

**ODS LAYOUT: Arranging ODS Output as You See Fit**  
**Brian T. Schellenberger, SAS® Institute Inc., Cary, NC**  
(revised for SCSUG)

**Abstract**

A new experimental feature for version 9 (both v9.0 and v9.1) is the ODS LAYOUT feature. Using this feature, you can easily mix graphics, text, and tables on the same page, arranging them in any position you want on the page. It's a bit like PROC GREPLAY only not just for graphics anymore. Complex reports and even forms, which were previously almost impossible to produce with ODS, are now quite straightforward to create. If you are required to present your data in a very specific format, with different parts of the report at different positions on the page, we finally have a solution for you.

**Introduction**

The Output Delivery System, introduced in version 7, provided from the beginning the means to format your SAS® output with flexible formatting in a variety of different formats. However, in its initial releases it lacked the ability to produce complete *reports*, necessitating additional formatting steps for many users of ODS. Missing were the ability to format *in the small*, such as emphasizing certain words within a report, and to format *in the large*, as in arranging the different parts of a report in the desired manner.

**Formatting in the Small**

The problem of formatting in the small was addressed with the introduction of in-line formatting in version 8.2. This

feature permits you to put formatting "in-line" with your data, thereby allowing you to change the font and foreground color for certain words within a cell or paragraph of text. It also allows special effects, such as forcing or "suggesting" line breaks, causing pages to break at particular places, inserting space, producing special characters, subscripts and superscripts, and so forth. This feature is described in my paper entitled *Presentation-Quality Output for ODS PRINTER*, which is available at <http://support.sas.com/rnd/base/topics/odsprinter/qual.pdf>.

To be perfectly honest, this feature is a bit rough (the syntax is extraordinarily awkward and some of the syntax simply won't work in certain foreign character codes), and in actual point of fact we "un-documented" it in v9 in anticipation of a an improved syntax which, unfortunately will *not* make it into v9.1 as originally hoped. However, for those willing to do a little digging for information and willing to put up with the rather wretched syntax, it makes it possible to do "real" formatting with ODS. In fact, this paper (and the paper on that feature) are both written *in SAS*, using ODS PRINTER together with in-line formatting and two other new v8.2/v9 features: COLUMNS=, to allow for multi-column formatting, and STARTPAGE=, to allow output from multiple procedures to be combined onto a single page.

### Formatting in the large

The in-line formatting largely, if imperfectly, addressed the issues of formatting in the small, but the issue of formatting in the large still remains. The COLUMNS= feature was perhaps a small hint of things to come, but provides very little flexibility. But available experimentally in both v9 and SAS 9.1 is the ODS LAYOUT feature, which provides a great deal of flexibility to lay out your output as you see fit.

This feature is experimental in v9.1 for more than one reason. The *primary* reason that it is experimental is that we do not wish to repeat the mistake we made with in-line formatting: we want to be certain that the feature truly meets the needs of our users, both in terms of functionality and ease of use, before we declare it production. This has a couple of implications: first, it means that the syntax is *subject to change* before it becomes production; second, it means that we hoping to get lots of feedback on whether we "got it right" before we finalize it. This means that we strongly encourage you to try it, but please understand that any code you write with the current version of ODS LAYOUT may need to be completely re-written when the final version is released.

*"Being experimental means never having to say you're sorry."*

The second reason it is experimental is that it is still a little rough around the edges. The feature will be subject to

various failures when stressed, and some of the features described herein will not be working in v9.1. All such features will be pointed out in the paper.

### Limited Destinations

#### (sorry, no RTF)

One of the more unfortunate limitations of this feature is that it will not not work for all destinations. In the first release it is only being really tested for the PRINTER destination family. The gridded layout should work reasonably well for HTML as well, but it will have far less testing. Absolute layout (explained later, of course) is not supported at all for HTML in v9.1. An even bigger problem is support for RTF. Inherent limitations of the current version of the RTF format make it impossible for us to support this feature at all. (It supports neither nested tables nor absolute positioning, the two features necessary to support ODS LAYOUT.) We hope to convert to a later version of RTF in the future but this will probably be sometime past SAS 9.2 – it would be wise to not expect LAYOUT to work for RTF at all for the next couple of releases.

### ODS LAYOUT

This brings us, finally, to the feature itself. There are actually two similar but different varieties of ODS LAYOUT: *absolute* layout and *gridded* layout.

