Paper TS03

Pagination in the ODS

John Kirkpatrick, Insight Statistical Consulting Ltd, Huntingdon, Great Britain

ABSTRACT

The ability to use non-proportional fonts in ODS output creates attractive tabular presentations but considerably increases the complexity of accurate manual pagination. In this talk, I illustrate why the problem is complex and demonstrate a simple, flexible and reliable solution that is easily integrated into existing workflows.

INTRODUCTION

Out-of-the-box procedural (or data step) output created by SAS® software is legible and informative, but it does not always carry impact. Nor, often, does it conform to various technical, organizational or regulatory requirements. To satisfy these requirements, a degree of manual control is required. When creating an attractive, informative document that is compliant with standards, manual control is required in two broad areas: the appearance of various page elements (such as the font and point size of column titles, data elements and page furniture such as page numbers and footnotes); and control of the positioning of these elements.

It has always been possible to produce RTF, PDF, PostScript and other publication file formats in SAS (see, for example, Wehr (1996)). The ODS has greatly simplified the technical difficulty of producing these file formats. Coupled with standard SAS formats, ODS style templates give the user great control over the appearance of various elements of output. ODS styles and SAS system options also provide some basic control over some aspects of the location of some output elements. For example, the page orientation and paper size can be defined in advance, together with the page margins. However, the control that the ODS provides is coarse at best, and provides absolutely no data-dependent feedback to the user. For example, although it's possible to determine in advance how many lines of output will appear on a page, the ODS will not tell you how many observations will fit on the page when the length of some data values causes text wrapping within the cells of your output table.

In this paper I will outline the approach that I have adopted to overcome these shortcomings of the ODS in my day-to-day work as a consultant statistician. I have used the algorithms described for over three years and in projects for many clients. They are accurate, reliable and flexible.

THE PROBLEM OF PAGINATION IN THE ODS

One of the principal problems with manually formatting an output document is that of pagination: exactly where do the page breaks go? When writing to the list file, whether or not the list file is subsequently converted to another format, working out where the pages break is relatively straightforward: the list file is a plain text ASCII file, so the page can be regarded as a grid of character cells of fixed size, and determining the size of any individual piece of output, and hence the amount of output that can fit on a page, reduces simply to counting characters.

LOCATING THE PAGE BREAKS

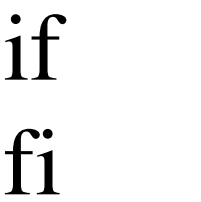
Modern document formats, such as RTF, PDF or PostScript introduce a further level of complexity to the problem in the form of proportional fonts. With proportional fonts, determining the length of a character string is no longer simply a matter of counting characters: different characters have different widths:

The quick brown fox.

The quick brown fox.

The quick brown fox.

All three examples use a point size of 12, but different fonts. Allow the point size to vary as well, and the problems multiply. Sometimes, even the order of the characters can affect the length of the string:



Both these lines are written using the Times New Roman font and a point size of 36, to make the difference more easily seen.

SAS does not provide any built-in functionality that, given a text string, a font and a point size, will calculate the space required to render the text string using the given font at the given point size. It might be possible to write some code to perform these calculations, but it would be long and complex: at the very least you would need some sort of look-up table that provided the width of every possible character when rendered in every possible combination of font and point size. I, for one, wouldn't like to try to do this, let alone maintain the code. Fortunately, as we will see, there are other ways of obtaining the required information, and of making the information available to SAS.

HANDLING THE PAGE FURNITURE

Assuming that the page breaks in your document have been accurately located, the second part of the pagination problem is handling the page furniture. That is, positioning the page titles, footers and so forth, as well as determining the number of pages in the document.

There is no doubt that pagination is difficult. SAS acknowledges this by essentially avoiding the problem. When the ODS is used to create a multi-page RTF file, SAS ignores the issue of pagination entirely, writing a single table to the RTF file, and relying on the RTF viewer, usually Microsoft Word, to handle the page layout for itself. SAS helps a little by writing the system titles and footnotes to the document header and footer sections, but that's about it. When viewing the document, tell tale signs are, for example, on the interior pages, the table has no top and bottom borders.

