Paper TS06

Experiences of using ODS : moving from ASCII to RTF output

Steve Prust, Covance, Leeds, UK

ABSTRACT

This paper examines the steps necessary to change SAS® code from producing ASCII text to that capable of creating Microsoft Word compatible RTF (rich text format). The starting point is an environment using PROC REPORT where ASCII output was post-processed to use appropriate fonts, page margins, etc. The target was to develop code that would create Word documents directly from SAS with richer formatting capabilities.

This paper relates to ODS within SAS version 8.2

INTRODUCTION

The impetus for moving to RTF output was principally a user requirement to use a non-monospace font for the tables and listings. Monospace fonts, where each character takes the same width on the line, make it easy to align tables and listings. With a non-monospace font – and still with a requirement to align data – we had to find a new method.

At this time we were using PROC REPORT to generate our tables and listings. The ASCII output produced was then post-processed to create Microsoft Word files. This seemed to fit well with the new ODS opportunities offered by SAS version 8.2 (the first version to fully support version of ODS) especially the RTF component.

Other considerations were that ODS appeared to offer a gateway to newer technologies such as HTML and we were also keen to investigate what new formatting capabilities could give the reader.

RTF OUTPUT

Creating RTF output is a three step process :

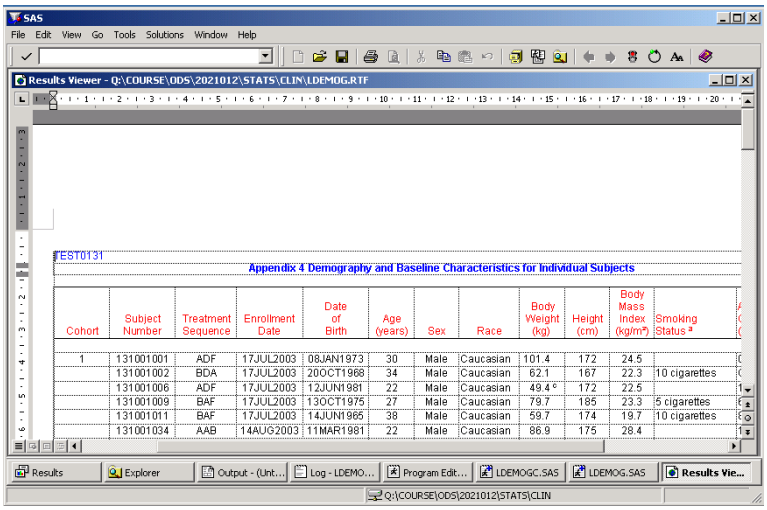
a) tell SAS you want to send output to an RTF file
b) generate output (e.g. via a PROC PRINT or PROC REPORT)
c) close the RTF destination

e.g. :

```
ods rtf file="c:\mytest.rtf";
proc print data=test;
run;
ods rtf close;
```

When the RTF file is opened SAS opens a file (note that this file is opened with the exclusive use option, so no other resources will be able to use it until the SAS program frees the file via the “ods rtf close” statement). If a file already exists with the same name this isn’t a problem as SAS will overwrite it.

When SAS has generated and closed RTF output then SAS can automatically view the output (the switch for whether or not this happens is found in *Tools/Options/Preferences – Results* tab – *View results as they are generated* check box). Viewing the output is done by SAS starting an instance of Microsoft Word as a sub-window. For example :



ODS TEMPLATES

We were aware that we needed to create our own ODS templates in order to control the appearance of our output. Initially we found templates quite difficult both in terms of how they work and also the ergonomics of how they work in SAS, e.g. :

- To view a template it is necessary to go to the Results window, then from the menu select the command Template, then navigate a template tree not the most obvious or easy method.
- When we tried to understand how templates worked there were some new concepts of inheritance – which although elegant we found in practice tended to mask how things were set up.
- Lastly when editing PROC TEMPLATE code in enhanced editor the editor seemed to be warning us that our code was invalid.

After much trial and error we created a basic template. Trial and error was necessary as there were many aspects of PROC REPORT output could be controlled by either the template or by PROC REPORT options and there were sometimes better ergonomics and/or side effects of pursuing one way or the other.

The table below shows our interpretation of what the principal options were :

ELEMENT OF PROC REPORT OUTPUT	TEMPLATE STYLE	SAS OPTIONS	TITLE OR FOOTNOTE STATEMENTS	COLUMN STATEMENT	DEFINE STATEMENT
Margin size	**Body** (parameters : topmargin, leftmargin etc.)	-	-	-	-
Page size and orientation	-	Papersize, orientation	-	-	-
Title and footer	**Systitleandfootercontainer**, **Systemtitle**, and **systemfooter** (parameters : font_face, font_weight, font_size, justify, color, protectspecialchars)	-	Text, justify, bold, color	-	-
Column headers	**Header** (parameters : font_face, font_weight, font_size, color, protectspecialchars)	-	-	-	Text, justify (via RTF tag) Justify via style (header)
Column headers that span multiple columns	[as for Column headers above]	-	-	Text, justify (via RTF tag), borders / underlining (via RTF tag)	-
Column values	**Data** (parameters : font_face, font_weight, font_size, justify, color, protectspecialchars)	-	-	-	Justify, pretext (can do decimal alignment via RTF tag)
Page number	**PageNo** (parameters : font_face, font_weight, font_size, color)	Pageno	-	-	-
Table borders, spacing and alignment	**Table** (parameters : cellspacing, cellpadding, frame, rules, just)	-	-	-	-

Table 1 – Elements of PROC REPORT and how they may be controlled

A BASIC TEMPLATE FOR PROC REPORT

We created code for a basic template as follows :

```
ods path studylib.penguin (write) sashelp.tmplmst (read); /* path for new
template and read access to system template */

proc template;
    define style Penguin;
        style titleAndNoteContainer /
            outputwidth = _undef_;
        style data /
            foreground= black
            font_face = Arial
            font_weight = medium
            font_size = 11pt
```

```
            protectspecialchars=off;
        style header /
            protectspecialchars=off
            font_face = Arial
            font_weight = medium
            font_size = 11pt;
        style Table  /
            cellspacing=1pt
            cellpadding=2
            frame=hsides
            rules=groups;
        style systemtitle /
            font_face = Arial
            font_weight = medium
            font_size = 11pt
            protectspecialchars=off;
        style systemfooter /
            font_face = Arial
            font_weight = medium
            font_size = 11pt;
        style column /
            protectspecialchars=off;
        style notecontent;
        style PageNo /
            foreground=white;
        style SysTitleAndFooterContainer;
        style body "Set page margins" /
             bottommargin = 1in
             topmargin = 1.25in
             rightmargin = 1in
            leftmargin = 1.44in;
        end;
quit;

/* reset path to read system templates only */
ods path sashelp.tmplmst (read);
```

(note – the above parameters will be discussed in more detail later)

TEMPLATE INHERITANCE

The inheritance concept used to create a mechanism to have a general project template and then adjust as necessary for individual programs. We did this by using code such as this:

```
ods path work.&progname (write) studylib.penguin (read); /* write path→work */
proc template;
  define style &progname;
     parent=penguin;     /* additional and changed options ... */
        style data from data /
            font_size = 9pt;
        style header from header /
            font_size = 9pt;
        style systemtitle from systemtitle/
            font_size = 9pt;
        style systemfooter from systemfooter/
            font_size = 9pt;
        style systitleandfootercontainer  /
          outputwidth=100%;
        style table from table /
          outputwidth=100%;
  end;
run;quit;
ods path sashelp.tmplmst(read);                        /* write path → default */
```

This would then be used when we ran PROC REPORT as follows :

```
ods path studylib.penguin(read) &progname(read);  /* make templates available */
ods rtf file="output file" bodytitle style=&progname startpage=yes; /*open rtf*/
{invoke macro(s) to create output e.g. %printit;}
ods rtf close;                                              /* close rtf file */
ods path sashelp.tmplmst (read);                            /* set to default */
```

TITLES, FOOTNOTES AND SEPARATOR LINES

A LITTLE HISTORICAL EXPLANATION

Using PROC REPORT to create ASCII output gave us a problem regarding footers in that footers would always go to the bottom of the page. In cases where we had only a small amount of data on the page there would be a large gap between the end of the table and the footer. We avoided this problem by a technique of "brute-force" paging. To do this we would determine ourselves how many pages were needed for a table and set up a variable (usually called "page" that indicated which page an observation on the dataset would be placed on. We would then run PROC REPORT multiple times, once for each page. Lastly, we made this variable a grouped variable so we could use it in a compute block e.g.

```
compute after page;
```

Because of this and other site-specific techniques some of the following code may look a little unusual or constrained.

