Paper 128-27

What's New in the Output Delivery System, Version 9.0

Sandy McNeill, SAS, Cary, NC

ABSTRACT

Along with every new version of SAS®, the ODS developers work hard to bring you new features. So what's new with Version 9? A new destination called MARKUP along with something called markup definitions (aka TAGSETs) will become production. This new destination allows configurability of your markup output. Another new destination (also available as a PROC) that will be production for Version 9 is the DOCUMENT. If you've ever wanted to the ability to run your procedures once and then go back and pick and choose which pieces you want or how you would like them rendered, then this is for you.

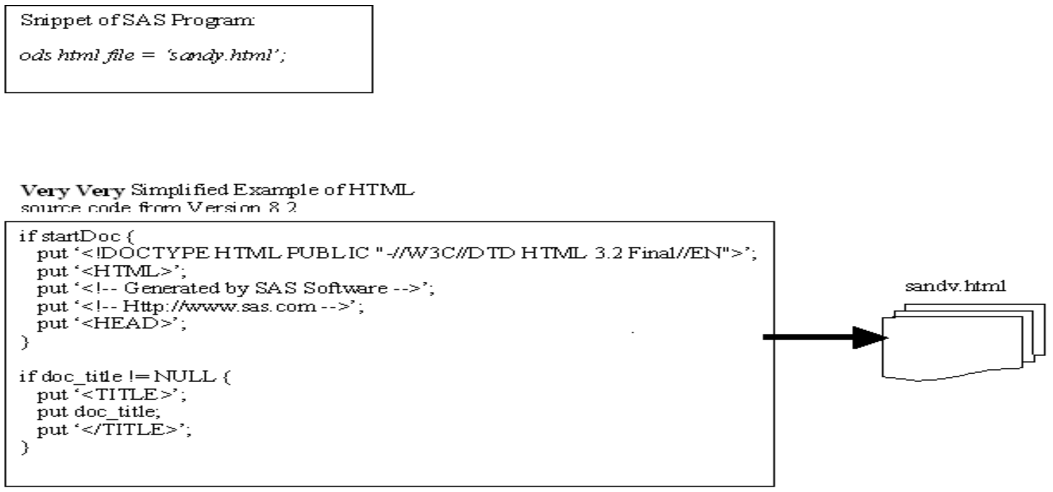
In addition to these new destinations, RTF has some new features and functionality -- options for Page X of Y, decimal alignment, and cell indention are a few of the highlighted features.

INTRODUCTION

Ahhh……another year and more new features. We'll start this paper by discussing the two larger features, MARKUP destination and the DOCUMENT destination, and then finish up by discussing the other smaller but equally important features. Three of these smaller features are specific to RTF-- the PAGEOF option, the ability to change orientation without closing the destination, and SASDATE. The other options are specific to both RTF and the PRINTER destination – TEXT=, COLUMNS=, the ability to specify margins on cells, and indention within cells.

MARKUP DESTINATION AND TAGSET TEMPLATES

The MARKUP destination is very important because it is a shift for ODS. The MARKUP destination started as a way for ODS to produce XML, but it is (or will be) the brain of any of our markup destinations, such as HTML, LaTeX, TROFF, or XML, to name a few. To really understand the MARKUP destination (and inevitably TAGSETS), it is best to understand how ODS has produced any markup output up to now. To produce ODS HTML files (we'll use HTML since that is probably the best understood markup destination), everything is contained in our source code. All the control processing, all the tags (those things that are surrounded by '<' and '>') -- everything has been hard-coded in the source code. This worked ok for a while, but then we started receiving requests from users that they wanted to have more control. Or aside from having more control, there were instances where a bug was encountered (EEK!) and there was no way to work around the problem which left users with having to wait for a maintenance release or a hotfix (insert your favorite grumblings here). But probably the most important item for consideration, especially from a development point of view, was software reuse. The ODS department was receiving more and more requests for different markup destinations (all different flavors of XML) and other destinations that don't seem so "markup"ish, but resembled the same flow of control as the markup destinations. An example of one of these destinations is CSV, comma separated variable files. For each one of these new destinations, if everything were to remain hard-coded, we would end up with separate code paths for each of these, which means more code, more maintenance, and bigger headaches. What were the similarities of each of these destinations? The flow of control was basically the same, but differences were encountered at similar areas such as the tags used (or not used) to denote the start of a row, the tags used (or not used) to separate cell values, etc. It became clear that at these similar points in the code (EVENTS), it would be nice to be able to call out to some centralized, shared code to request the tags to be used at this point of the document given the type of document trying to be produced and the location at this point in the document. It was also at this time that it became clear that it would be very nice if this centralized, shared code were able to be modified by users so that if they needed to tweak the tags, or even generate their own form of markup, it would be possible. And so was born a new form of ODS template which can be created and modified by using Proc Template – the ODS TAGSET template, or what I like to refer to as MARKUP definition files. Figure 1 below is a very simplified example of how the ODS source code in Version 8.2 got your output to the HTML file that you specified. Notice that all the "put" statements are inside the source code. This is compiled source code, so you have no hope of changing something that you do not like. Figure 2 is a diagram which shows a simplified view of what the MARKUP source code looks like now and the interaction with the TAGSET template.