SAS Institute admit they put little effort into the control of pagination in the RTF destination. As Sandy McNeil said at SUGI 27, "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 don't do vertical measurement we don't know where the page breaks are." (McNeil (2002)).

THE PROBLEMS SAS CAUSES US

The half-hearted approach that SAS Institute initially adopted with respect to pagination in the ODS and the RTF destination in particular creates several problems for us. For a start, the decision to place the titles and footnotes in the document header and footer sections means that, should the RTF file be imported as a child of a master document, any running headers and footers created in the master document are obliterated. This is unhelpful.

True, SAS Institute quickly recognized the shortcomings of their decision and did their best to ameliorate the problem with the `BODYTITLE` option, but this is not without problems itself. With single page tables, it's likely that the titles and footnotes are physically quite separated from the body of the table. This is aesthetically unappealing. With multi-page tables, in SAS version 8, the titles appear on each page, but the footnotes only appear on the last page. Curiously, the bug here is the appearance of the titles, not the non-appearance of the footnotes (SAS defect number S0184681). This has been "fixed" in SAS 9, and now the titles appear on the first page, the footnotes on the last page and the interior pages are naked.

There is also a bug caused by the interaction of the `BODYTITLE` ODS option and the `NODATE` and `NONUMBER` system options. In SAS version 8, RTF files created with all three of these options set cannot be opened by Microsoft Word. The workaround is to create a style definition that defines the `PageNo` and `BodyDate` style elements as follows:

```
style PageNo from PageNo/
  font_size=0.5pt
  foreground=white;
style BodyDate from BodyDate/
  foreground = white
  font_size=0.5pt;
```

The bug is fixed in SAS 9.

THE FUTURE

Ultimate control over positioning and appearance of output requires the use of a tagset. A tagset-based RTF destination would provide users with a simple method of precise control over the positioning of page furniture. A customized tagset could, for example, easily add titles as header rows to data tables. (I show how to do this in my tutorial presentation at this conference (Kirkpatrick (2005)). In SAS version 8, the limited features available to tagset creators meant that the creation of a generic RTF tagset simply wasn't possible. In SAS 9, tagsets are much more flexible and powerful, but creating a generic RTF tagset is still a massive undertaking.

The much promised RTF tagset is due to be delivered by SAS Institute in SAS 9.2. (http://support.sas.com/rnd/base/topics/odsrtf/measured.html) But even this tagset will have its limitations: the same web page reports "an option called `TABLEROWS` enables the user to create a page break after a determined number of rows are added to the table". Note that you can count table rows (observations), not physical lines of output. Therefore, the tagset is unlikely to solve the problem of predicting where page breaks will occur.

The problem of pagination in the ODS can be broken down into two components

1. Locating the position of page breaks in advance, and
2. Correctly handling the page furniture associated with each page break

In the remainder of this paper I will present a solution to each component, demonstrate it use, and discuss how it can be easily integrated with standard SAS code.

CALCULATING STRING LENGTHS AND LOCATING PAGE BREAKS

As I have already said, SAS has no in-built ability to determine the physical length of strings when they are rendered in proportional fonts. Other programming packages do. Sun's Java language and the Microsoft Foundation Class Library for C++ are but two that have single line solutions. Given that accurate solutions exist, the problem becomes one of integration with the calling SAS program. Writing a C or C++ solution would give the possibility of providing the tightest integration of the solution with SAS – via the SAS/TOOLKIT® module – but at the cost of a relatively expensive and complex solution that is also specific to a particular combination of operating system and SAS version. Java, on the other hand, is by nature independent of operating system, freely available and, thanks to the JAVAOBJ data step construct in SAS 9, almost as tightly integrated to SAS as a solution based on SAS/TOOLKIT. I have therefore adopted a Java based solution for my own work.