A SIMPLE REPORT

The "look" we wanted in terms of titles, footers, and separator lines was something like this :

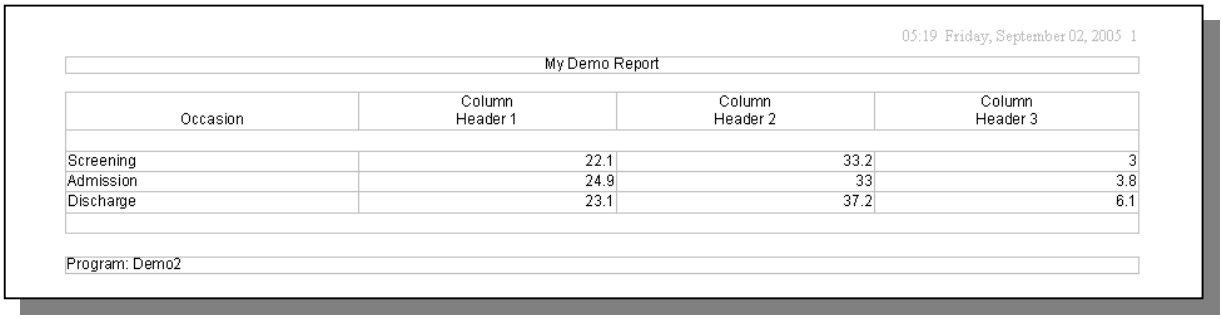
Title

	Column Header1	Column Header 2	Column Header 3
Screening	.	.	.
Admission	.	.	.
Discharge	.	.	.

Footnotes

The relevant coding in getting this output was :

- in the style parameter for table the options of *frame=hsides* and *rules=groups*
- use of compute blocks to space output before and after the main data area



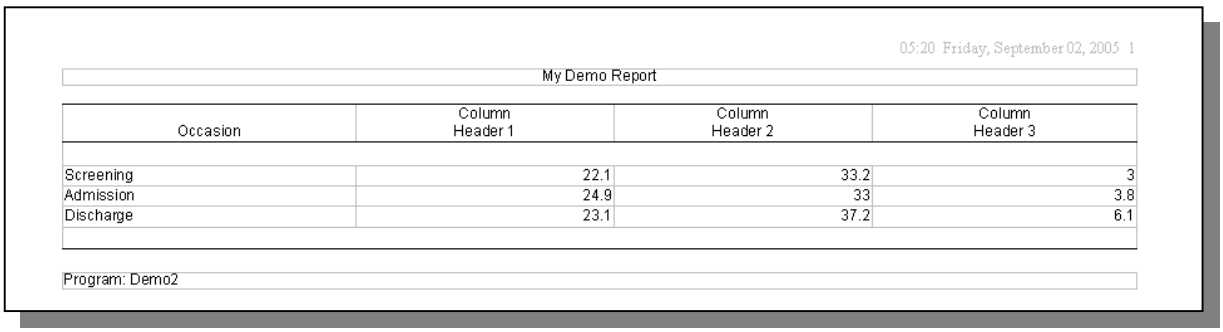
05:19 Friday, September 02, 2005 1

My Demo Report

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3
Admission	24.9	33	3.8
Discharge	23.1	37.2	6.1

Program: Demo2

Sample output without using frame=hsides and rules=groups



05:20 Friday, September 02, 2005 1

My Demo Report

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3
Admission	24.9	33	3.8
Discharge	23.1	37.2	6.1

Program: Demo2

Sample output using frame=hsides and rules=groups

[Note : the grey lines in the above examples show the boundaries of table cells in the RTF file and are not printed]

The code used to produce the above (in conjunction with the previous template and code structure) was :

```
data fred;
  mypage=1;
  occ='Screening';a=22.1;b=33.2;c=3.0;output;
  occ='Admission';a=24.9;b=33.0;c=3.8;output;
  occ='Discharge';a=23.1;b=37.2;c=6.1;output;
run;

...

%macro printit;
title 'My Demo Report';
footnote j=l 'Program: Demo2';
proc report data=fred nowindows norkeys center missing split='#';
  column mypage (occ a b c);
  define mypage / group noprint;
  define occ    / 'Occasion';
  define a      / 'Column#Header 1';
  define b      / 'Column#Header 2';
  define c      / 'Column#Header 3';
  compute before mypage;
    line ' ';
  endcomp;
  compute after mypage;
    line ' ' ;
  endcomp;
run;quit;
%mend;
```

GETTING RID OF THE DEFAULT PAGE NUMBERS

Unfortunately getting rid of the standard SAS page numbering / datestamp in the top right of the output page – which we did by using the SAS options of *nodate* and *nonumber* – causes the code to no longer work (no workaround is possible or was known about at the time of writing).

What we decided to do therefore was a fix of simply setting the relevant template style (*PageNo*) to have a colour of white (*foreground=white*). This has the slightly spooky effect of the page number / datestamp appearing to be there when viewing the output in Word (because the page / date information is stored in the header area and in Word this is shown in grey) but it disappears on printing. We did however want the header to be less obtrusive when viewing in Word and so used the SAS option of *nodate* thus leaving just a page number in this header area.

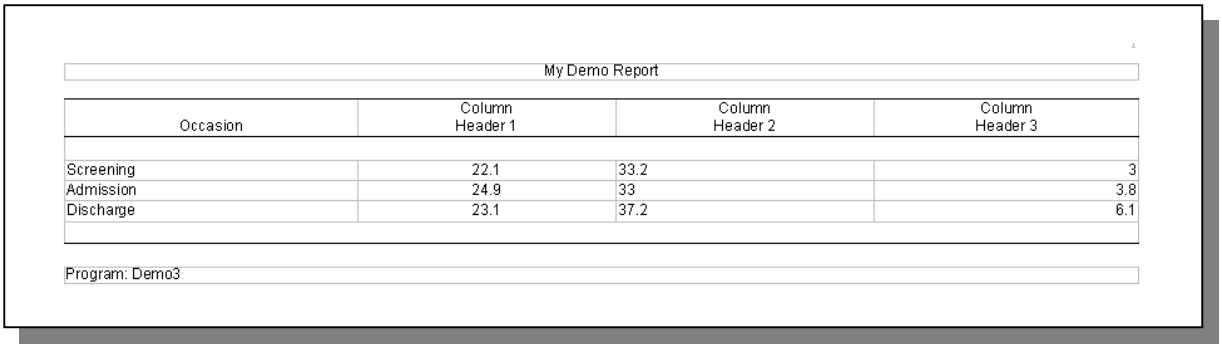
One further note on this is that this technique does have an effect on the printable page area and it may be necessary to adjust margins accordingly.

TEXT OPTIONS, SIZE AND PLACEMENT

The above example is already looking like what we want to see. Next we tackled alignment and a few text options.

COLUMN ALIGNMENT

Alignment for an entire column can be done by using *center*, *left* or *right* options on the PROC REPORT DEFINE statement. For example :



My Demo Report

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3
Admission	24.9	33	3.8
Discharge	23.1	37.2	6.1

Program: Demo3

Sample output using center, left, and right on define statements

Notice that the column headers are not affected (see table 1 above - Elements of PROC REPORT and how they may be controlled). RTF tags in the column header text can control column headers, for example :

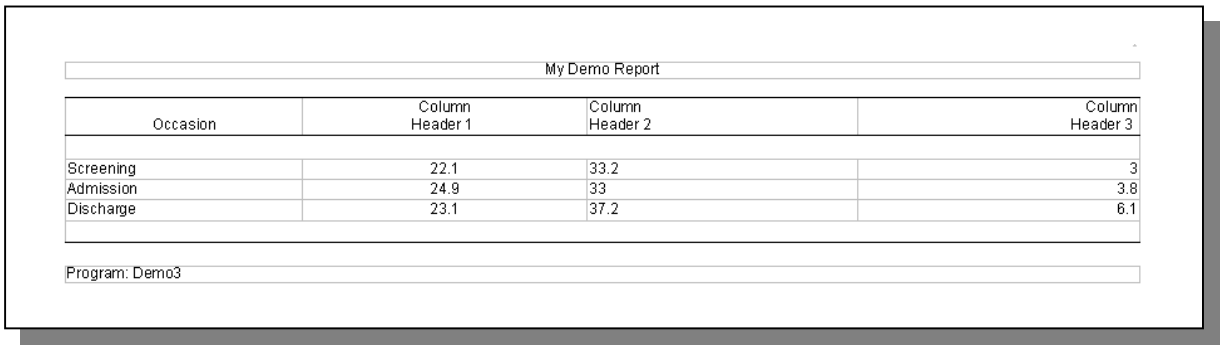
```
define a      / 'Column#Header 1 \qc' center;
define b      / 'Column#Header 2 \ql' left;
define c      / 'Column#Header 3 \qr' right;
```

Notice that in the code the RTF tag (e.g. "\qc") is placed after the text of the column header. A space precedes it – this is not necessary in this particular case but generally helps when trying to delimit RTF tags. The tag can in fact be placed elsewhere but because of using the space to delimit we have preferred it at the back of the column header text.