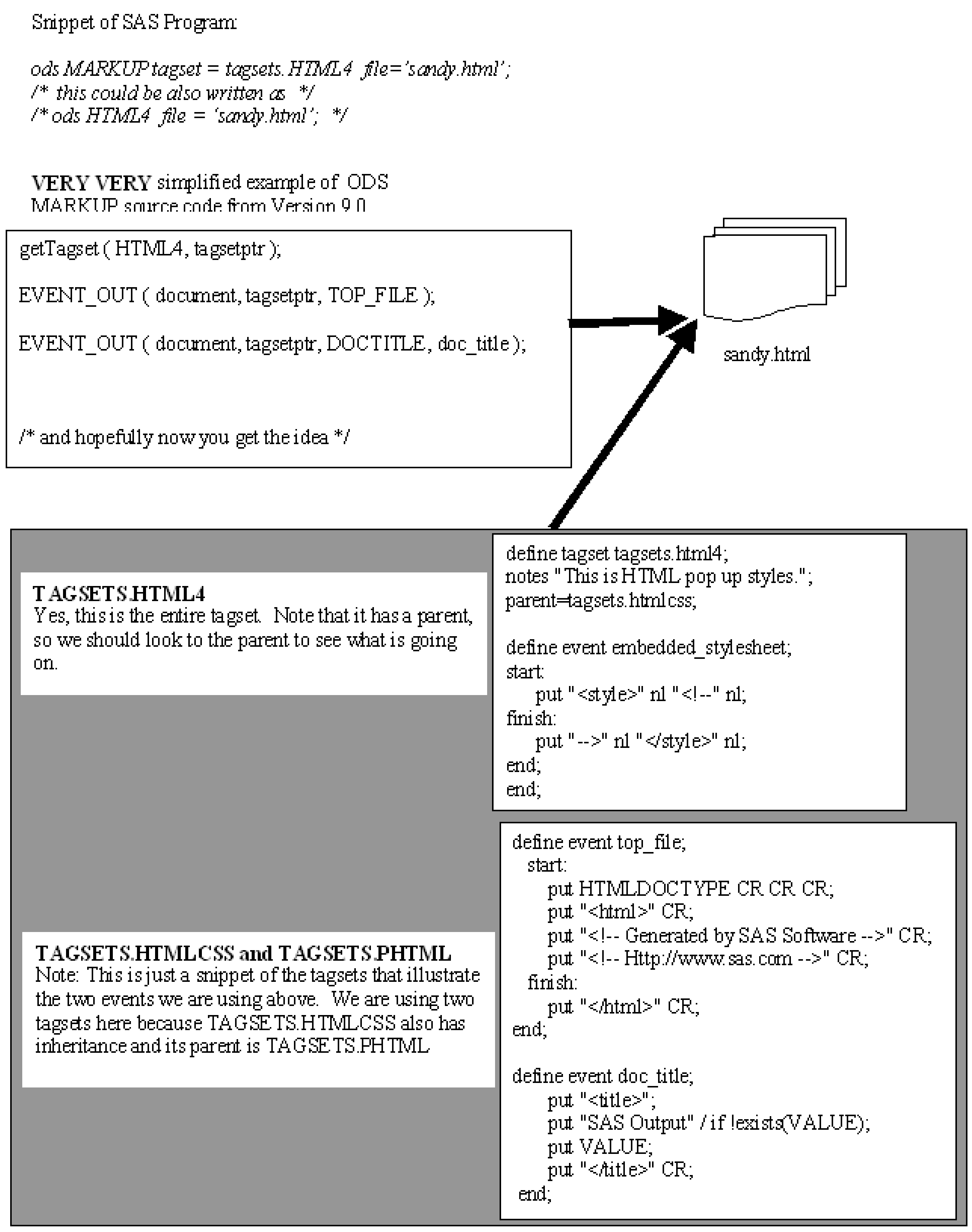

Snippet of SAS Program:

```
ods html file = 'sandy.html';
```

Very Very Simplified Example of HTML source code from Version 8.2

```
if startDoc {
   put '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">';
   put '<HTML>';
   put '<!-- Generated by SAS Software -->';
   put '<!-- Http://www.sas.com -->';
   put '<HEAD>';
}

if doc_title != NULL {
   put '<TITLE>';
   put doc_title;
   put '</TITLE>';
}
```

Figure 1: ODS HTML Version 8.2 Source Example



Snippet of SAS Program:

```
ods MARKUP tagset = tagsets.HTML4  file='sandy.html';
/* this could be also written as  */
/* ods HTML4  file = 'sandy.html';  */
```

VERY VERY simplified example of ODS MARKUP source code from Version 9.0

```
getTagset ( HTML4, tagsetptr );

EVENT_OUT ( document, tagsetptr, TOP_FILE );

EVENT_OUT ( document, tagsetptr, DOCTITLE, doc_title );



/* and hopefully now you get the idea */
```

TAGSETS.HTML4
Yes, this is the entire tagset. Note that it has a parent, so we should look to the parent to see what is going on.

```
define tagset tagsets.html4;
notes "This is HTML pop up styles.";
parent=tagsets.htmlcss;

define event embedded_stylesheet;
start:
     put "<style>" nl "<!--" nl;
finish:
     put "-->" nl "</style>" nl;
end;
end;
```

TAGSETS.HTMLCSS and TAGSETS.PHTML
Note: This is just a snippet of the tagsets that illustrate the two events we are using above. We are using two tagsets here because TAGSETS.HTMLCSS also has inheritance and its parent is TAGSETS.PHTML

```
define event top_file;
  start:
      put HTMLDOCTYPE CR CR CR;
      put "<html>" CR;
      put "<!-- Generated by SAS Software -->" CR;
      put "<!-- Http://www.sas.com -->" CR;
  finish:
      put "</html>" CR;
end;

define event doc_title;
      put "<title>";
      put "SAS Output" / if !exists(VALUE);
      put VALUE;
      put "</title>" CR;
 end;
```

Figure 2: ODS MARKUP and TAGSETS Source Example

EXAMPLE OF A TAGSET
Once the code was written to interface with the tagset definition, generating different forms of markup output is now fast and easy.

Here is a list of the different tagset definitions that are supplied by SAS.

```
Listing of: SASHELP.TMPLMST
Path Filter is: Tagsets
Sort by: PATH/ASCENDING

Obs    Path                          Type
ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ
 1     Tagsets                       Dir
 2     Tagsets.Chtml                 Tagset
 3     Tagsets.Colorlatex            Tagset
 4     Tagsets.Csv                   Tagset
 5     Tagsets.Csvall                Tagset
 6     Tagsets.Csvbyline             Tagset
 7     Tagsets.Default               Tagset
 8     Tagsets.Docbook               Tagset
 9     Tagsets.Event_map             Tagset
10     Tagsets.GTableApplet          Tagset
11     Tagsets.Graph                 Tagset
12     Tagsets.Html4                 Tagset
13     Tagsets.Htmlcss               Tagset
14     Tagsets.Imode                 Tagset
15     Tagsets.Latex                 Tagset
16     Tagsets.Latex2                Tagset
17     Tagsets.Namedhtml             Tagset
18     Tagsets.Odsstyle              Tagset
19     Tagsets.Phtml                 Tagset
20     Tagsets.Pyx                   Tagset
21     Tagsets.SASReport             Tagset
22     Tagsets.Sasxmog               Tagset
23     Tagsets.Sasxmoh               Tagset
24     Tagsets.Sasxmoim              Tagset
25     Tagsets.Sasxmor               Tagset
26     Tagsets.Short_map             Tagset
27     Tagsets.Statgraph             Tagset
28     Tagsets.Style_display         Tagset
29     Tagsets.Style_popup           Tagset
30     Tagsets.Text_map              Tagset
31     Tagsets.Tpl_style_list        Tagset
32     Tagsets.Tpl_style_map         Tagset
33     Tagsets.Troff                 Tagset
34     Tagsets.Wml                   Tagset
35     Tagsets.Wmlolist              Tagset
36     Tagsets.sasXML                Tagset
37      Tagsets.sasioXML             Tagset
```

The "Tagsets." which precedes the names of the tagset is the directory within the SASHELP.TMPLMST itemstore in which these tagsets are stored. You can generate this list on your own using a little bit of Proc Template code:

```
Proc Template ;
    List tagsets / store = sashelp.tmplmst;
Run;
```

Please refer to “Getting Started with the Output Delivery System” for more information regarding the general use of Proc Template and its statements since that it out of the scope of this paper.