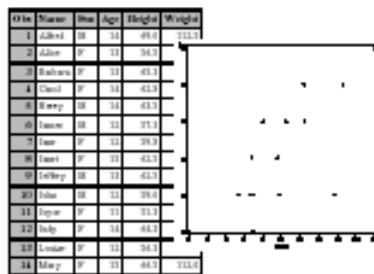
Absolute layout is where you specify the exact position of each region within the layout; gridded layout essentially acts like a table with each "cell" containing the output (usually) from a single procedure.

Some simple examples will probably make this more clear.

Here is some code that uses the **absolute** layout approach:

```
ods layout start width=6in
height=4in;
ods region width=4in height=4in
x=0 y=0;
proc print data=sashelp.class;
run;
ods region width=3in height=3in
x=3in y=.5in;
proc gplot data=sashelp.class;
plot age*weight; run;
ods layout end;
```

This will produce output like this:



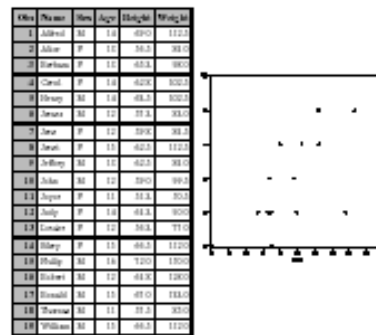
Note that here we have specified the exact position of each of the regions. This approach is very precise and flexible; using it you can produce a report that precisely matches a specified format, and it is suitable for producing things like monthly utility bills where the exact position of an item is important and the size of the items in the report is fixed. However it has its drawbacks as well.

First, it is complex and somewhat error-prone; for example, in the sample above

the table is being over-printed by the graph. That is because I messed up the specification. The first region was made too wide.

Another drawback is that when you use absolute layout, the size of each region is fixed when you specify it; if the output will not fit in the allotted space then the "extra" output is simply discarded. (There will be some options to modify this behavior available in v9.2, but this is the "usual" behavior of the absolute layout.) This is also shown in the above example, where the class data set is cut off at observation 14.

Here is the correct result for this example:



And the corrected specification:

```
ods layout start width=6in
height=6in;
ods region width=3in height=6in
x=0 y=0;
proc print data=sashelp.class;
run;
ods region width=3in height=3in
x=3in y=1in;
proc gplot data=sashelp.class;
plot age*weight; run;
ods layout end;
```

To deal with these problems, v9.1 introduces **gridded** layout as a simpler alternative. (Actually, this syntax is available in v9 as well but the implementation is so poor that I recommend you just ignore it until you install v9.1. Much of what I say about gridded output in this paper is not true of it in v9.) With gridded output, you simply tell ODS how many columns you want, much as with a table, and it will make each region as big as it needs to be to accommodate the output, even if the output goes across multiple pages. Here is an example of gridded output syntax:

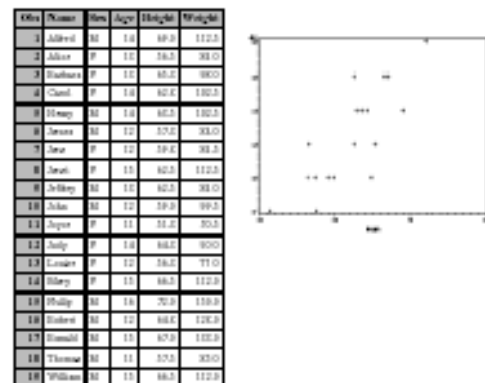
```
ods layout start columns=2;
ods region;
proc print data=sashelp.class;
run;
ods region;
proc gplot data=sashelp.class;
plot age*height; run;
ods layout end;
```

In fact, the syntax can be even simpler than this (at least in v9.1; this is one of those features about which I'd like *feedback--is this a good idea?*). Within gridded layout, an explicit newpage request is interpreted as a request to go to the next region, just as it is interpreted as a request to go to the next column when the COLUMNS= option is used. (Of course you can use STARTPAGE=NO to suppress this behavior.) Since ODS always produces an explicit newpage before each procedure, the practical effect most of the time is that each procedure is

automatically placed into the next region. Thus, the above code can actually be reduced to the below code--the region statements are not actually necessary.

```
ods layout start columns=2;
proc print data=sashelp.class;
run;
proc gplot data=sashelp.class;
plot age*height; run;
ods layout end;
```

The output from this is as follows:



Just as with normal table cells or absolute layout, it is *possible* to explicitly specify the size of an individual region, but it is not necessary, and I would predict that this ability will be only rarely used. Then again, my track record at predicting how SAS users will want to use our features is quite poor, so this prediction may turn out to be utterly wrong. Because gridded layout is, as the name suggests, laid out like a rectangular grid, or table, it is not as flexible as absolute layout. Special effects such

as overlapping regions, which are permitted in absolute layout are not possible in gridded layout, although it *is* possible to have a region span across multiple rows (down) or columns (to the right) by using options to the ODS REGION command.