A more effective way to achieve the above output is to use the *style* parameter as follows :

```
define a      / 'Column#Header 1' style(header)={just=center} center;
define b      / 'Column#Header 2' style(header)={just=left} left;
define c      / 'Column#Header 3' style(header)={just=right} right;
```

either of the above two methods will give the following :



My Demo Report

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3
Admission	24.9	33	3.8
Discharge	23.1	37.2	6.1

Program: Demo3

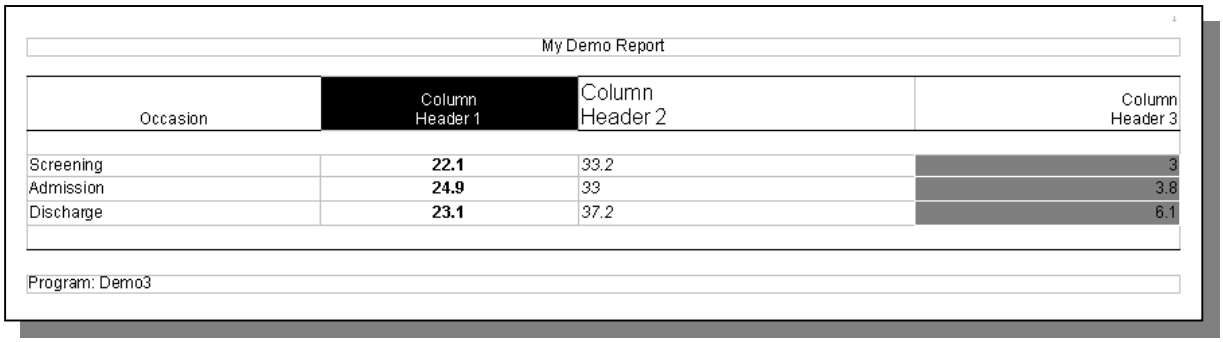
Sample output using either RTF tags or *style(header)={just= }* to align column headers

The *style*(*header*) way is generally to be preferred to the RTF tag method but later we shall see a requirement for which the RTF tag is the only method.

The style parameter can be used twice on a define statement: once for the column header and once for the column data (note the style for the header should follow the style for data). For instance :

```
define a / 'Column#Header 1' style={font_weight=bold}
         style(header)={just=center foreground=white background=black} center;
define b / 'Column#Header 2' style={font_style=italic}
         style(header)={just=left font_size=12pt} left;
define c / 'Column#Header 3' style={background=gray}
         style(header)={just=right} right;
```

Which gives :



My Demo Report

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	**22.1**	33.2	3
Admission	**24.9**	33	3.8
Discharge	**23.1**	37.2	6.1

Program: Demo3

Sample output using *style={ }* and *style(header)={ }*

SPECIFYING PARAMETERS IN THE TEMPLATE OR IN PROC REPORT STATEMENTS

Suddenly with this new functionality we were starting to have a little fun – putting columns in italics, using different colours etc. But it was also introducing a lot more parameters and we could already see that some of our conciseness in code might be lost. One question we were asking was whether it was better to put parameters in templates – where they are slightly away from the "main action" or to specify everything in PROC REPORT statements which could get verbose.

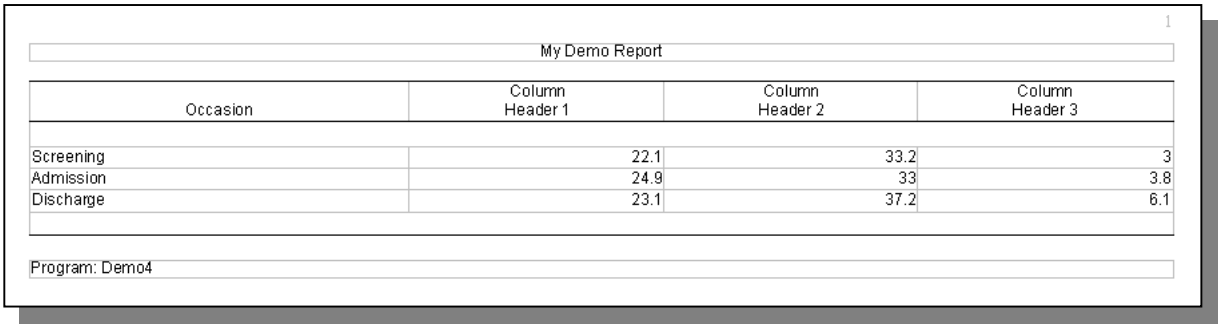
We haven't a good answer to this – other than to try and keep much of the code physically in the same area (we have some very long programs) and use good layout.

CONTROLLING THE WIDTH

Style parameters allow the width of the overall report and individual columns to be controlled. For example to control the column widths in the previous examples we might use :

```
define occ    / 'Occasion'  style={cellwidth=6.0cm};
define a      / 'Column#Header 1' style={cellwidth=4.0cm};
define b      / 'Column#Header 2' style={cellwidth=4.0cm};
define c      / 'Column#Header 3' style={cellwidth=4.0cm};
```

Which then gives



My Demo Report

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3
Admission	24.9	33	3.8
Discharge	23.1	37.2	6.1

Program: Demo4

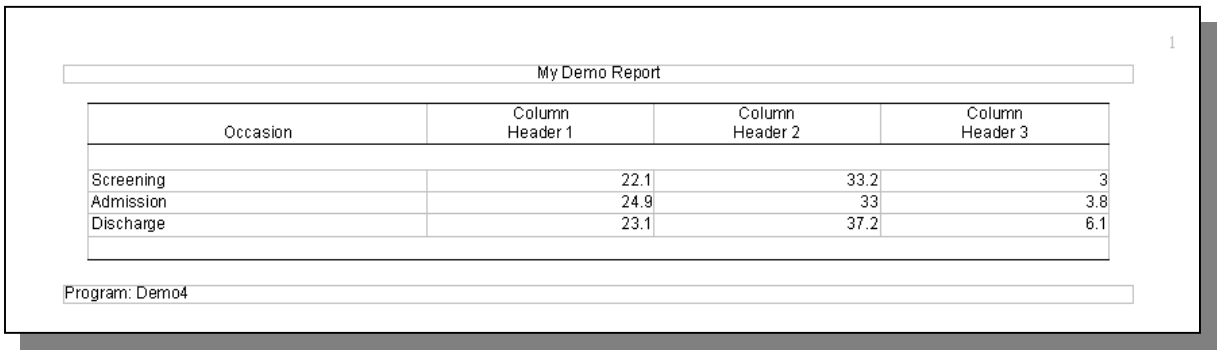
Sample output using *style={cellwidth= }*

Before the columns were allocated equal width. Now their width is based on the cellwidth options. Because the four define statements do not sum to the available width (in the above example the available width was 22.65 cm) SAS then allocates widths in what appears to be a pro-rata manner. Thus the actual widths instead of being 6.0cm followed by three columns of 4.0cm are actually 7.52 cm followed by 5.03cm.

The main body of the report as well as the title and footer section can have their widths adjusted using similar cellwidth options :

```
proc template;
  define style &progname;
    parent=penguin;
    style systitleandfootercontainer / outputwidth=92.5%;
    style table from table           / outputwidth=20.0cm;
  end;
run;quit;
```

This would change the above output to :



My Demo Report

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3
Admission	24.9	33	3.8
Discharge	23.1	37.2	6.1

Program: Demo4

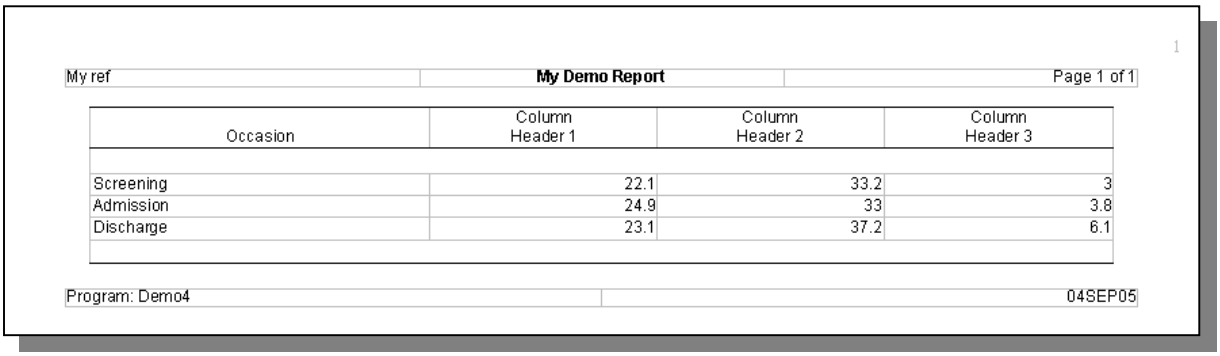
Sample output different output widths

USING MULTIPLE JUSTIFY TAGS ON HEADERS / FOOTERS

Headers and footers can be split into two or three regular parts by using the justify options e.g. :

```
title j=l 'My ref' j=c bold 'My Demo Report' j=r "Page &page of &totpage" ;
footnote j=l 'Program: Demo4' j=r "&sysdate";
```

which results in the following output



My ref | My Demo Report | Page 1 of 1

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3
Admission	24.9	33	3.8
Discharge	23.1	37.2	6.1