I am certainly not going to launch into a thesis of what each of these tagsets is for since that is documented at the BASE R&D website I list below. Also, additional information can be found in Eric Gebhart’s SUGI27 paper “ ODS MARKUP: The Power of Choice and Change“. The main point, however, is that before MARKUP and TAGSETS, each of these markup destinations would have been a separate code path in our source code and no way for you, the user, to create your own markup destinations.

To really make a point as to the versatility of these tagsets, here’s a tagset definition which creates “data set”-looking output from Proc Print. Now we normally wouldn’t consider a data set as a form of markup output, but it does have things in common with regular markup destinations: something denoting the beginning of the data, rows of data, data values separated by some delimiter (even if it is a blank), and something denoting the end of the data. These strings such as “Data” and “Run;” would not necessarily be thought of as normal “tags”, but if you think of tags as delineating the different areas of the output, then it conforms nicely.

```
proc template;
define tagset Tagsets.dataset;
   notes "This is the Dataset definition";
   define event table;
      start:
         put "data;" NL;
      finish:
         put "run;" NL;
   end;
   define event row;
      finish:
         put NL;
   end;
   define event table_head;
      start:
        put "input ";
      finish:
         put ";";
   end;
   define event table_body;
      start:
         put "cards;" NL;
   end;
   define event header;
      start:
         trigger data;
      finish:
         trigger data;
   end;
   define event data;
      start:
      put " " TEXT;
   end;
end;
run;


ods markup type=datastep file="b_out.sas";
proc print data=sashelp.class; run;
ods markup close;
```

When using any of the SAS-supplied tagsets, as those 37 listed above, you can use a shortcut notation, just by using the name of tagset. To create DOCBOOK markup:

```
   ODS DOCBOOK file = 'docbookex.xml';
```

However, when you reference a tagset that you have created, as in the DATASET tagset above, you need to use the two-level reference which is the directory and the name of the tagset.

```
   ODS TAGSETS.DATASET file = 'dataset.txt';
```

Another common form of output desired by users is comma-separated variable (CSV) output since a CSV file can be opened by Excel. Again, you wouldn’t normally think of CSV output as a form of markup, but it does have "tags", which are commas, separating the different data values. As an example of how easy it is to modify one of these tagsets, suppose we didn’t want comma-separated value output, but semicolon-separated value output. This request actually came about from email from the European office. If the comma data-value delimiters had been hard-coded in the source code, you would have been out of luck. However, now it’s as easy as modifying the tagset and creating your own tagset. Again, since this is now a user-defined tagset, you have to specify the entire name of the tagset (TAGSETS.SCSV) instead of the shortcut of just using the name of the tagset.

```
proc template;
define tagset Tagsets.SCSV;
notes "This is the semi-colon SV definition";
parent = tagsets.CSV;

define event header;
  start:
     put ";" / if !cmp( COLSTART , "1" );
     put """";
     put VALUE;
  finish:
     put """";
  end;

define event data;
  start:
     put ";" / if !cmp( COLSTART , "1" );
     put """";
     put VALUE;
  finish:
     put """";
  end;

define event colspanfill;
  put ";";
  end;

define event rowspanfill;
  put ";";
end;
end;
run;

ods tagsets.scsv file='foo.txt';
proc print data=sashelp.class;run;
ods tagsets.scsv close;
```

These markup definitions in conjunction with the MARKUP destination have proven to be quite powerful. Since all this was released experimentally with Version 8.2, two users have written their own tagset templates for such things as producing ODS output in SYLK and XHTML (contributed by Jack Hamilton) and for writing ODS output directly into a running EXCEL application using EXCEL DDE macro syntax (contributed by Frank Poppe). You can download these tagsets from http://www.sas.com/rnd/base/topics/odstagsets/#user in addition to reading some pre-production documentation concerning Tagsets and the MARKUP destination. For a more in-depth understanding of ODS Tagsets and the MARKUP destination, you can also refer to Eric Gebhart's SUGI27 paper " ODS MARKUP: The Power of Choice and Change".

Hopefully you now too have a sense of the capabilities and power of the MARKUP destination and TAGSETS. Visit the website and let us know at ods@sas.com if you come up with a tagset that might be of interest to other users. We'd be happy to add it to our user showcase list.

DOCUMENT DESTINATION AND PROC DOCUMENT

The DOCUMENT destination started years ago when ODS was in its infancy. It originally started as the Output Hierarchy and was intended as a way of visualizing the hierarchy of output that was generated from ODS and from there, a way of manipulating and re-arranging this hierarchy. Along the way as ODS matured, the name has changed, but its purpose has somewhat remained intact. The purpose of the DOCUMENT destination is to enable SAS users to re-render ODS output without rerunning procedures and to give the user more control over the structure of the output than is currently available.

OUTPUT WHEN YOU WANT IT

The beauty of ODS is that the user can easily generate SAS output in a variety of formats (destinations). This can happen in parallel -- you just specify as many formats as you want when you run your procedures. It may turn out to be the case that you need to rerun the procedures one or more times. If you are developing a corporate style for SAS reports to be published to the Web, you'll be experimenting with colors and fonts, and thus rerunning procedures. Or perhaps you didn't anticipate that old-guard members of your organization would continue to demand SAS listing output instead of HTML or RTF, so you have to rerun your job in "legacy mode". Rerunning procedures simply to reformat their output is expensive in terms of computation and/or I/O. Data sets have to be rescanned, analyses have to be reapplied, etc. This assumes that it's even possible to rerun the job. If your job uses transient data, then you can't leverage ODS to reformat your output. You're stuck with what you've already generated.

This is where the ODS DOCUMENT comes in. The user may specify to ODS to create an ODS DOCUMENT when generating procedure output. That output will be stored in raw form in the named document, which is managed as a SAS library member. Subsequently the user can browse, edit and rerender (replay) output contained in the document using server-side applications provided especially for that purpose. The ODS document persists in the SAS system until it or the library containing it is deleted. This means that a document created in the WORK library persists no longer than the SAS session that created it. A document created in the SASUSER library might persist for days, weeks or even forever, in which case it could be considered a permanent archive of procedure output.