Another limitation of the gridded layout, though not in my opinion a particularly problematic one, is that the input can only be specified in order; that is, the data is always filled in left to right and then top to bottom, whereas absolute layout can be produced in any order--subject, however, to its *own* limitation that a single absolute LAYOUT cannot span pages.

### The nitty-gritty

The ODS LAYOUT statement supports the following syntax:

```
ods layout start [columns=int]
               [rows=int]
               [column_widths=(unit, ...)]
               [row_heights=(unit, ...)]
               [width=unit]
               [height=unit]
               [column_gutter=unit]
               [row_gutter=unit]
               [entirepage];
ods region [column=int]
           [row=int]
           [column_span=int]
           [row_span=int]
           [x=unit]
           [y=unit]
           [width=unit]
           [height=unit]
           [chain]
           [continue];
ods layout end;
```

Above, *int* is of course an integer (which should in all cases be at least one), and *unit* means a measurement including a unit, such as 1.5in or 25% or 17pt. Due to the context in which LAYOUT/REGION is used, font-based measurements don't really make any sense, so the units em and ex are not supported. However, a special wt unit is accepted by the grammar and *may be* supported in the future. It would tell the "weight" that the column would take up, and be usable only for gridded output (and later for ordinary tables as well) and tell us how to distribute any non-fixed space that is "left over" after accounting for all of the columns with fixed sizes. Feedback on the utility of such a measurement would be most welcome.

The options are described below. Next to each item is the statement(s) to which it applies as well as the type of layout to which it applies, and the status of the feature in v9.1 and then (starting on the next line) a description of what it does. Please understand that in v9 *nothing* related to **gridded** layout should be considered reliable at any time. Also, at the end of many of the descriptions is a "feedback" section. Here I am posting questions to *you* about how you think this should work.

#### **COLUMNS=.** – LAYOUT – **gridded** – *reliable*

Tells the number of columns for the gridded layout. In fact, the presence of the COLUMNS= option is the key that tells ODS LAYOUT that it is doing gridded

layout rather than absolute layout.

*Feedback:* Is this too "magical"?

Would it be better to instead require an ABSOLUTE or GRIDDED keyword to make this distinction? I've already gotten some feedback that this *would* be a better design but more thoughts are very welcome.

**ROWS=.** – LAYOUT – **gridded** – *ignored*

This tells the number of rows that the gridded layout occupies. It seems like a sort of nice thing to specify; it's parallel with the COLUMNS= value, after all. However, in practice it serves no real purpose. I know how many rows there are, after all, based on how much data (how many regions) there are. It *could* be used to cause ODS to simply ignore any data that occurs after the cutoff point provided by the ROWS= value but at this point it does not do even that.

*Feedback:* Should I get rid of the ROWS= or use it to allow the gridded layout to cut off the output at certain point?

**COLUMN\_WIDTHS=.** – LAYOUT – **gridded** – *ignored*

This is a list of measurements, telling how wide to make each column. The first unit applies to the first column, the next one to the second column, and so forth. The list is expected to be complete if it exists at all.

This is actually ignored in v9.1 (it did work, ironically, with v9) and I'm also not sure how terribly useful it is since, after all, you can specify width on individual regions and ODS LAYOUT will automatically use the maximum width, just as with a table.

*Feedback:* Is this a useful option to retain? If so, should we lift the restriction to having a complete list?

**ROW\_HEIGHTS=.** – LAYOUT – **gridded** – *ignored*

This is a list of measurements, telling how tall to make each row. If the row is never used in the course of the table, the measurement will be ignored. If the row height is specified, then the row will be forced to be that height just as if each region had specified that height. If there are more rows than values, the remaining row will have their "natural" height.

As of this writing it is ignored with v9.1, and its utility is questionable.

*Feedback:* Is this a useful option to retain?