A PRACTICAL EXAMPLE

Consider a listing of protocol violations in a clinical trial. The listing has columns for the treatment group, subject number, visit name or data type, details of the violation, the investigator comment and a flag to indicate whether or not the violations was considered to be major. The details and comment fields are free text and their length will dictate the amount of space required to display each observation. The following data step code illustrates the use of the `Paginator` Java class:

```
DATA Violations;
  /* Initialise the Paginator javaobj.  DECLARE is executable,
     so must do it only once! */
  IF _N_ EQ 1 THEN  DO;
    DECLARE JAVAOBJ j ("srs/Paginator");
    j.callVoidMethod("setTableHeight", "9cm");
    j.callVoidMethod("setFont", "Arial", 9);
  END;
  SET Violations END=E;
  BY Drggroup PID;
  /* Inform the Paginator of the text for each cell in the row */
  j.callVoidMethod("startRow");
  j.callVoidMethod("addCell", DevDet, "7.5cm");
  j.callVoidMethod("addCell", Comment, "5cm");
  j.callVoidMethod("endRow");
  /* Get the row height and other dimensions from the paginator */
  j.callDoubleMethod("getCurrentPage", _Page);
  j.callDoubleMethod("getRowHeight", "cm", _RowHeight);
  j.callDoubleMethod("getRowHeight", "lines", _RowLines);
  j.callDoubleMethod("getSpaceRemaining", "cm", _Space);
  /* Delete the paginator */
  IF E THEN j.delete();
RUN;
```

Before the first observation in the data set is processed, the paginator must be initialised. This is done within the IF THEN block at the top of the data step. The DECLARE statement makes the paginator available to SAS, and the next two lines define the vertical height available on the page for the listing and the default font to use when printing the table.

Next, the paginator's startRow method is called for each observation in the dataset. The startRow method initialises various internal counters. In a more complex example, in which multiple observations in the dataset were to be displayed on the same row of the table, startRow could be called conditionally to allow the correct row and cell heights to be calculated. Next, cells for the violation details (DevDet) and the investigator's comment are defined, together with their widths. Although the example here always expresses dimensions in centimetres, the Paginator can also handle units expressed in inches, points or twips. As these two variables are the only two that are relevant to the height of the observation in the table, the paginator's endRow method is now called. Finally, the data set _Page, _RowHeight, _RowLines and _Space variables are populated with relevant data from the paginator. In a real example, only _Page is mandatory. The other variables are included for illustration only. Once the last observation in the dataset has been processed, the paginator can be deleted.

Note that for the _Page variable to be populated correctly, the data set must be sorted into the order in which it will be printed, and should include only those observations to be printed. There are no such order or content restrictions on the other variables derived from the paginator.

The data set now contains all the information required to produce an accurately paginated listing using the REPORT procedure:

```
OPTIONS ORIENTATION=Landscape PAPERSIZE=ISO_A4;
ODS ESCAPECHAR="^";
ODS LISTING CLOSE;
ODS RTF FILE="&outdir\violations.rtf" STYLE=styles.LandscapeA4;

PROC REPORT DATA=Violations SPLIT="*" MISSING;
  COLUMN _Page Drggroup PID Visit DevDet Comment Major
         ("Measurement" _RowHeight _Space _RowLines );
  DEFINE _Page/ORDER NOPRINT;
  DEFINE Drggroup/ORDER ORDER=INTERNAL STYLE=[CELLWIDTH=2cm] LEFT;
  DEFINE PID/ORDER ORDER=INTERNAL STYLE=[CELLWIDTH=1.5cm];
  DEFINE Visit/DISPLAY STYLE=[CELLWIDTH=3cm];
  DEFINE DevDet/DISPLAY STYLE=[CELLWIDTH=7.5cm];
  DEFINE Comment/DISPLAY STYLE=[CELLWIDTH=5cm];
  DEFINE Major/DISPLAY STYLE=[CELLWIDTH=1.5cm] LEFT;
  DEFINE _RowHeight/DISPLAY "Row height*(cm)" FORMAT=F4.1;
  DEFINE _Space/DISPLAY "Space remaining*(cm)" FORMAT=F4.1;
  DEFINE _RowLines/DISPLAY "Row lines*(lines)";
  BREAK AFTER _Page/SUPPRESS PAGE;
  TITLE "Listing of protocol violations";
  FOOTNOTE1 J=R "Page ^{pageof}";
  FOOTNOTE2 J=L ITALIC H=8pt "Program: violations.sas";
  FOOTNOTE3 J=L ITALIC H=8pt "Output: violations.rtf (&sysdate9 &systime)";
RUN;
ODS RTF CLOSE;
```

