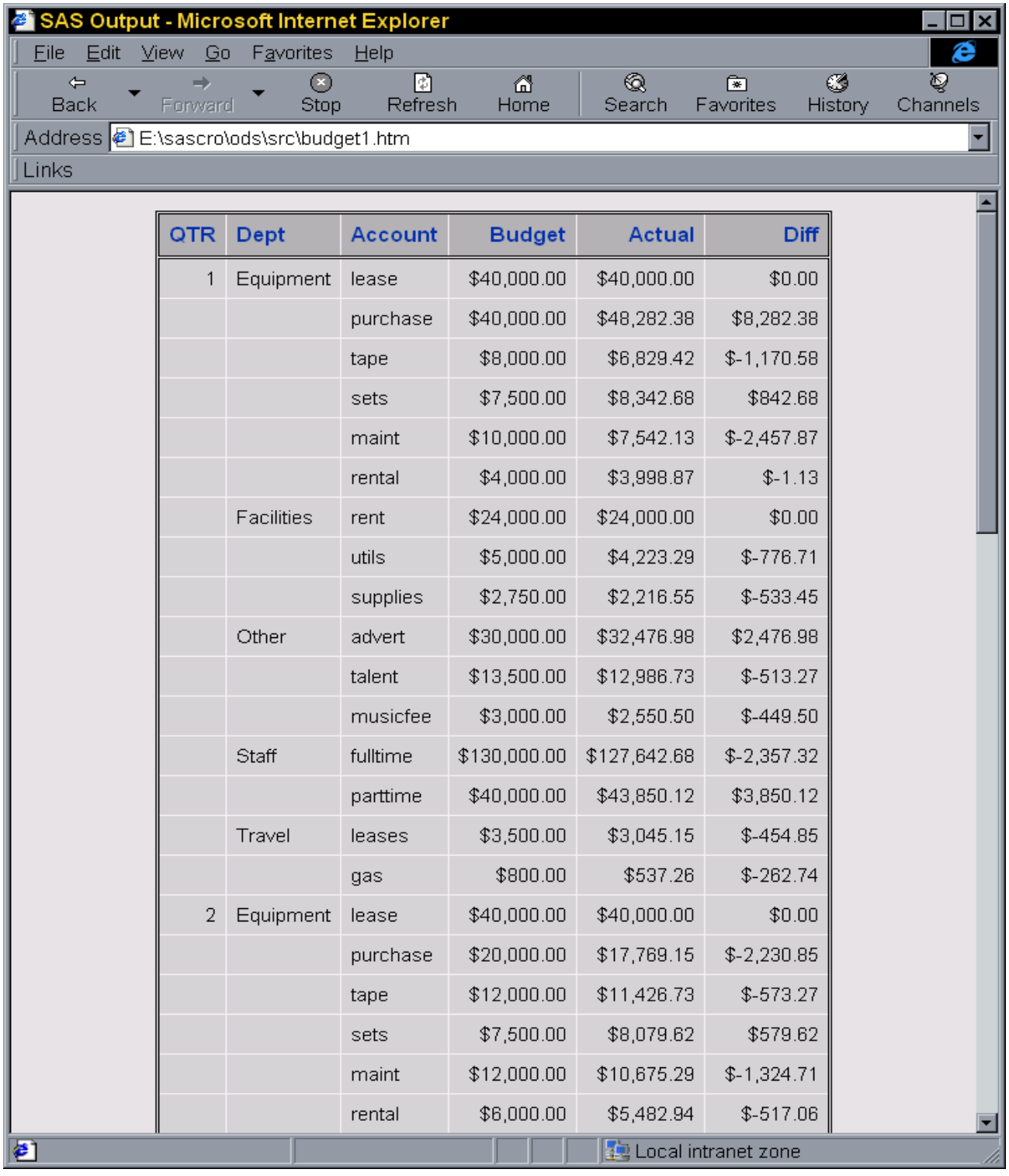
```
 1     Other         advert          30000.00        32476.98
 2     Other         advert          30000.00        37325.64
 1     Other         talent          13500.00        12986.73
 2     Other         talent          19500.00        18424.64
 1     Other         musicfee         3000.00         2550.50
 2     Other         musicfee         5000.00         4875.95
 4     Facilities    utils            3500.00         3444.81
 3     Facilities    supplies         2750.00         2216.55
 4     Facilities    supplies         2750.00         2742.48
 3     Travel        leases           3500.00         3045.15
 4     Travel        leases           4500.00         3889.65
 3     Travel        gas               800.00          537.26
 4     Travel        gas              1200.00          984.93
 3     Other         advert          30000.00        32476.98
 4     Other         advert          30000.00        39325.64
 3     Other         talent          13500.00        12986.73
 4     Other         talent          19500.00        39424.64
 3     Other         musicfee         3000.00         2550.50
 4     Other         musicfee         5000.00         4875.95
;
run;
```