**WIDTH=.** – LAYOUT – **absolute** – *reliable*  
REGION – **absolute/gridded** – *reliable*

This allows you to specify the width of the layout or region. LAYOUT: In the case of the LAYOUT statement, only

absolute layout supports WIDTH=. If the WIDTH= is omitted from the LAYOUT statement, it defaults to the width of the current region, column, or page.

REGION: In the case of the REGION statement, the width is *required* for absolute layout and optional for gridded layout. If omitted, the gridded region will default to its "natural" width or, if it is an item that has no natural width (such a graphic), to its "fair share" of the width; that, is to half the width for COLUMNS=2, one third for COLUMNS=3, and so forth.

*Feedback:* should we allow WIDTH= for the entire layout in the gridded case as well? It's not immediately obvious to me what purpose that would serve but it is possible to specify widths for tables, so perhaps it could be of some use.

**HEIGHT=.** – LAYOUT – **absolute** – *reliable*

REGION – **absolute/gridded** – *reliable*

This allows you specify the height of the layout or region.

LAYOUT: In the case of the LAYOUT statement, only absolute layout supports HEIGHT=. If the HEIGHT= is omitted from the LAYOUT statement, it defaults to the height of the current region, column, or page.

REGION: In the case of the REGION statement, the height is *required* for absolute layout and optional for gridded layout. If omitted, the gridded region will default to its "natural" height or, if it is an item that has no natural height (such a graphic), to its "fair share" of the height; that, is to half the height for COLUMNS=2, one third for COLUMNS=3, and so forth. It might seem odd that the height default to a value based on the *column* count, but it keeps graphics from getting distorted since the width defaults in this manner.

*Feedback:* should we allow HEIGHT= for the entire layout in the gridded case as well? It's not immediately obvious to me what purpose that would serve but it is possible to specify heights for tables, so perhaps it could be of some use.

**COLUMN\_GUTTER=.** – LAYOUT – **gridded** – *untested*

This specifies the space to leave between columns. Actually, that's the *intention*, but right now the space between columns and the space between must be identical, so really this and the ROW\_GUTTER are averaged to get the space between both rows and columns.

**ROW\_GUTTER=.** – LAYOUT – **gridded** – *untested*

This specifies the space to leave between rows. Actually, that's the *intention*, but right now the space between columns and the space between must be identical, so really this and the ROW\_GUTTER are averaged to get the space between both rows and columns.

**ENTIREPAGE.** – LAYOUT – **absolute** – *unimplemented*

This keyword (once it works) will specify that the (absolute) layout should occupy the entire page. If this options appears at all, it should appear by itself on the ODS LAYOUT statement. Note that since the layout occupies the *entire* page, the layout is automatically started on the next page if there is already output on the current page, and all normal page numbering, dates, titles, and footnotes are suppressed. Therefore if you want any of these elements to appear on that page you must produce them yourself within the layout.

*Not yet implemented--targetted for v9.2.*

**COLUMN=.** – REGION – **gridded** – *untested*

This option lets you specify the column where the region should be placed. If this is later than the next column in the sequence, then columns will be skipped until we reach the designated column. If it is *before* the next

column in the sequence, then the next row will be started and columns will be skipped until the specified column is reached. If used in combination with COLUMN\_SPAN=, this refers to the *first* column that the region occupies.

**ROW=.** – REGION – **gridded** – *untested*

This option lets you specify the row where the region should be placed. If this is later than the next row in the sequence, then row will be processed until we reach the designated row. Normally, this does nothing interesting at all since the skipped rows occupy no space, but if you used the ROW\_HEIGHTS= option on the LAYOUT statement, then the skipped rows will still skip the indicated amount of space. It is not legal to specify a row *before* the current row. If used in combination with ROW\_SPAN=, this refers to the *first* row that the region occupies.

**COLUMN\_SPAN=.** – REGION – **gridded** – *untested*

This option lets you specify a number of grid columns for the region to occupy. This option in combination with ROW\_SPAN allow gridded layout to have somewhat more flexibility and thus to solve *some* of the problems that would otherwise

require the use of absolute layout.

**ROW\_SPAN=.** – REGION – **gridded** – *untested*

This option lets you specify a number of grid columns for the region to occupy. This option in combination with COLUMN\_SPAN allow gridded layout to have somewhat more flexibility and thus to solve *some* of the problems that would otherwise require the use of absolute layout.