Note the use of the _Page variable as the left most and invisible order variable. This, and the BREAK AFTER statement are the only two additional lines required to achieve the accurate pagination of the listing. The first footnote statement illustrates a neat RTF-specific shortcut available in SAS 9 for obtaining "Page x of y" page numbering. Naturally, in a real life example, the final three measurement columns wouldn't be required, but I've included them here to demonstrate the working of the paginator. Here's a screen shot of one page of the output file:

Listing of protocol violations

						Measurement		
Treatment	Subject	Visit(s)/Data type	Details	Comment	Major?	Row height (cm)	Space remaining (cm)	Row lines (lines)
High	002-002	EXPOSURE DAY 0	TIME OF METHOTREXATE ADMINISTRATION IS PRIOR TO BREAKFAST.		No	0.9	8.1	2
	002-006	ALL VISITS	ALL VISITS WERE CONDUCTED OUTSIDE THE AGREED VISIT SCHEDULE DUE TO A CHANGE IN CLINIC DATES.		No	1.3	6.8	3
	002-010	ALL VISITS	ALL VISITS WERE CONDUCTED OUTSIDE THE AGREED SCHEDULE DUE TO A CHANGE IN CLINIC DATES.		No	1.3	5.4	3
	002-013	ALL VISITS	ALL VISITS WERE CONDUCTED OUTSIDE THE AGREED SCHEDULE DUE TO A CHANGE IN CLINIC DATES.		No	1.3	4.1	3
	002-014	VISIT SCHEDULE	DAY 1 WAS CONDUCTED TWICE.	DECODING ENVELOPES WERE NOT AVAILABLE, SO DOSING WAS POSTPONED UNTIL 27/07/04.	No	1.7	2.5	4
	002-017	EXPOSURE DAY 0	TIME OF METHOTREXATE ADMINISTRATION IS PRIOR TO BREAKFAST.		No	0.9	1.5	2

Program: violations.sas
Output: violations.rtf (14JUL2005 10:43)

Note that the titles and footnotes have not been tidied up at this point. Methods for doing this are discussed later on in this paper.

If you're not writing to RTF, or aren't using SAS 9, you can still achieve the required pagination and numbering, albeit with a loss of efficiency, by wrapping the `REPORT` procedure code in a macro do loop and subsetting the input dataset by `_Page` within each execution of the loop.

BUT I DON'T HAVE THE OPTION OF USING THE JAVAOBJ CONSTRUCT

JAVAOBJ is a new and experimental feature in SAS 9. So, if you are using SAS Version 8, or using SAS in a production environment, you can't (or shouldn't) use JAVAOBJ. All is not lost. The paginator has a command line interface and the ability to read xml files defining the columns of a table, their contents and for fonts used to render them. Output is to another XML file that can then be read in to the SAS job and merged with the original data.

DELVING A LITTLE DEEPER

The paginator can handle changes of font and point size both in mid-row and mid-cell, forced line breaks, superscripts and subscripts and has a limited ability to handle raw RTF code that may appear in the source data. The width of the cell padding can also be specified.