The format of the output stored in a document is neutral with respect to any other supported ODS formats. This insures a measure of forward compatibility with any as-yet-invented formats that ODS may support in the future.

OUTPUT LIKE YOU WANT IT

Procedures produce data objects which are paired with a template definition to generate *output objects*. Output objects (tables, notes, equations, etc.) are handed over to ODS for rendering into the formats (destinations) selected by the user. Each procedure organizes the output objects into a hierarchy. This *output hierarchy* is presented to users in two ways:

The SAS Explorer Results window
As output objects are created during the SAS session, a GUI treeview representing the hierarchy is dynamically updated. The hierarchy persists no longer than the session.

Markup format table of contents
Users can direct ODS to generate the hierarchy as a table of contents when using markup formats such as HTML and PostScript/PDF. The table of contents is an artifact of the SAS session.

Currently ODS provides the user a great deal of control over the structure of individual output objects. However, the order in which output objects are formatted, and the grouping of output objects into hierarchies, are fixed by the procedure that generates them. This hierarchy inflexibility is as much a pain for the users as the pre-version7 formatting inflexibility. Users don't like this inflexibility. They want to customize the layout of output objects.

The ODS document improves upon this situation. Users can create their own output hierarchies or modify existing ones. Output objects contained in a hierarchy can be ordered any way the user sees fit.

EXAMPLE USAGE

This is a classic example of not liking the current output hierarchy generated from SAS and wanting (needing) to reorder it. This example (borrowed from Training – thanks Cynthia!) uses a simple data set and illustrates the typical output from running two procedures when using bygroups. Since each procedure is run for each bygroup, we see the default hierarchy in Figure 3 where the all the bygroups for Proc Tabulate are together and all the bygroups from Proc Freq are together.

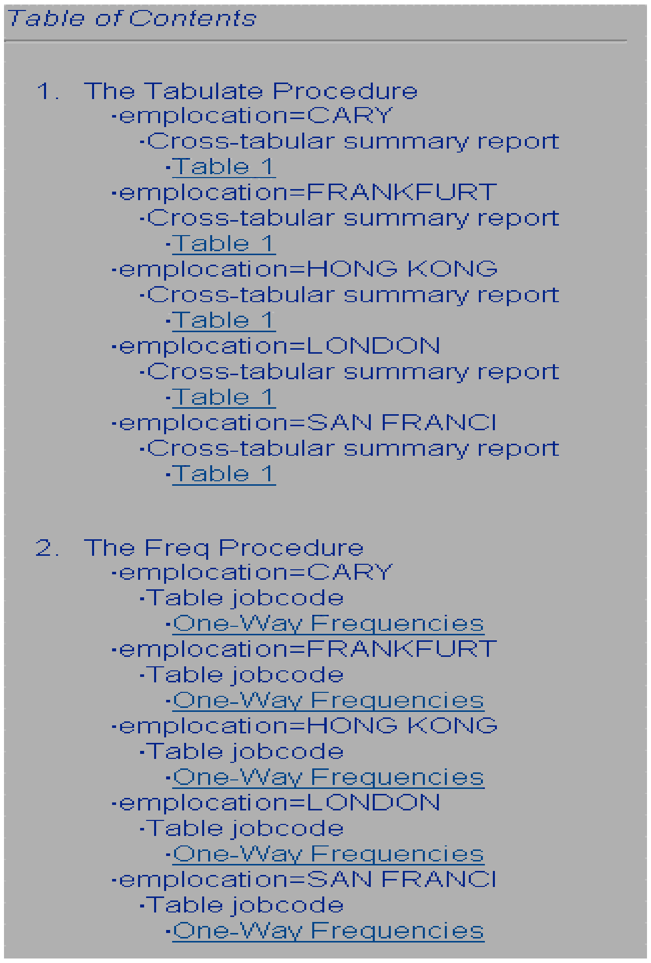


Table of Contents

1. The Tabulate Procedure
   ·emplocation=CARY
      ·Cross-tabular summary report
         ·Table 1
   ·emplocation=FRANKFURT
      ·Cross-tabular summary report
         ·Table 1
   ·emplocation=HONG KONG
      ·Cross-tabular summary report
         ·Table 1
   ·emplocation=LONDON
      ·Cross-tabular summary report
         ·Table 1
   ·emplocation=SAN FRANCI
      ·Cross-tabular summary report
         ·Table 1

2. The Freq Procedure
   ·emplocation=CARY
      ·Table jobcode
         ·One-Way Frequencies
   ·emplocation=FRANKFURT
      ·Table jobcode
         ·One-Way Frequencies
   ·emplocation=HONG KONG
      ·Table jobcode
         ·One-Way Frequencies
   ·emplocation=LONDON
      ·Table jobcode
         ·One-Way Frequencies
   ·emplocation=SAN FRANCI
      ·Table jobcode
         ·One-Way Frequencies

Figure 3: Default TOC. Two Procs and bygroups

What you really would like to see is the output with the bygroups together, so that the bygroup for each employee location would have the two procedures listed under it (see Figure 4).

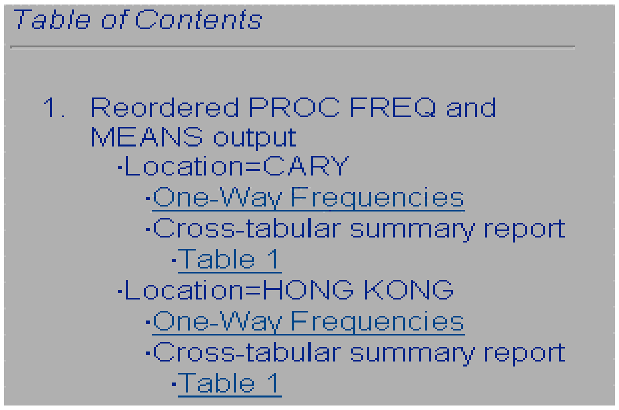


Table of Contents

1. Reordered PROC FREQ and MEANS output
   ·Location=CARY
      ·One-Way Frequencies
      ·Cross-tabular summary report
         ·Table 1
   ·Location=HONG KONG
      ·One-Way Frequencies
      ·Cross-tabular summary report
         ·Table 1

Figure 4: Table of Contents After Proc Document

So how do you get this wishful output? By opening an ODS DOCUMENT as your destination, running the procedures as you normally would, and then close the DOCUMENT. See Step 2 of the following code. Now the output from these procedures has been persisted in the document and you can now retrieve and manipulate this output. If you happen to be satisfied with the current output and you just need to send this output to a different destination, open the destination of your choosing (we chose RTF) and use PROC DOCUMENT to replay the contents of the document (see Step 4). But if you do want to manipulate the stored output, you can use PROC DOCUMENT to create a new DOCUMENT and copy items from the original document into the new one, in whatever order you wish (see Step 5). Besides just reordering the output, one other thing this code points out is that you can retrieve only the pieces of the document that you want. As you can see in Figure 4, there are only two bygroups, CARY and HONG KONG. This is because in the code, the only items that were retrieved from the EMPLOC document and placed into the new EMPLOC2 document were the procedural output for these two bygroups.