Program: Demo4 | 04SEP05

Sample output using multiple justification options in Header and Footer statements

PRACTICAL ALLOCATION OF CELL WIDTHS

In practice the calculation of cell widths has been one of the most time-consuming aspects of producing ODS RTF output. Previously we had calculated cell widths on the basis of number of characters – with non-monospace fonts this calculation is not available to us and thus a certain amount of guesswork results. Typically we have found that we have more space available but when space is tight we sometimes have to resort to an iterative process of adjust-run-adjust-run and so on.

RTF TAGS

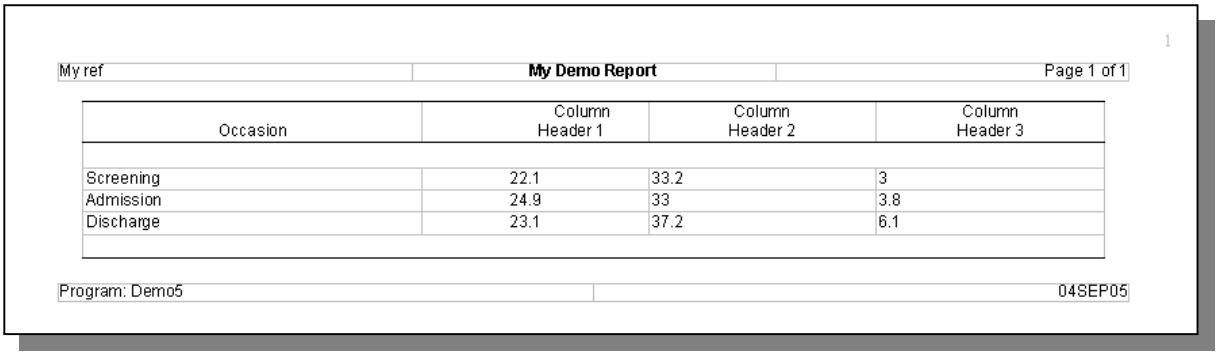
DECIMAL POINT ALIGNMENT

Using the previous techniques we were getting close to what we wanted and could see that we could enhance our output but we hadn't yet solved the problem of decimal point alignment.

The solution we found was to extend the use of RTF tags from what was shown above (column header alignment). To do this we had to use the *pretext* option, for example :

```
define a /'Column#Header 1' left style={cellwidth=4cm pretext='\tqdec\tx1134 '};
```

which gives the output :



My ref | My Demo Report | Page 1 of 1

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3
Admission	24.9	33	3.8
Discharge	23.1	37.2	6.1

Program: Demo5 | 04SEP05

Sample output using decimal point justification

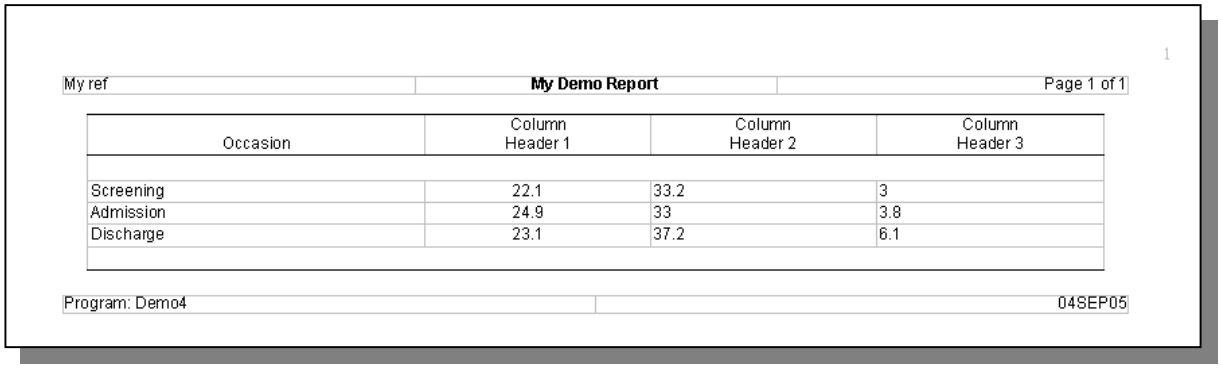
The pretext string above of "\tqdec\tx1134" contains two RTF tags. The first requests a decimal point tab mark the second states the position of the tab mark. The measurement system here is *twips*. A twips is usually defined as 1/20th of a point. We have used the calculation of one centimetre is equal to 567 twips (one inch is 1440 twips).

Unfortunately the column header is now mis-aligned with the data below. This doesn't always happen (in the above example the line break between "Column" and "Header" is what causes the mis-alignment). But can get even worse when numbers are in the early part of a column header.

The cause of the mis-alignment is the pretext being applied both to the column data and the column header areas. To avoid this we must issue a second style parameter for the column header, for example :

```
define a / 'Column#Header 1' left style={cellwidth=4cm pretext='\tqdec\tx1134 '}
                                 style(header)={just=center};
```

which gives the output :



My ref | My Demo Report | Page 1 of 1

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3
Admission	24.9	33	3.8
Discharge	23.1	37.2	6.1

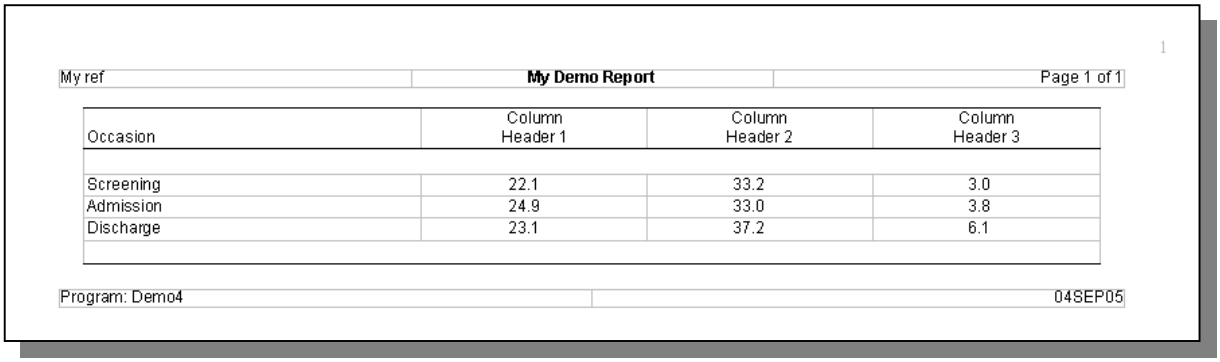
Program: Demo4 | 04SEP05

Sample output using decimal point justification and *style(header)*=

At this point we just needed to apply the techniques learnt across all the code and we had our initial goal. Thus the code was :

```
define occ / 'Occasion'      left style={cellwidth=6cm}
                                  style(header)={just=left};
define a / 'Column#Header 1' left style={cellwidth=4cm pretext='\tqdec\tx1134 '}
                                  style(header)={just=center} format=5.1;
define b / 'Column#Header 2' left style={cellwidth=4cm pretext='\tqdec\tx1134 '}
                                  style(header)={just=center} format=5.1;
define c / 'Column#Header 3' left style={cellwidth=4cm pretext='\tqdec\tx1134 '}
                                  style(header)={just=center} format=5.1;
```

And the output looked like :



My ref | My Demo Report | Page 1 of 1

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3.0
Admission	24.9	33.0	3.8
Discharge	23.1	37.2	6.1

Program: Demo4 | 04SEP05

Sample output – the initial target achieved !

But we didn't want to stop there

FLAGGING USING ODS ESCAPECHAR AND RTF

Until now we had flagged data in our listings using symbols such as *, ~, # etc simply because there was not other method available. RTF made the use of subscripted and superscripted characters possible.