Formatted numeric variables are also easy to handle: simply apply the format to the variable when calling the `addCell` method:

```
j.callVoidMethod("addCell", TRIM(PUT(Major, NoYes.)), "1.5cm");
```

If grouping or order variables might affect the space required to display an observation, the `addIDCell` method is a useful variant of `addCell`. `addIDCell` only considers the text when calculating the row height if the observation is either the first on a page or if the user flags its printing as mandatory, typically by using a `FIRST.varname` flag:

```
j.callVoidMethod("addIDCell", PID, "1.5cm", FIRST.PID);
```

Page breaks can be forced by calling the `startPage` method:

```
j.callVoidMethod("startPage");
```

And the unused space remaining on the page can be retrieved by the `getSpaceRemaining` method:

```
j.callVoidMethod("getSpaceRemaining", "cm");
```

The Paginator creates a log file so that its workings can be checked and validated at a later date. The user can specify the level of detail written to the log file. Here's a section of the log file from the job that created the table above.

```
Adding cell 4...
  Cell width: 7.5cm [4252.0twips].
  Cell padding: 3.00pt [60.0twips].
  Available width: 7.29cm [4132.0twips].
  Font: Arial 9pt.
  Display text [1, auto split]: 'ALL VISITS WERE 7 DAYS OUTSIDE THE ' [width=6.49cm,
'SCHEDULE' is 1.80cm long].
  Display text [2, auto split]: 'SCHEDULE AS STUDY DRUG WAS LATE ' [width=6.49cm,
'ARRIVING' is 1.62cm long].
  Display text [3, final]: 'ARRIVING AT SITE.' [width=3.03cm].
  Width: 7.5cm [7.29cm allowing for padding.]
  Height: 37.05pt [3 lines].
  Input text: ALL VISITS WERE 7 DAYS OUTSIDE THE SCHEDULE AS STUDY DRUG WAS LATE
ARRIVING AT SITE.
  Font: 9pt Arial
```

Finally, the cell height problem can be reversed. If the user specifies a text and a maximum number of lines, the Paginator will calculate the minimum column width that will allow the text to be displayed.

```
j.callDoubleMethod("getWidthRequired", Text, 3, "cm", minWidth);
```

HANDLING TITLES AND FOOTNOTES

As I have already explained, the default methods available within SAS for handling titles and footnotes are unsatisfactory for both technical and aesthetic reasons. In some procedures, such as `REPORT`, it is possible to define pseudo-titles using spanning column headers, but the method is not general, and does not work for footnotes. Using `MVAR`s with redefined procedural templates would create a satisfactory solution, but would require considerable effort – the table template associated with each output object from each procedure would need to be modified. In addition, the implementation would introduce an additional level of syntax that would need to be remembered by the user.

OVERVIEW

My preferred approach has been to leave the analysis to SAS and the formatting to a word processor: Microsoft Word being the obvious choice. In this way, the strengths of both packages are utilised, additional coding for the user is minimised, validation is simplified and additional features that are currently beyond the functionality of the ODS, such as column separators on only a subset of columns, are easy to implement.

Therefore, I use standard `TITLE` and `FOOTNOTE` statements to specify the text of the titles and footnotes that I require in the finished document. I know that these will initially be in the wrong place, but I then use a simple program to move them to the correct positions. The `X` command allows this program to be called from within the same SAS job that creates the table in the first place, thus ensuring that the post processing is never overlooked.

Most users are familiar with the concept of using Word macros to automate simple repetitive tasks within Word. These macros are recorded in Visual Basic for Applications (VBA), which is a programming language in its own right. In fact, it's not just simple tasks that can be recorded in VBA: any action that the user can execute using keyboard strokes or menu options has its VBA equivalent. One advantage of using VBA is that validation is simplified: the object model used by VBA to manipulate RTF files (or Word documents) means that it is very easy to verify that any manipulation performed by the VBA does not affect the informational content of the file, simply its presentation. This is not the case if the program manipulates the underlying raw RTF code produced by the ODS.

THE WORD TYPE LIBRARY

One weakness with VBA is that it is an interpreted language. This means that, generally speaking, the source code has to be available for the program to execute. If the source code is available, it can be modified. If the source code can be modified, then validation is made more complicated.