Here is the source code for the above figures. I have not included the source for the data set used since that would consume too much space.

```
ods listing close;

/* Set your path accordingly */
ods html path='c:\temp'(url=none)
    frame="bef_frm.html"
    body="bef_bod.html"
    contents="bef_toc.html";

/* 1: All we are doing is running these  */
/* procedures to show what the default   */
/* output hierarchy looks like.          */

proc sort data=data1 out=newdata;
where JobCode contains 'FLTAT' or
      JobCode contains 'PILOT' ;
   by EmpLocation Division;
   run;

Title 'Proc Tabulate Step';
Proc tabulate data=newdata ;
by Emplocation;
var salary;
class jobcode;
table jobcode,salary*(min mean max std);
run;

Title 'Proc Freq Step';
proc freq data=newdata;
   by EmpLocation;
   table JobCode;
   run;
ods html close;
ods listing;

/* 2: Use ODS Document to capture output  */
/* into the work.emploc document.         */
title 'Use ODS DOCUMENT to persist output';
ods document name=emploc(write);

Proc tabulate data=newdata ;
by Emplocation;
var salary;
class jobcode;
table jobcode,salary*(min mean max std);
run;
```

```
proc freq data=newdata;
Title 'Proc Freq Step';
   by EmpLocation;
   table JobCode;
   run;

ods document close;

/*-- 3: List contents of work.emploc --*/
/*-- new ods document copy of output --*/
/*-- hierarchy. Note how output is   --*/
/*-- organized into "directories."   --*/
title 'Contents Of Work.Emploc';
proc document name=emploc;
  list/levels=all;
run;
quit;

/*-- 4: Replay (rerender) the saved  --*/
/*-- output in the work.emploc       --*/
/*-- document as RTF.                --*/
ods rtf file = 'c:\temp\reRendered.rtf';
proc document name=emploc;
replay; run;
quit;
ods rtf close;

/*-- 5: Create document work.emploc2   --*/
/*-- to hold the reordered output.     --*/
/*-- Basically, want the proc freq     --*/
/*-- output followed by the proc       --*/
/*-- tabulate output for the same by   --*/
/*-- group.  Note that the original    --*/
/*-- order of the procedures was       --*/
/*-- TABULATE and then FREQ.           --*/
/*-- Use the 'brute force' method to   --*/
/*-- write copy statements.            --*/

proc document name=emploc2(write);
 make Loc;
 run;
 setlabel Loc
    'Reordered PROC FREQ and MEANS output';
 make CARY;
 run;
 setlabel CARY 'Location=CARY';
 dir CARY;
copy
\Work.Emploc\Freq#1\ByGroup1#1\Table1#1\onewayfr
eqs#1 to ^;
copy \Work.Emploc\Tabulate#1\ByGroup1#1\Report
     to ^;

 dir ^^;  /* go up to the parent directory */
 run;
 make HONGKONG; /* make a new directory */
 run;
 setlabel HONGKONG 'Location=HONG KONG';
 dir HONGKONG; /* go to this new directory */
 copy
\Work.Emploc\Freq#1\ByGroup3#1\Table1#1\onewayfr
eqs#1 to ^;
 copy \Work.Emploc\Tabulate#1\ByGroup3#1\Report
to ^;
 dir ^^;
 run;
 run;
quit;

/*-- 6: List the contents of the new   --*/
/*-- document work.emploc2.            --*/
```

```
title 'Contents Of Work.Emploc2';
proc document name=emploc2;
  list/levels=all;
run;
quit;

/*-- 7: Show "after" with rearranged output. --
*/
ods html path='c:\temp\' (url=none)
    frame="aft_frm.html"
    body="aft_bod.html"
    contents="aft_toc.html";

proc document name=emploc2;
replay; run;
   quit;

ods html close;
dm "wbrowse 'c:\temp\aft_frm.html' ";

ods listing;
```

PROC DOCUMENT is an interactive procedure. This dictates the need for the "quit" statements in the code above. Did you notice the curious caret '^' characters? One caret represents the current directory, as '.' does in Unix, just as two carets represents the parent directory, or '..' in Unix. Unfortunately, we were unable to use dots as the notation due to the grammar parser in SAS; so the caret '^' was chosen instead.

You will continue to see new features and capabilities emerge for the ODS DOCUMENT. The need for wildcards has already been noted, and not all procedures can be stored into the DOCUMENT. Proc Report is the sole procedure lacking DOCUMENT integration. (And yes, the author is the support for Proc Report, so I guess I should stop writing SUGI papers and get more development done.) ☺

One feature that I have not mentioned at all regarding the DOCUMENT is that it also has a graphical user interface. Due to lack of space in this paper, however, I am not going to write about it. Please refer to the documentation for Version 9.0.

PRINTER AND RTF NEW FEATURES

RTF ONLY

First let me digress just a bit from the purpose of this paper which is to talk about new features of ODS for Version 9.0. We have received some questions from users asking specific questions regarding RTF such as why don't we have vertical measurement in RTF, why don't we keep the date and time that the SAS program was run, and why do we put the titles and footnotes into the section header and section footer areas. The answers to these questions come down to what the ODS team had originally conceived would be the purpose of RTF -- to give users an editable destination. We had thought that users would be using the RTF destination so that they could run their SAS programs, get their SAS output into RTF, and then edit the file using Microsoft Word, adding text or whatever else they needed to do. Since we assumed that the RTF file would be edited, there was no sense in performing vertical measurement and using the CPU cycles since after more information was added, the document would need to be repaginated. Not performing vertical measurement also had to do with the reason that we placed the titles and footnotes in the header and footer sections. Since we do not do vertical measurement due to our thought process regarding editing, if we did not place the titles/footnotes in the header/footer sections, the titles and footnotes would not be repeated on each page since we don't know where the page breaks are. We also assumed that since the document would be edited, the date and time would need to be the last date and time the document was opened, saved, or printed.