Figure 2:



QTR	Dept	Account	Budget	Actual	Diff
1	Equipment	lease	$40,000.00	$40,000.00	$0.00
1	Equipment	purchase	$40,000.00	$48,282.38	$8,282.38
1	Equipment	tape	$8,000.00	$6,829.42	$-1,170.58
1	Equipment	sets	$7,500.00	$8,342.68	$842.68
1	Equipment	maint	$10,000.00	$7,542.13	$-2,457.87
1	Equipment	rental	$4,000.00	$3,998.87	$-1.13
1	Facilities	rent	$24,000.00	$24,000.00	$0.00
1	Facilities	utils	$5,000.00	$4,223.29	$-776.71
1	Facilities	supplies	$2,750.00	$2,216.55	$-533.45
1	Other	advert	$30,000.00	$32,476.98	$2,476.98
1	Other	talent	$13,500.00	$12,986.73	$-513.27
1	Other	musicfee	$3,000.00	$2,550.50	$-449.50
1	Staff	fulltime	$130,000.00	$127,642.68	$-2,357.32
1	Staff	parttime	$40,000.00	$43,850.12	$3,850.12
1	Travel	leases	$3,500.00	$3,045.15	$-454.85
1	Travel	gas	$800.00	$537.26	$-262.74
2	Equipment	lease	$40,000.00	$40,000.00	$0.00
2	Equipment	purchase	$20,000.00	$17,769.15	$-2,230.85
2	Equipment	tape	$12,000.00	$11,426.73	$-573.27
2	Equipment	sets	$7,500.00	$8,079.62	$579.62
2	Equipment	maint	$12,000.00	$10,675.29	$-1,324.71
2	Equipment	rental	$6,000.00	$5,482.94	$-517.06

Figure 3:



QTR	Dept	Account	Budget	Actual	Diff
1	Equipment	lease	$40,000.00	$40,000.00	$0.00
		purchase	$40,000.00	$48,282.38	$8,282.38
		tape	$8,000.00	$6,829.42	$-1,170.58
		sets	$7,500.00	$8,342.68	$842.68
		maint	$10,000.00	$7,542.13	$-2,457.87
		rental	$4,000.00	$3,998.87	$-1.13
	Facilities	rent	$24,000.00	$24,000.00	$0.00
		utils	$5,000.00	$4,223.29	$-776.71
		supplies	$2,750.00	$2,216.55	$-533.45
	Other	advert	$30,000.00	$32,476.98	$2,476.98
		talent	$13,500.00	$12,986.73	$-513.27
		musicfee	$3,000.00	$2,550.50	$-449.50
	Staff	fulltime	$130,000.00	$127,642.68	$-2,357.32
		parttime	$40,000.00	$43,850.12	$3,850.12
	Travel	leases	$3,500.00	$3,045.15	$-454.85
		gas	$800.00	$537.26	$-262.74
2	Equipment	lease	$40,000.00	$40,000.00	$0.00
		purchase	$20,000.00	$17,769.15	$-2,230.85
		tape	$12,000.00	$11,426.73	$-573.27
		sets	$7,500.00	$8,079.62	$579.62
		maint	$12,000.00	$10,675.29	$-1,324.71
		rental	$6,000.00	$5,482.94	$-517.06

Figure 4:

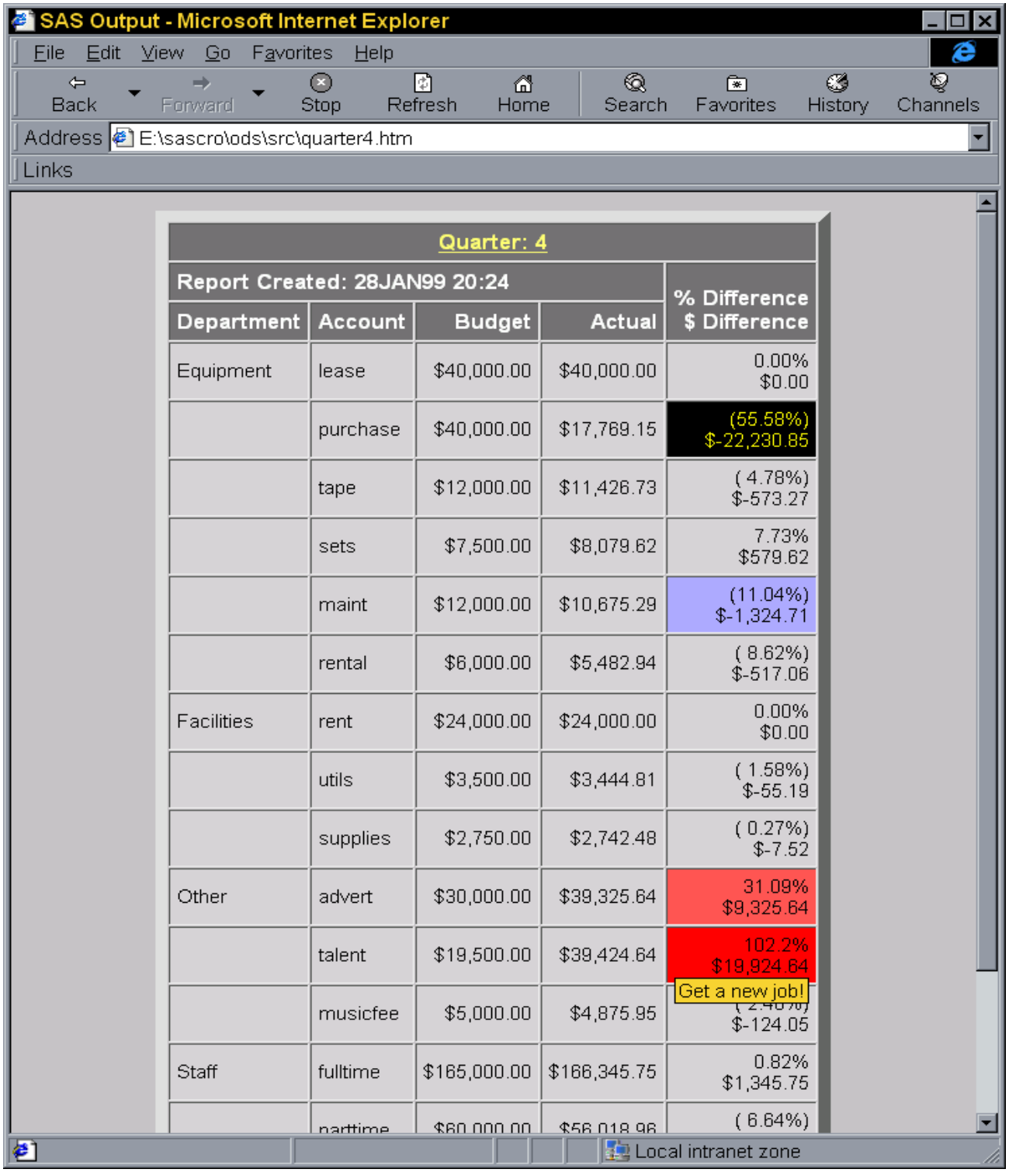
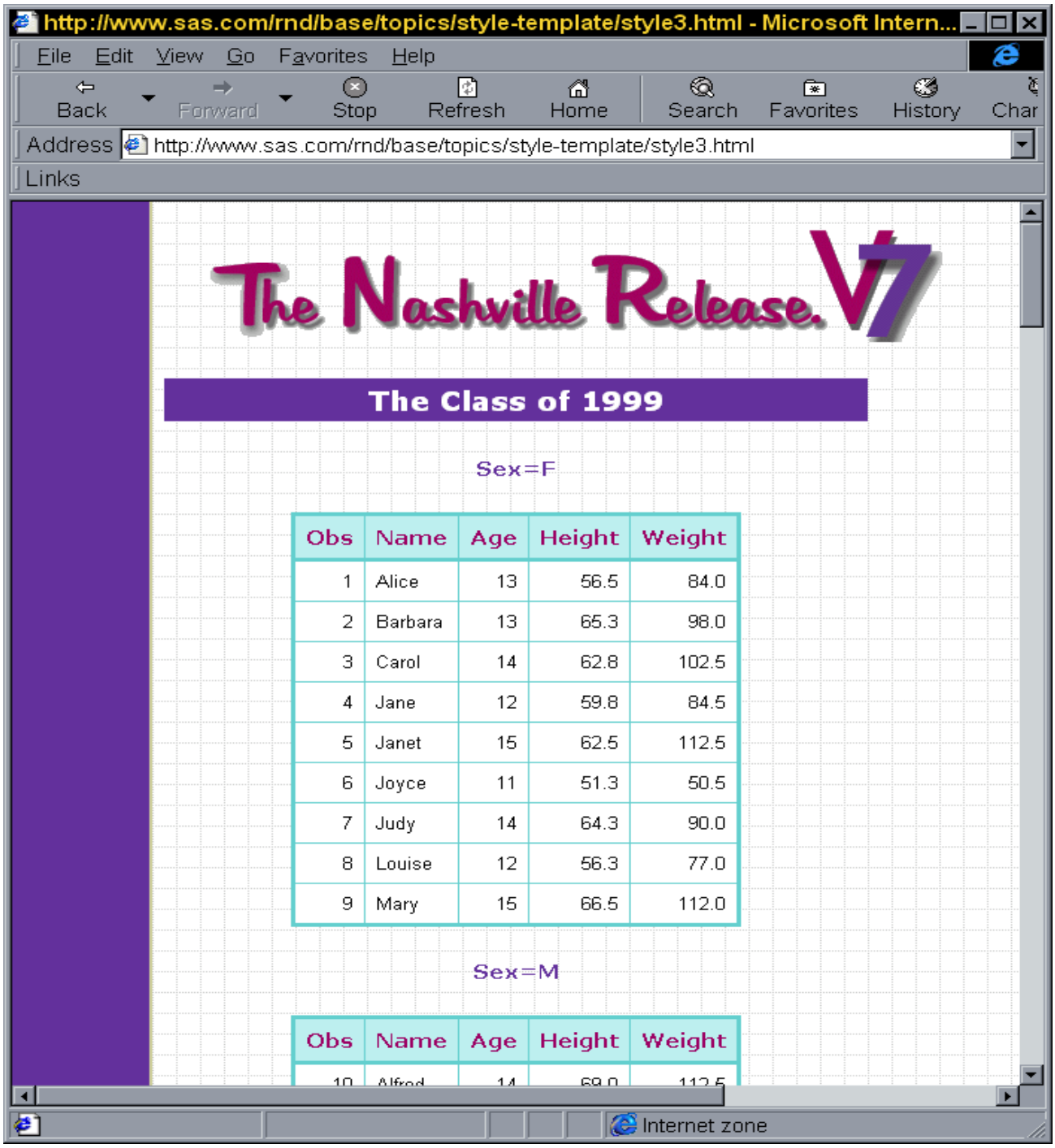