**X=.** – REGION – **gridded** – *reliable*

This option specifies the horizontal position of the region, which will extend to the right of this position for WIDTH. If omitted, it defaults to 0.

**Y=.** – REGION – **gridded** – *reliable*

This option specifies the vertical position of the region, which will extend down from this position for HEIGHT. If omitted, it defaults to 0.

**CHAIN.** – REGION – **absolute** – *unimplemented*

If the output overflows this region, then flow it into the top of the next specified region.  
*Not yet implemented--targetted for v9.2.*

**CONTINUE.** – REGION – **absolute** – *unimplemented*

If the output overflows this region, then duplicate the entire

layout and continue the contents of this region onto the same region space in the duplicated layout. Repeat as necessary to keep from truncating the output.

*Not yet implemented--targetted for v9.2.*

## Noteworthy options

### ENTIREPAGE

Normally, if you omit WIDTH and HEIGHT the absolute region will already default to occupy all of the available space in the page, but there are still some problems with this when you are trying to follow a design specification:

1. The available space does not include the space occupied by system titles, footnotes, page numbers, and such.

2. The layout is still only within the page margins. Of course you often can't draw outside of the margins anyway on a lot of printers, but even if you don't actually want to *draw* outside of the margins, they still get in your way: without knowing where the margins are, you don't know how far from the edge of the page something is. For example, if you do an absolute layout, and put a region at position  $x=1$  in, that means 1 inch from the left side of the layout, and that in turn means one inch from the margin. This makes the calculations complex when

working from a design specification that has positions relative to the edge of the paper.

Without the ENTIREPAGE option, you can get around these problems by setting all four margins to zero (for the second problem) and setting "OPTIONS NONUMBER NODATE;" and "TITLE; FOOTNOTE;," but this is a bit awkward. For v9.1, that is in fact the only solution, but in the next release we will introduce the ENTIREPAGE option. When this option is used, ODS LAYOUT will ensure that it is starting a new page, will suppress all normal titles, footnotes, page numbers, and so forth, so that *only* the layout will be printed on the page, and will interpret all layout positions as relative to the upper left-hand corner of the *paper*, since the layout will be presumed to fill up the entire physical page.

### CONTINUE and CHAIN

Probably the biggest single potential drawback of absolute layout is that each region has a fixed size, and if the output will not fit within that size, it has no place else to go. So the normal behavior and the only behavior currently working in v9.1 is that the extra output simply vanishes. That is unfortunate; it can lead to data loss if you are unaware of it.

There will be two keywords to help deal with this problem. The CONTINUE keyword (which maybe should be named REPEAT or something; as usual, feedback is welcome) will cause the region to be continued onto an identical

copy. That is, if there is a CONTINUE region and it runs out of room, then when the rest of the layout has been completed, the entire layout will be repeated and the region with the remaining output will be replicated at the same relative position.

The usual expectation for usage of this option is that you would use it with regions that basically occupy a page (either ENTIREPAGE or something that occupies all the remaining space, or at least more than half of it), so that the effect is that the region continues across pages (in the same location) until the output is exhausted. However, it's not actually *required* that the layout occupy an entire page in order to use this option.

The other option to help with handling overfull text is the CHAIN option. If the CHAIN option is used, then if the current region is full, the output will continue into the next region, appearing prior to any output specified in that region. The normal expectation is that this region will be empty. This option would be used to do "magazine-style" layout, for example, by using an absolute layout with a graph, perhaps, in the middle of the page and then a series of chained regions that would specify the space around the graphic. For example, the following might be used:

```
ods layout entirepage;

/* First, a title section */
ods region x=0.5in y=0.5in
```

```

        width=7.5in height=0.5in;
ods printer text="TITLE . . .
/* Then the mid-page graphic */
ods region x=3in    y=4in
        width=2in height=4in;
proc gplot; plot . . . .
/* Then the region that will specify the
* text. Specify chain so it can overflow
* the remaining regions.
*/
ods region x=0.5in    y=1.5in
        width=3.5in height=1.5in chain;
ods printer text="First paragraph . . .
ods printer text="Second paragraph . . .
:
: more text
:
/* and the other locations for the text
*/
ods region x=0.5in    y=4in
        width=2.25in height=4in chain;
ods region x=0.5in    y=8in
        width=3.5in height=2.5in
chain;
ods region x=4.5in    y=1.5in
        width=3.5in height=1.5in
chain;
ods region x=5.25in    y=4in
        width=2.25in height=4in chain;
ods region x=4.5in    y=8in
        width=3.5in height=2.5in;

```

### The paper itself

The “true” version of this paper (both the printed version and the overheads) were themselves produced using ODS PRINTER and its features, including the ODS LAYOUT statement.