So now we have heard your comments (complaints?) and we are currently investigating creating a vertically-measured RTF destination. We are also adding some options to allow users to get around some of these decisions and assumptions that we have made. One of the first options, BODYTITLE, was actually added late in the Version 8.2 cycle. This was to put the titles and the footnotes in the body section of the document instead of into the header/footer sections. This option was experimental in Version 8.2 (and for good reason since it had quite a few interactions with other options) and will stay experimental in Version 9.0. A second option, SASDATE, is described below.

The ODS team would like to invite you to let us know how you use the RTF destination: do you use it as a destination in which you will be editing (so you do not need vertical measurement), or do you need the vertical measurement and the titles/footnotes within the body of the document? Email us as ods@sas.com and let us know or also let us know if there is something regarding the current RTF that you are having to workaround due to our "RTF is an editable destination" decision.

SASDATE

This option is in response to one of those comments that we received as mentioned above. In prior versions of SAS, when users requested the RTF destination, ODS inserted the special DATE macro in the document. If you were to open the RTF document in Notepad, you would see something like:

```
   {DATE \\@ "hh:mm dddd, MMMM dd, yyyy " }
```

This DATE macro would insert the current date and time when the document was opened or when the document was printed. What this means is that the date and time when you actually first ran your SAS program was now lost. This information, however, (as we found out) is very important. The solution that we have starting with version 9.0 is to use the SASDATE option when you open the RTF destination. This option tells the RTF destination to do as the other destinations have done which is to get the date and time and place that string inside the RTF document. This ensures that no matter how many times you open or print the document, that date and time will stay the same as when the SAS program was initially run. Since RTF is the only destination that was doing this (since it was taking advantage of the DATE macro), this option is only needed for RTF.

```
ODS RTF file = 'foo.rtf'  SASDATE;
```

PAGE X OF Y

A common request for users using the RTF destination has been to be able to generate "Page X of Y" in their documents. This has been possible before Version 9.0 with a little bit of raw RTF code embedded usually inside the title or footnote statement. However, we got so many requests for this little bit of code, ODS decided to make this a little bit easier for you by now having a keyword that you specify that will embed that special RTF code for you. This PAGEOF option is usually used in the title or footnote statement as inline style code, similar to SUPER and SUB.

```
ods escapechar  = '\';
title   'This document will have page x of y  '
       j=r  'Page \{pageof}';
ods rtf file='foo.rtf';
proc print data=sashelp.class;run;
ods rtf close;
```

CHANGING ORIENTATION WHILE RTF IS OPEN

In Version 8.2, you could set the orientation for your output, but the orientation had to be set BEFORE the destination was opened. This, as you could well imagine, caused quite a few calls to technical support since users were trying:

```
ods rtf file = ‘ChgOrient.rtf’;
options orientation = landscape;
/* favorite procedural code here */
run;
ods rtf close;
```

only to be VERY disappointed when the output was not in landscape orientation since the RTF destination was already open when the orientation was set. (This, by the way, is the behavior for both RTF and the PRINTER destinations at Version 8.2).

Our resident RTF expert, Mr. Wayne Hester, worked on the RTF destination for Version 9.0 to allow you to be able to set, and even change, the orientation for your output after the destination has been opened. This might not be quite as intuitive as you would have hoped, but it’s the only way that it is going to work. The trick is that after you set the orientation, you must have a call to the RTF destination which signals RTF that it needs to go check the options again.

```
ods rtf  file = ‘ChgOrientation.rtf’;
/* This will be in Portrait orientation since
that is the default */

proc print data = sashelp.class; run;

options orientation = landscape;

/* The option has changed, but now we need to
tell RTF that it changed */

ods rtf;

/* Now this will be in landscape */

proc print data = sashelp.class; run;
ods rtf close;
```

This feature is only valid for RTF. It will not work for the PRINTER destinations or any of the other destinations, for that matter.

PRINTER AND RTF

The next four options that I will discuss are valid for both the PRINTER and the RTF destinations and target some of the stylistic requests that we have received.

COLUMNS=

The COLUMNS= option give you the ability to output in multi-column format, just as this document is in two-column format. You can change the COLUMNS= value in between your programs without closing the destination. So for example, you could start off with COLUMNS=2, run a proc, change to COLUMNS=3, and then run another procedure. See Example 1 below for an example of setting COLUMNS=2 and a screen shot of the output.

```
ods rtf file=’columns.rtf’  columns=2;
/* insert your favorite code here */
ods rtf columns = 3;
/* insert some more of your favorite code */
ods rtf close;
```

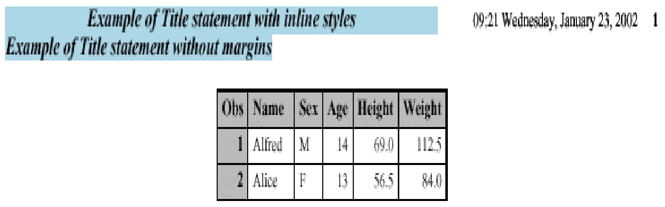
TEXT=

This TEXT= option was actually in Version 8.2, but is has been improved for Version 9.0. This option is used on the ODS RTF or PRINTER statement and places the given text string out to the RTF or PRINTER destination. This feature is handy to output a block of text before or after your tables. One of the improvements from Version 8.2 is that the text block is placed within its own table which means that you can specify a style on this same statement using the STYLE= option to stylize this text block. This STYLE option is the same STYLE= option that you currently use when you open the destination, so keep in mind that you need to use the *name* of the style, not the name of the *style element*. If you want to use a little bit of Proc Template code to modify the looks of this text block, the style element you should modify is USERTEXT. I have an example of this in Example 1. Also shown in Example 1 with the TEXT= option is the use of inline styles within the text block.