The ODS escapechar statement sets up a special character that may be used later for inline RTF instructions This command needs only executing once, a suitable place for it would be immediately prior to the ODS RTF statement. For example :

```
ods escapechar='¬';
ods rtf file="…
```

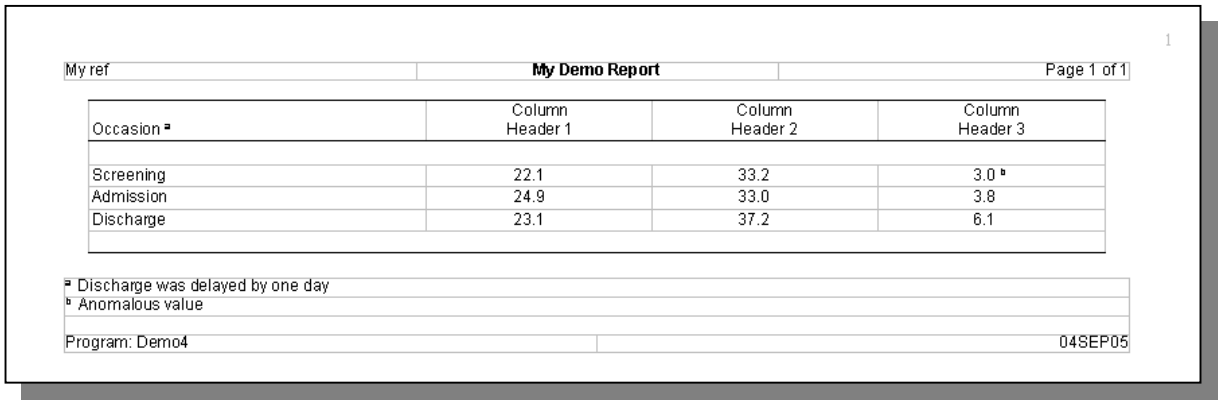
Having issued the ODS escapechar statement we could ensure that both static text in our report definition as well as dynamic text (i.e. stored as data) could use RTF functionality. We can use superscripting by adding the string "¬ {super x}" (where ¬ is our defined escape character and x the letter to be superscripted).

Consider the following changes to the code used so far :

```
data fred2;
  set fred;
  length ca cb cc $ 12;
  ca = put(a,5.1);
```

```
  cb = put(b,5.1);
  cc = put(c,5.1);
  if c <= 3 then
    cc = trim(cc) || ' ¬{super b}';
run;
...
footnote1 j=l '¬{super a} Refer to protocol' ;
footnote2 j=l '¬{super b} Anomalous value' ;
footnote3 j=l ' ';
footnote4 j=l 'Program: Demo4' j=r "&sysdate";
...
define occ    / 'Occasion ¬{super a}' left style={cellwidth=6.0cm}
                                          style(header)={just=left};
```

this will place the superscripted letters into the output like so :



My ref | **My Demo Report** | Page 1 of 1

Occasion [a]	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3.0 [b]
Admission	24.9	33.0	3.8
Discharge	23.1	37.2	6.1

[a] Discharge was delayed by one day
[b] Anomalous value

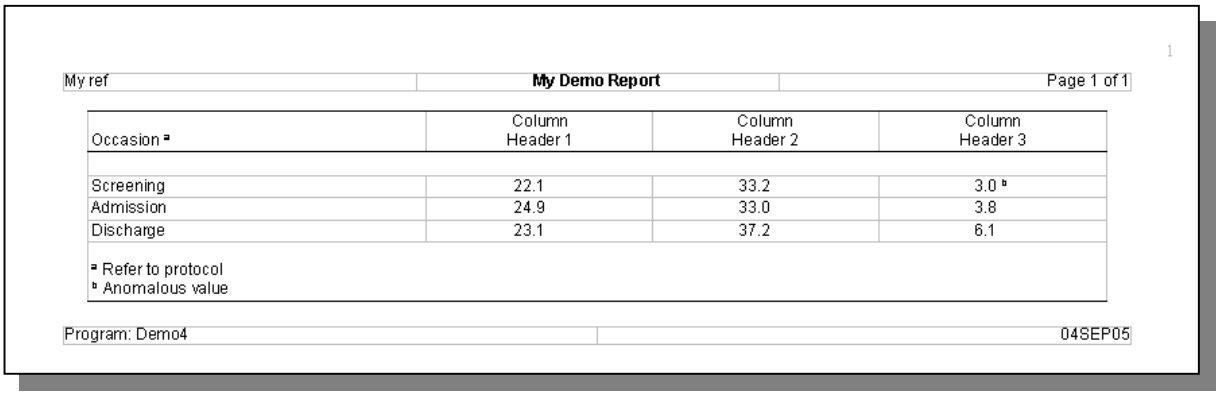
Program: Demo4 | 04SEP05

Sample output with superscripted letters

If a different place is wanted for footnotes then this code could be used (removing the previous changes to the footers) :

```
compute after mypage / style=systemtitle{just=left} ;
  line '\tab ' ;
  line '¬{super a} Refer to protocol';
  line '¬{super b} Anomalous value' ;
endcomp;
```

Which gives the output :



My ref | **My Demo Report** | Page 1 of 1

Occasion [a]	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3.0 [b]
Admission	24.9	33.0	3.8
Discharge	23.1	37.2	6.1

[a] Refer to protocol
[b] Anomalous value

Program: Demo4 | 04SEP05

Sample output with superscripted letters – alternative placement

Note that the statement “line '\tab ' ;” is simply a means of inserting a blank line. We have not been able to find a simpler means of adding line spacing (when there are other text lines in the compute block).

SPECIAL CHARACTERS

ASCII is necessarily a constrained character set and if you need an unusual character in your report it often isn’t available. With RTF we have access to a far greater range of characters

The following macro variables :

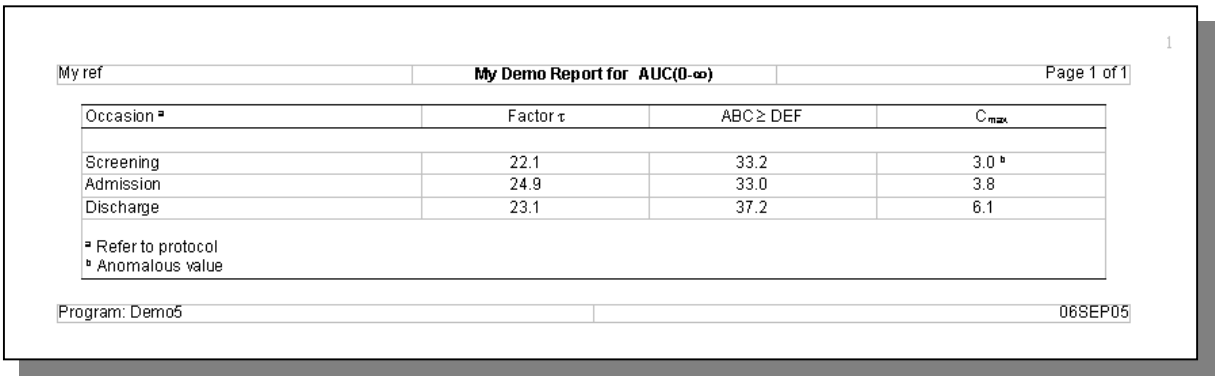
```
%let msinf      =%str(¬S={font_face=symbol}\%'a5¬S={});
```

```
%let msaucinf   = ¬S={font_face=arial} AUC(0-&msinf);
%let mstau      =%str(¬S={font_face=symbol}\%'74¬S={});
%let msge       =%str(¬S={font_face=symbol}\%'B3¬S={});
%let mscmax     = ¬S={font_face=arial} C¬{sub max};
```

which could be used like so :

```
title j=l 'My ref' j=c bold "My Demo Report for &msaucinf" ...
define ca     / "Factor &mstau" ...
define cb     / "ABC &msge DEF" ...
define cc     / "&mscmax" ...
```

Which gives the output :



My ref | **My Demo Report for AUC(0-∞)** | Page 1 of 1

Occasion [a]	Factor τ	ABC ≥ DEF	C_{max}
Screening	22.1	33.2	3.0 [b]
Admission	24.9	33.0	3.8
Discharge	23.1	37.2	6.1

[a] Refer to protocol
[b] Anomalous value

Program: Demo5 | 06SEP05

Sample output with special characters

We have found when concatenating RTF tags (for instance when an analyte contains consecutive special characters) that the RTF file can run into problems (e.g. on alternate pages the second special character does not appear). Whilst there are probably more elegant ways to deal with this we found that using a space between the tags (and therefore introducing a space into the output document) was a quick and not too dirty solution.

SEPARATOR LINES

Sometimes we need to produce separator lines elsewhere either within the table / listing or as a method of grouping column headers.