However, *this version of the paper – the one in the SCSUG proceedings – was not produced directly by SAS!* In order to

meet the requirements of the SCSUG proceedings it was converted to Microsoft Word format (something at which I am *very* inexperienced which is no doubt why this looks so darn ugly).

If you want to see the original version of the paper, it can be found in the proceedings of the SUGI conference, the PharmaSUG conference, or the NESUG conference, if you happen to have access to them.

Otherwise, it is on the web at <http://support.sas.com/rnd/base/topics/odsprinter/layout.pdf>. Some of what I write below will probably make a little more sense if you check out that version. The SAS source used to produce the original version of the paper can also be found on the parent page there.

The formatting macros themselves are available on the web (see the Contact information below for the URL). The way that the sample output was included was to actually run the formatting code in line, surrounding by formatting macros to create a “scaled page.” The code in this paper looks like this (this is the first example in the paper):

```

%sp_start(0.5, yfactor=0.2);
ods layout start width=3in
height=2in;
ods region width=2in height=2in
x=0 y=0;
proc print data=sashelp.class;
run;

```

```

ods region width=1.5in
height=1.5in x=1.5in y=0.25in;
proc gplot data=sashelp.class;
plot age*weight; run;
ods layout end;
%sp_end;

```

This says to do a scaled page where the page is shrunk down to half of its usual size, with the y dimension being only 20% of the usual size. The way that this macro works internally is to do two things: First, establish a layout and region, as follows:

```

%if &xfactor = . %then %let
xfactor = &factor;
%if &yfactor = . %then %let
yfactor = &factor;
%let x = %scale_style(7,in,
thisfactor=&xfactor);
%let y = %scale_style(9,in,
thisfactor=&yfactor);
ods layout start width=&x
height=&y;
ods region x=0 y=0 width=&x
height=&y;

```

And then change to a style which will scale down all of the normal elements to &factor of their usual size. Here are a few "selections" from that style definition:

```

proc template;
define Style styles.Scale;
parent = styles.printer
;
/* ----- */
replace fonts /
"docFont" = ("Times
Roman", %scale_style(10,pt)
)

```

```

:
style Table from Output /
BorderWidth =
%scale_style(.75,pt)
CellSpacing =
%scale_style(.25,pt)
CellPadding =
%scale_style(4,pt)
Both of these use the
scale_style macro, which is very
simple:
%macro
scale_style(amount,unit,thisfact
or=&factor);
%let scaled =
%sysevalf(&amount *
&thisfactor);
&scaled.&unit
%mend;

```

So, as you can see, the layout provides a way, in conjunction with existing features, to provide an abstract "mini-page" which can function just like an entire page and entirely shrunk down. I used this technique for the enclosed examples, so I never had to capture images and paste them in or anything like that. The entire paper was produced in-line in a single SAS file. Although SAS is in many ways an imperfect formatting tool (the error messages from any macro-based program are inherently confusing for one thing), it is a clear demonstration of the power that ODS PRINTER now provides you for formatting your output.

## Conclusion

The ODS LAYOUT statement will provide you a powerful new tool for preparing reports using ODS, provided that you don't need RTF output. I especially look

forward to hearing feedback about this feature and the plans for it.

### **Contact Information**

As stated throughout the paper, feedback is encouraged about any aspect of this feature, this paper, or indeed anything else relating to ODS PRINTER and ODS in general. You may contact the author at:

Brian T. Schellenberger  
SAS Institute Inc.  
SAS Campus Drive  
Cary, NC 27519

My e-mail address is  
Brian.Schellenberger@sas.com,  
but for general feedback on ODS please  
send mail instead to ods@sas.com,  
which is our general "ODS developer"  
address, and goes out to all of the ODS  
developers. It is great for general  
suggestions and questions about  
experimental features. General bug  
reports or technical support issues  
should, as always, go to SAS technical  
support.

The URL for all of my papers is:

<http://support.sas.com/rnd/base/topics/odsprinter/>

The original version of this paper, along with the SAS source and macros used to produce it are among the many treasures to be found there.

### **Trademark Information**

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 brands and product names (not that there actually are any in this paper) are registered trademarks or trademarks of their respective companies.