MARGINS ON CELLS

You can now set the margin values for a particular cell by using RIGHTMARGIN= and LEFTMARGIN= style attributes. These style attributes can be used either on the STYLE option as shown in Example 2, or they can be used within the inline formatting which is a popular thing to do with TITLE or FOOTNOTE statements. Here's a code snippet of what a title statement would be with inline formatting using the margin attributes.

```
title j=left '*S={background=lightblue
       LEFTMARGIN=1in RIGHTMARGIN=1in}
          Example of Title statement with
          inline styles';
title2 j=left '*S={background=lightblue} Example
of Title statement without margins';
proc print data=sashelp.class;run;
```



```
          Example of Title statement with inline styles          09:21 Wednesday, January 23, 2002    1
Example of Title statement without margins
```

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0

Figure 5: Margins & Inline Formatting in Title Stmt

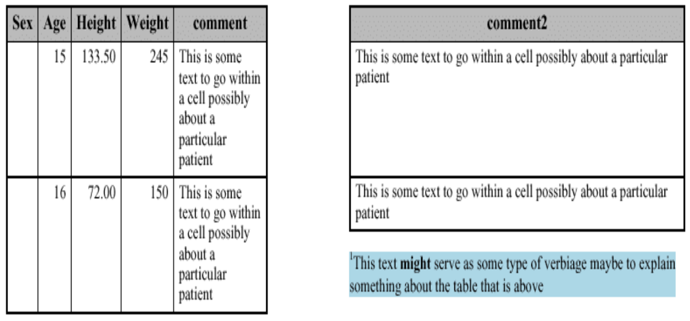
INDENTION ON CELLS

In addition to margins, you can now set the indention of a particular cell. The indention value can be either a positive or a negative number. A positive number indents that specified amount from the left margin. A negative number indents backwards that specified amount from the left margin (so the indention will be left of the left margin). Example 2 below shows the column COMMENT4 with a positive indent value and the column COMMENT5 with a negative indent value.

EXAMPLE 1: COLUMNS= AND TEXT=

Here is an example using the COLUMNS= and the TEXT= options. I have output these to both RTF and PDF, but for the sake of space, I am only including a screen capture of the PDF output from the second page.

```
/* The escapechar is used for the inline
 * style on the text= line at the bottom. */
ods escapechar = '\';
ods listing close;

/* This path statement is to set our path
 * when we do our proc template code. It
 * would normally write to sasuser.templat,
 * but I do not want to clutter everyone's
 * sasuser catalog. */
ods path (prepend) work.templat(update);

/* Change the style of the textbox so the
 * TEXT= text stands out better */

proc template;
/* textstyle is the name of our new style.
 * This is what you will use on the ODS stmt
 */
define style textstyle;
parent = styles.printer;
/* usertext is the style ELEMENT name     */
/* background is the style ATTRIBUTE name */
/* lightblue is the style ATTRIBUTE value */
style usertext from usertext /
    background = lightblue;
end;
run;

title 'COLUMNS= and TEXT= ';
ods pdf file='ODSNewEx1.pdf' startpage=no
columns=2;
ods rtf file='ODSNewEx1.rtf' startpage=no
columns=2;

proc report data=sashelp.class nowd;
column sex age height weight comment comment2;
define sex    / group  ;
define age    / group ;
define height  / sum format=6.2 ;
define weight / sum;
define comment/ computed style={cellwidth=75pt};
define comment2 / computed;
compute comment / length=100 character;
comment = 'This is some text to go within a cell
possibly about a particular patient';
endcomp;

compute comment2 / length=100 character;
comment2 = 'This is some text to go within a
cell possibly about a particular patient';
endcomp;
run;

/* Invoke the destinations with the TEXT=
 * option.  The PDF destination is going to
 * use the new style and also some inline
 * formatting. Notice that the STYLE name is
 * used on the ODS statement (see Proc
 * Template code above) */
ods pdf text='\{super 1}This text
\S={font_weight=bold}might\S={} serve as some
type of verbiage maybe to explain something
about the table that is above' style=textstyle;

ods rtf text='\{super 1}This text might serve as
some type of verbiage maybe to explain something
about the table that is above';

/* Close the destinations, reset the
 * itemstore path, and turn listing back on.
 */
ods _all_ close;
ods path reset;
ods listing;
```



Sex	Age	Height	Weight	comment
	15	133.50	245	This is some text to go within a cell possibly about a particular patient
	16	72.00	150	This is some text to go within a cell possibly about a particular patient

comment2
This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient

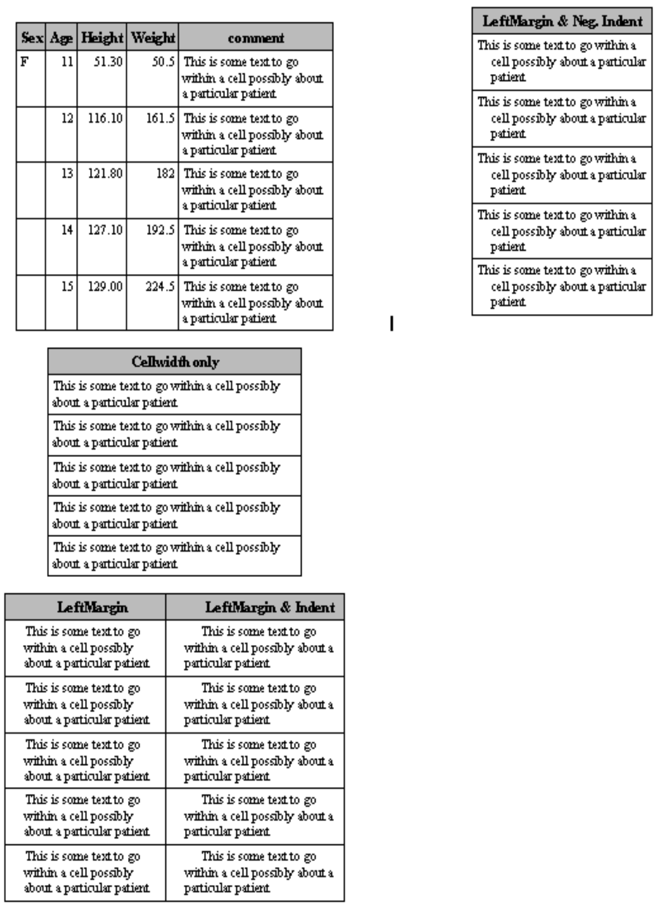
¹This text **might** serve as some type of verbiage maybe to explain something about the table that is above