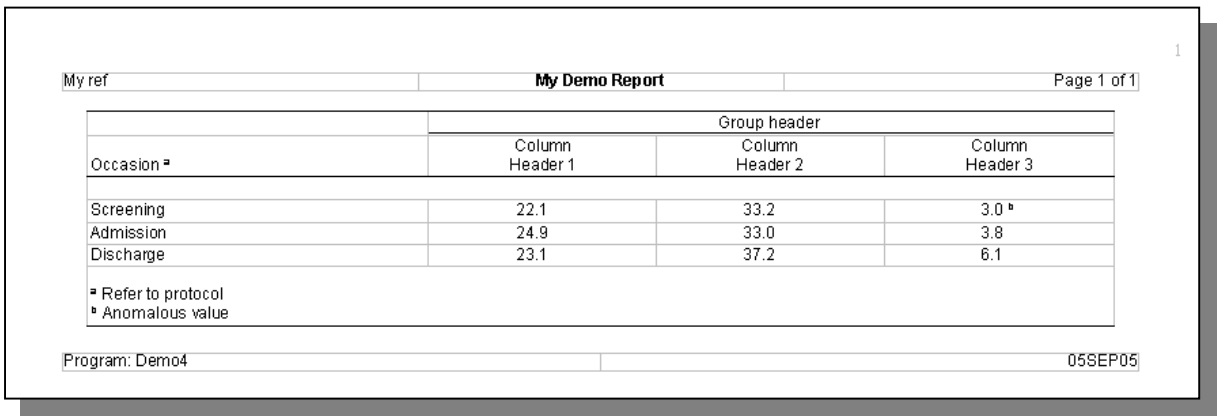
Our most common technique is to use a cell border RTF tag, we set macro variables for two useful RTF tagsets as follows :

```
%let linebot = \brdrb\brdrs\brdrw15;
%let linetop = \brdrt\brdrs\brdrw15;
```

We then use this in the column statement e.g. :

```
column mypage (occ ("Group header &linebot" ca cb cc));
```

Giving the results



My ref | **My Demo Report** | Page 1 of 1

	Group header		
Occasion [a]	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3.0 [b]
Admission	24.9	33.0	3.8
Discharge	23.1	37.2	6.1

[a] Refer to protocol
[b] Anomalous value

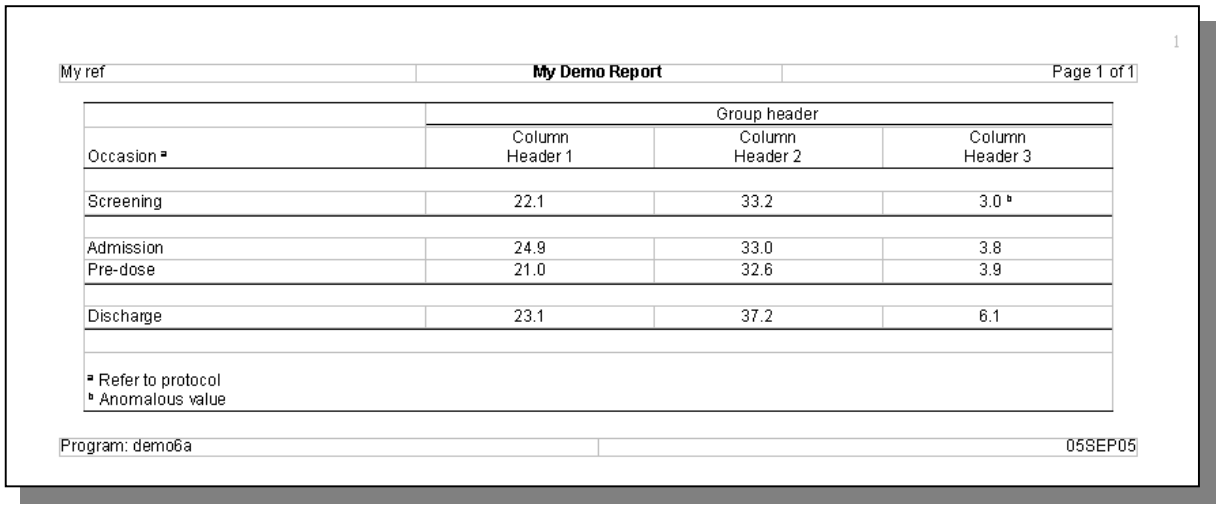
Program: Demo4 | 05SEP05

Sample output with underlining for a grouped header

This also had the advantage of not using up additional space on the page. The technique can also be used to do separator lines in other areas. For example, by adding an appropriate variable that can be used for a compute block then lines can be added in the report :

```
compute after period / style=systemtitle;
  line "&linetop" ;
endcomp;
```

Giving the output :



My ref | **My Demo Report** | Page 1 of 1

	Group header		
Occasion [a]	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3.0 [b]
Admission	24.9	33.0	3.8
Pre-dose	21.0	32.6	3.9
Discharge	23.1	37.2	6.1

[a] Refer to protocol
[b] Anomalous value

Program: demo6a | 05SEP05

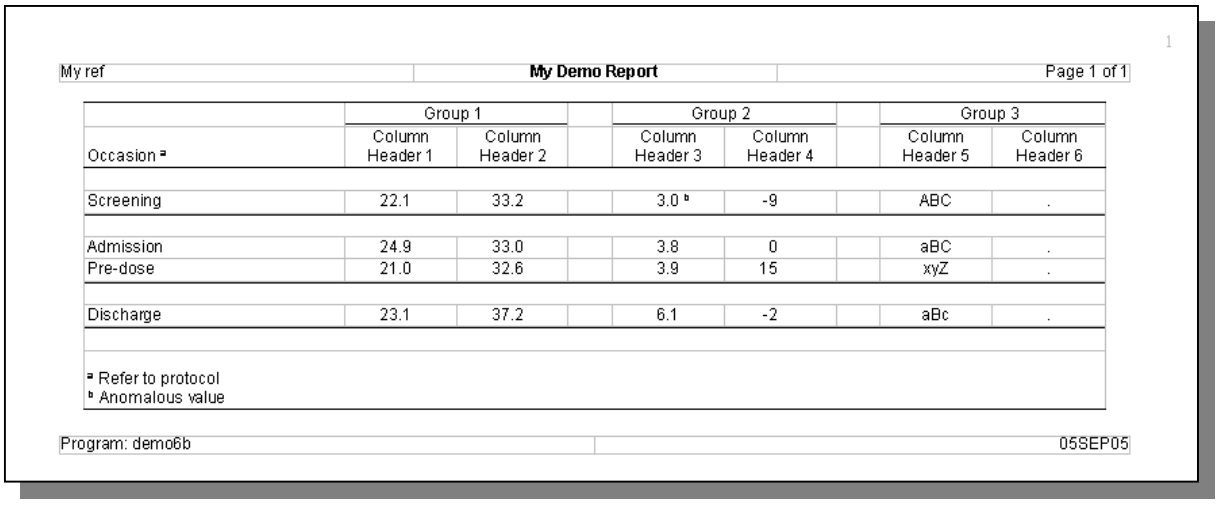
Sample output with extra separator lines

(With a little ingenuity we could also use this technique to replace the lines produced by the *frame=hsides* and *rules=groups* seen earlier.)

For group headers we found that if we had two or more sets of grouped column headers that we needed to physically separate them using a dummy variable. Unless we did this we would simply find the group headers merged together. For example :

```
column mypage period (occ ("Group 1 &linebot" ca cb) gap
                          ("Group 2 &linebot" cc cd) gap
                          ("Group 3 &linebot" ce cf) );
define gap    / ' ' style={cellwidth=0.1cm};
```

With the result



My ref | **My Demo Report** | Page 1 of 1

	Group 1		Group 2		Group 3	
Occasion [a]	Column Header 1	Column Header 2	Column Header 3	Column Header 4	Column Header 5	Column Header 6
Screening	22.1	33.2	3.0 [b]	-9	ABC	.
Admission	24.9	33.0	3.8	0	aBC	.
Pre-dose	21.0	32.6	3.9	15	xyZ	.
Discharge	23.1	37.2	6.1	-2	aBc	.