SAS Output - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels
Address E:\sascro\ods\src\quarter4.htm
Links
Quarter: 4
Report Created: 28JAN99 20:24
% Difference
$ Difference
Department
Account
Budget
Actual
Equipment lease $40,000.00 $40,000.00 0.00% $0.00
purchase $40,000.00 $17,769.15 (55.58%) $-22,230.85
tape $12,000.00 $11,426.73 (4.78%) $-573.27
sets $7,500.00 $8,079.62 7.73% $579.62
maint $12,000.00 $10,675.29 (11.04%) $-1,324.71
rental $6,000.00 $5,482.94 (8.62%) $-517.06
Facilities rent $24,000.00 $24,000.00 0.00% $0.00
utils $3,500.00 $3,444.81 (1.58%) $-55.19
supplies $2,750.00 $2,742.48 (0.27%) $-7.52
Other advert $30,000.00 $39,325.64 31.09% $9,325.64
talent $19,500.00 $39,424.64 102.2% $19,924.64
Get a new job!
musicfee $5,000.00 $4,875.95 $-124.05
Staff fulltime $165,000.00 $166,345.75 0.82% $1,345.75
parttime $60,000.00 $56,018.96 (6.64%)
Local intranet zone

Figure 5:


http://www.sas.com/rnd/base/topics/style-template/style3.html - Microsoft Intern...
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Char
Address http://www.sas.com/rnd/base/topics/style-template/style3.html
Links
The Nashville Release. V7
The Class of 1999
Sex=F
Obs Name Age Height Weight
1 Alice 13 56.5 84.0
2 Barbara 13 65.3 98.0
3 Carol 14 62.8 102.5
4 Jane 12 59.8 84.5
5 Janet 15 62.5 112.5
6 Joyce 11 51.3 50.5
7 Judy 14 64.3 90.0
8 Louise 12 56.3 77.0
9 Mary 15 66.5 112.0
Sex=M
Obs Name Age Height Weight
Internet zone