Figure 6: EXAMPLE 1 OUTPUT – COLUMNS= and TEXT=

EXAMPLE 2: LEFTMARGIN=, INDENT=

Here is an example using the LEFTMARGIN= and the INDENT= options. I have output these to both RTF and PDF, but again for the sake of space, I am only including a screen capture of the RTF output.

```
ods pdf file='V9NewODS.pdf' startpage=no
columns=2;
ods rtf file='V9NewODS.rtf' startpage=no
columns=2;
proc report data=sashelp.class nowd;
where sex='F';
column sex age height weight comment comment2
comment3 comment4 comment5;
define sex    / group  ;
define age    / group ;
define height  / sum format=6.2 ;
define weight / sum;
define comment / computed ;
define comment2 / computed
style={cellwidth=200pt} 'Cellwidth only';

define comment3 / computed
           style={leftmargin=12pt} 'LeftMargin';
define comment4 / computed
           style={leftmargin=12pt indent=12pt}
           'LeftMargin & Indent';
define comment5 / computed
           style={leftmargin=12pt indent=-12pt}
           'LeftMargin & Neg. Indent';

compute comment / length=100 character;
comment = 'This is some text to go within a cell
possibly about a particular patient';
endcomp;
compute comment2 / length=100 character;
comment2 = 'This is some text to go within a
cell possibly about a particular patient';
endcomp;
compute comment3 / length=100 character;
comment3 = 'This is some text to go within a
cell possibly about a particular patient';
endcomp;
compute comment4 / length=100 character;
comment4 = 'This is some text to go within a
cell possibly about a particular patient';
endcomp;
compute comment5 / length=100 character;
comment5 = 'This is some text to go within a
cell possibly about a particular patient';
endcomp;
run;
ods _all_ close;
ods listing;
```



Sex	Age	Height	Weight	comment
F	11	51.30	50.5	This is some text to go within a cell possibly about a particular patient
	12	116.10	161.5	This is some text to go within a cell possibly about a particular patient
	13	121.80	182	This is some text to go within a cell possibly about a particular patient
	14	127.10	192.5	This is some text to go within a cell possibly about a particular patient
	15	129.00	224.5	This is some text to go within a cell possibly about a particular patient

LeftMargin & Neg. Indent
This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient

Cellwidth only
This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient

LeftMargin	LeftMargin & Indent
This is some text to go within a cell possibly about a particular patient	This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient	This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient	This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient	This is some text to go within a cell possibly about a particular patient
This is some text to go within a cell possibly about a particular patient	This is some text to go within a cell possibly about a particular patient

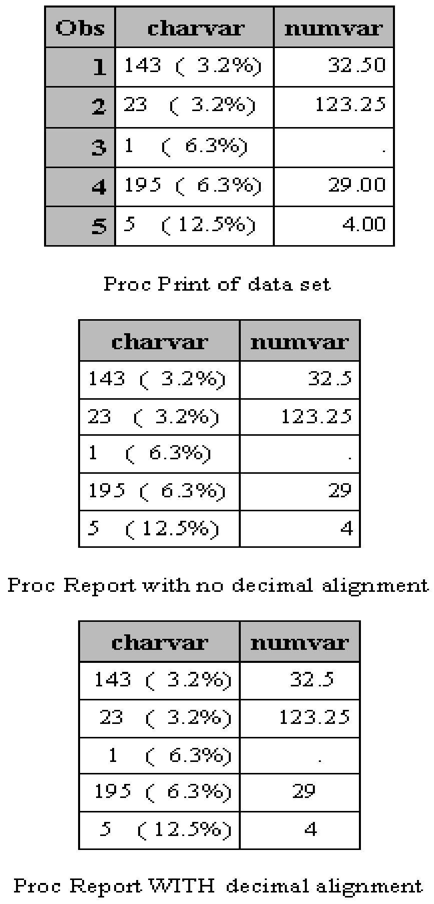
Figure 7: EXAMPLE 2 OUTPUT – Margins and INDENT=

DECIMAL ALIGNMENT

This feature has been long-awaited by many, especially for RTF output. You can now align the values in a column on the decimal boundary. This works for both numeric and character data. For the character data, it will treat the first period ('.') as the decimal boundary. This feature is implemented as a new attribute value D for the style attribute JUST (or justification). Since this is merely a new value for the justification style attribute, you can use this on the STYLE option or in your inline styles. Here's an example using Proc Report and the RTF destinations. Notice in the Proc Report example that uses the decimal alignment that the missing value in the NUMVAR column is also aligned with the decimals.

```
ods listing close;
ods path (prepend) work.templat(update);

proc template;
define style textstyle;
parent = styles.rtf;
style usertext from usertext / just = c;
end;
run;

data dectest;
input  charvar $15. numvar ;
cards;
143   (  3.2%)  32.5
23    (  3.2%)  123.25
1     (  6.3%)  .
195   (  6.3%)  29
5     ( 12.5%)  4.0;
```

```
title;
ods rtf file='dectest.rtf' startpage=no;
proc print data=work.dectest;
run;
ods rtf text='Proc Print of data set' style =
textstyle;

proc report data=work.dectest nowd;
column charvar numvar;
define charvar / display;
define numvar  / display;
run;
ods rtf text='Proc Report with no decimal
alignment' style=textstyle;

title 'Proc report with format=decimal on
column';
proc report data=work.dectest nowd;
column charvar numvar;
define charvar / display style(COLUMN)={just=d};
define numvar  / display style(COLUMN)={just=d};
run;
ods rtf text='Proc Report WITH decimal
alignment' style=textstyle;

ods rtf close;
ods path reset;
```



Obs	charvar	numvar
1	143 (3.2%)	32.50
2	23 (3.2%)	123.25
3	1 (6.3%)	.
4	195 (6.3%)	29.00
5	5 (12.5%)	4.00

Proc Print of data set

charvar	numvar
143 (3.2%)	32.5
23 (3.2%)	123.25
1 (6.3%)	.
195 (6.3%)	29
5 (12.5%)	4

Proc Report with no decimal alignment

charvar	numvar
143 (3.2%)	32.5
23 (3.2%)	123.25
1 (6.3%)	.
195 (6.3%)	29
5 (12.5%)	4

Proc Report WITH decimal alignment

Figure 8: DECIMAL ALIGNMENT OUTPUT

CONCLUSION

I hope that after reading all this you have a good appreciation for the work that we have tried to do in ODS for Version 9.0. The impact and usefulness of some of the features will, no doubt, be apparent to some immediately, as MARKUP and TAGSETS has already proven useful for those users that have already written their own TAGSETS. The DOCUMENT destination's usefulness might not be as immediate, but I hope in time to start hearing from users that have found a need for this capability. Internally to SAS, we already have found the DOCUMENT to be extremely important, as we are using it extensively in the experimental STAT/GRAPH/ODS project. And finally, this paper certainly does not cover every single change to ODS for Version 9.0, but hopefully I have covered most of the larger and more interesting features that you will be able to take advantage of when you progress to Version 9.0.

ACKNOWLEDGEMENTS

Thanks to:
David Kelley for the Proc DOCUMENT verbiage.

Cynthia Zender for the Proc DOCUMENT example which I used as the basis for my example.

CONTACT INFORMATION

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

Sandy McNeill
SAS
Cary, NC 27513
Email: sandy.mcneill@sas.com
ods@sas.com
Web: http://www.sas.com/rnd/base