[a] Refer to protocol
[b] Anomalous value

Program: demo6b | 05SEP05

Sample output with multiple grouped column headers

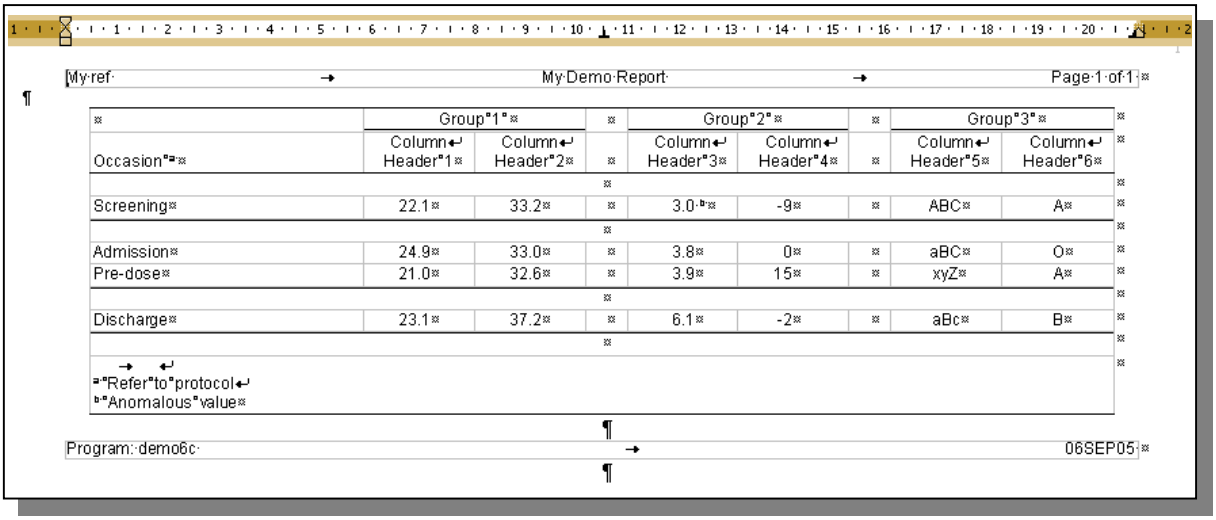
OTHER RTF TAGS

The above output is perhaps over-complicated with all the lines etc.. It also has a difficulty with how the example of how multiple justification options has been carried forward. Whilst “My Demo Report” fits nicely into the area available above as soon as the title gets wider than the available space some nasty things happen with wrapping occurring and lots of white space left.

To deal with this and for other purpose we have been using other RTF tags for tab stops. For example :

```
title j=l "My ref \tab My Demo Report \tab Page &page of &totpage \tqc\tx5950
\tqr\tx11900" ;
footnote j=l "Program: &progname \tab &sysdate \tqr\tx11900";
```

Which results in :



Sample output with tabs set in the title and footer areas

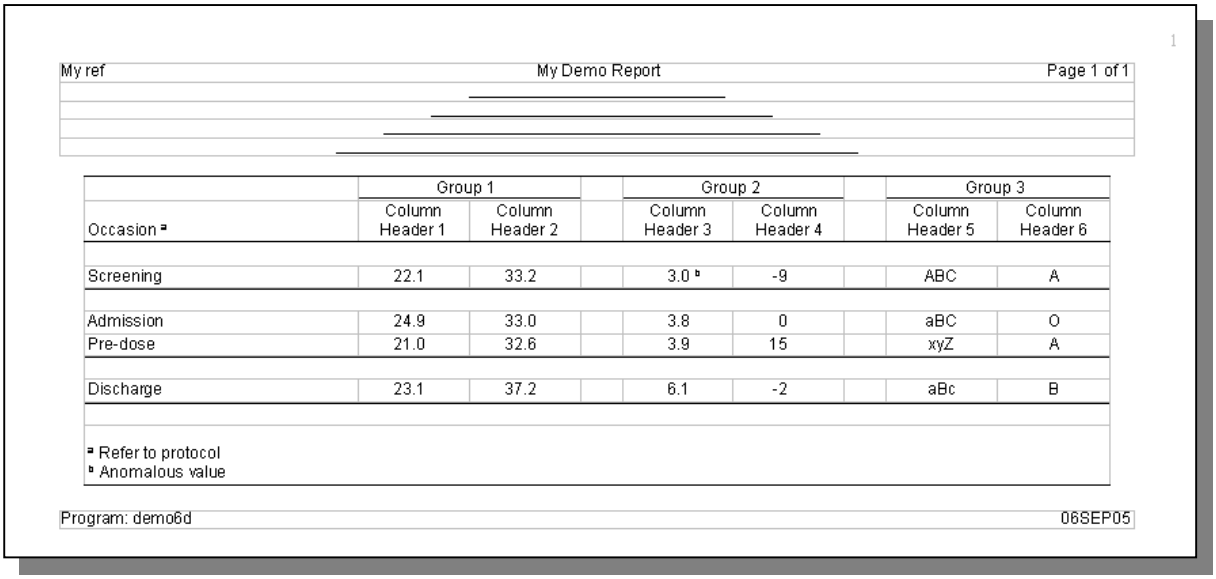
In the above screenshot the Word button marked with the pilcrow symbol (¶) has been clicked to show non-printing characters. In the title and footer areas the tab marks are shown as arrows. With the cursor on the title the Ruler bar shows the two tabs set by the RTF tagsets (i.e. \tqc\tx5950 and \tqr\tx11900) that is a centering tab positioned at 10.5cm and a right justify tab set at about 21.8cm .

Perhaps the smart thing to do with the header is simply move the title onto the title2 line !

The tabs technique can be used with dot leaders to produce a variety of interesting effects, for example the additional of the following code

```
title2 j=l "\tab \tab \tql\tx4500 \tlul\tx7400" ;
title3 j=l "\tab \tab \tql\tx4000 \tlul\tx7900" ;
title4 j=l "\tab \tab \tql\tx3500 \tlul\tx8400" ;
title5 j=l "\tab \tab \tql\tx3000 \tlul\tx8900" ;
```

produces this output :



Sample output with tabs set in the title and footer areas

One could use the dot leaders technique – if you wanted – to perform underlining of column headers but in practice we found it required too much measurement.

There are many more things tat can be done with RTF tags. For a full reference see the Microsoft web site (http://msdn.microsoft.com/library/en-us/dnrtfspec/html/rtfspec.asp)

CONDITIONAL FORMATTING

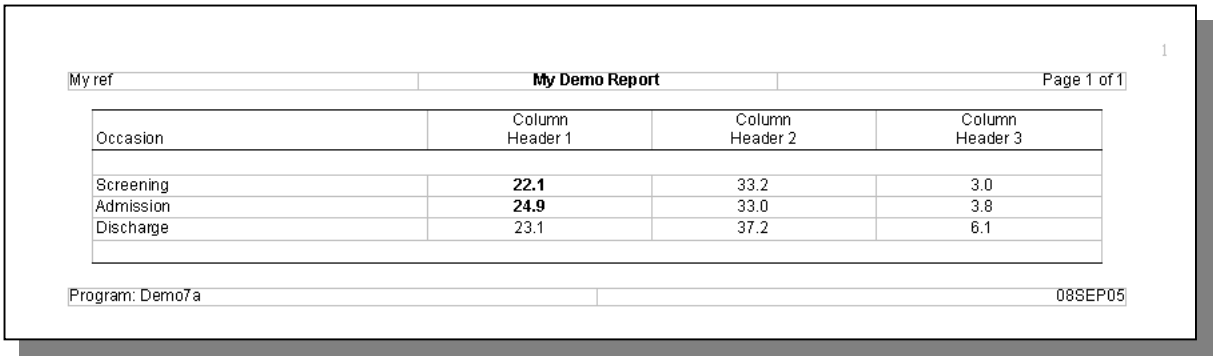
We have found two or three conditional formatting techniques, whilst powerful they need care in use.

FORMATS FOR STYLE PARAMETERS

Firstly formats can be used in place of style parameters. With the following code

```
proc format;
  value arange low-22.5   = 'bold'
               22.5<-24.5 = 'medium'
               24.5<-high = 'bold';
run;
...
define a / 'Column#Header 1' left
            style={cellwidth=4cm pretext='\tqdec\tx1134 ' font_weight=arange.}
            style(header)={just=center} format=5.1;
```

you can get the result :



1

My ref	My Demo Report	Page 1 of 1

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	**22.1**	33.2	3.0
Admission	**24.9**	33.0	3.8
Discharge	23.1	37.2	6.1

Program: Demo7a	08SEP05

Sample output with conditional formatting on individual values

This method can be used for other style characteristics such as background, font style etc. However it does only apply to the column in question. If you wish to conditionally format one column on the basis of another a different approach must be used.

FORMATTING MULTIPLE COLUMNS

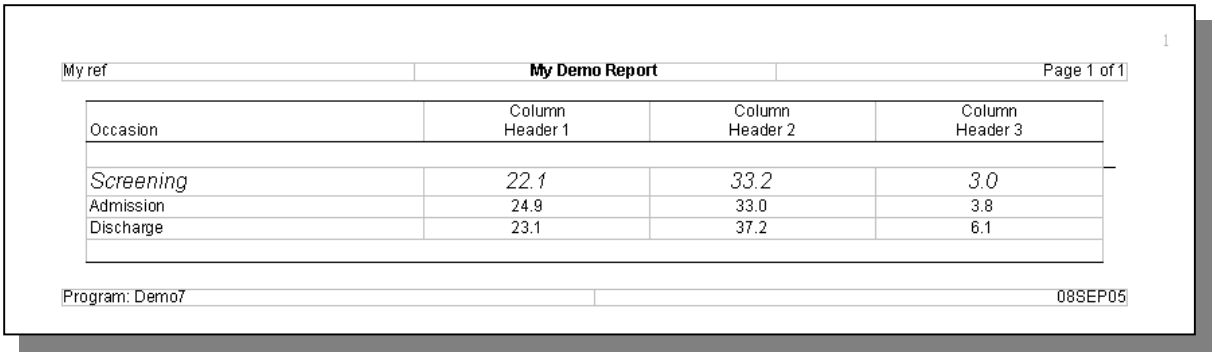
Using a compute block and the CALL DEFINE statement will allow other columns to be changed. The compute block has to have a variable that can be tested, in the following example this is done by augmenting the previous define statement :

```
define occ  /  ...   group order=data;
```

and this together with the following compute block

```
compute occ;
  if occ = 'Screening' then do;
    call define("_C2_","STYLE","style={font_style=italic font_size=12pt}");
    call define("_C3_","STYLE","style={font_style=italic font_size=12pt}");
    call define("_C4_","STYLE","style={font_style=italic font_size=12pt}");
    call define("_C5_","STYLE","style={font_style=italic font_size=12pt}");
  end;
endcomp;
```

will result in the following output :



My ref | My Demo Report | Page 1 of 1

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	*22.1*	*33.2*	*3.0*
Admission	24.9	33.0	3.8
Discharge	23.1	37.2	6.1

Program: Demo7 | 08SEP05

Sample output with conditional formatting for multiple columns

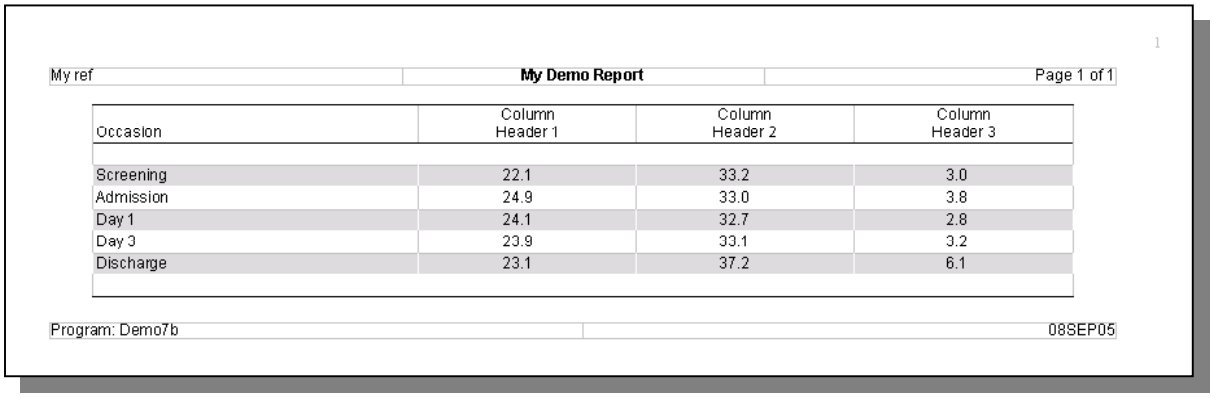
(In the above example the column variables must be accessed by the construct *_Cn_* , where n is the relative column number).

Another SAS construct that can be used in this way is *_ROW_*. The following example shows how alternate output lines can be shaded :

```
compute occ;
  linecount + 1;
  if mod(linecount,2) = 0 then
    call define(_row_,"STYLE","style={background=ltgray}");
endcomp;
```

(note : “linecount + 1;” creates a retained variable that is incremented each time the compute block is executed)

Giving the following output :



My ref | My Demo Report | Page 1 of 1

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3.0
Admission	24.9	33.0	3.8
Day 1	24.1	32.7	2.8
Day 3	23.9	33.1	3.2
Discharge	23.1	37.2	6.1

Program: Demo7b | 08SEP05

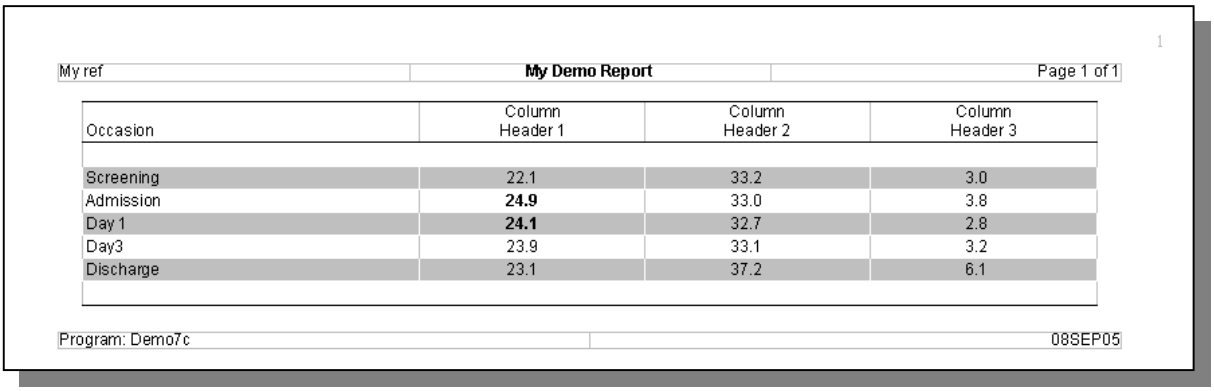
Sample output with alternate line shading

Note that if blank line spacing is used between sections of a report page then this will break-up the alternate shading of the output – possibly a good thing, possibly not depending on your preferences.

It the two preceeding examples we we fortunate in having a suitable variable upon which we could apply the group parameter (thus meaning we had a variable we could access in a compute block). In situations where this is not so it may be necessary to create and populate a variable solely for the purpose of formatting. This may be at time cumbersome.

A WARNING OF MIXING TECHNIQUES

Beware of mixing the techniques of compute block CALL DEFINE statements together with style instructions carried by formats. If the code involving the format *arange* from the earlier example is applied to the above example a most confusing thing happens. For example :



My ref | **My Demo Report** | Page 1 of 1

Occasion	Column Header 1	Column Header 2	Column Header 3
Screening	22.1	33.2	3.0
Admission	**24.9**	33.0	3.8
Day 1	**24.1**	32.7	2.8
Day3	23.9	33.1	3.2
Discharge	23.1	37.2	6.1

Program: Demo7c | 08SEP05

Sample output with incorrect formatting

As can be seen one value of 22.1 is no longer in bold, and one value (24.1) is incorrectly emboldened. It is very difficult to see why this should be happening or how it has happened.

REVIEW

We achieved our initial objective of making it possible to produce SAS output in non-monospace fonts with decimal alignment. The process of finding the solution was involved and time-consuming partly due to the inherent difficulties in the techniques of ODS RTF and partly due to that lack of documentation at that time (Since then version 9 of SAS has been available and documentation has improved).

We were able to find many opportunities of enhancing output using ODS RTF. Some techniques are either too marginal in benefit or too cumbersome to use regularly but there are several techniques that we will continue to use. In particular : special characters, sub-scripting and super-scripting, conditional formatting, cell borders.

Some of the more advanced techniques that we have found are to be used with some caution. Perhaps our difficulties are inherent in trying to do complex things between technologies (i.e. SAS and RTF).

Because of the complexity in ODS RTF and the many changes from our previous code we have found that it requires specific training for our programmers and have hence devised a specific training course for our programmers.

Our other objective of finding out more about outputting in HTML was probably not realistic. If and when we move towards HTML at lot of what we have used will be useful (e.g. templates, use of styles in PROC REPORT). But we anticipate there we be another steep-ish learning curve.

CONCLUSION

Using ODS RTF is going to be a useful tool for us and part of our core SAS skillset. We have a core set of robust features that we will use to enhance out output. More advanced features need using with caution.

We are not yet producing all our output in ODS RTF but we anticipate that this will happen. We will perhaps implement SAS version 9 before we make this move.

CONTACT INFORMATION

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

Steve Prust
Covance CRU
Apsley House
71 Wellington Street
Leeds LS1 2EQ
+(44) 113 237 3500
stephen.prust@covance .com