However, the VBA type library allows the functions and objects that manipulate Word documents to be accessed by any languages that support COM. Such languages include Visual Basic, C, C++ and Pascal. These are all compiled languages, which means that source code is not required to execute the program. This removes the vulnerability of VBA programs to the possibility of modifying the source code, thus making validation simple once more.

WHAT THE REFORMATTER DOES

The reformatter takes the contents of the header section of each page and inserts it at the head of the table that follows. It also appends the contents of the footer section of each page at the foot of the table. Optionally, if a page or section break is not preceded by footnotes and not followed by titles, it removes the page break, allowing for the simple construction of master/detail tables that are often used for the presentation of adverse event listings and laboratory profiles. I give an example of this feature in my other technical solutions presentation at this conference (Kirkpatrick (2005)). In addition, the RTF file can be converted to a Word document or PDF file, document fields converted to plain text (so that, for example, page numbering remains correct when tables are included in a master document).

Here's the violations listing after being processed by the reformatter:

Listing of protocol violations

						Measurement		
Treatment	Subject	Visit(s)/Data type	Details	Comment	Major?	Row height (cm)	Space remaining (cm)	Row lines (lines)
.	001-009	ALL VISITS TO DAY 28	ALL VISITS WERE 7 DAYS OUTSIDE THE SCHEDULE AS STUDY DRUG WAS LATE ARRIVING AT SITE.		No	1.3	7.7	3
	001-030	ALL VISITS	ALL VISITS WERE CONDUCTED OUTSIDE THE AGREED SCHEDULE DUE TO A CHANGE OF CLINIC DATES.		No	1.3	6.4	3
	002-008	ALL VISITS	ALL VISITS WERE CONDUCTED OUTSIDE THE AGREED SCHEDULE DUE TO A CHANGE IN CLINIC DATES.		No	1.3	5.1	3
	002-016	EXPOSURE DAY 0	TIME OF METHOTREXATE ADMINISTRATION IS THE SAME TIME AS THE TIME OF BREAKFAST.		No	1.3	3.8	3
Placebo	001-005	SCREENING	SUBJECT IS ALLERGIC TO METHOTREXATE.		No	0.6	3.2	1
	001-010	DAY 32	DAY 32 WAS PERFORMED 3 DAYS AFTER DAY 28, IT WAS IN AGREEMENT WITH PATIENT AND THE CRO.		No	1.3	1.9	3
	001-011	DAY 32	DAY 32 WAS PERFORMED 3 DAYS AFTER DAY 28, IT WAS IN AGREEMENT WITH PATIENT AND THE CRO.		No	1.3	0.6	3

Page 1 of 1

Program: violations.sas
Output: violations.rtf (18JUL2005 11:05)

The additional code required within the SAS job to perform the reformatting is simply:

```
X "F:\Insight\cpp\SRSFormat\SRSFormat.exe in=&outdir\violations.rtf";
```

Column separators can be requested by a simple addition to the command line:

```
X "F:\Insight\cpp\SRSFormat\SRSFormat.exe in=&outdir\violations.rtf sepcols=(2,6)";
```

Resulting in a file looking like this:

Listing of protocol violations

						Measurement		
Treatment	Subject	Visit(s)/Data type	Details	Comment	Major?	Row height (cm)	Space remaining (cm)	Row lines (lines)
.	001-009	ALL VISITS TO DAY 28	ALL VISITS WERE 7 DAYS OUTSIDE THE SCHEDULE AS STUDY DRUG WAS LATE ARRIVING AT SITE.		No	1.3	7.7	3
	001-030	ALL VISITS	ALL VISITS WERE CONDUCTED OUTSIDE THE AGREED SCHEDULE DUE TO A CHANGE OF CLINIC DATES.		No	1.3	6.4	3
	002-008	ALL VISITS	ALL VISITS WERE CONDUCTED OUTSIDE THE AGREED SCHEDULE DUE TO A CHANGE IN CLINIC DATES.		No	1.3	5.1	3
	002-016	EXPOSURE DAY 0	TIME OF METHOTREXATE ADMINISTRATION IS THE SAME TIME AS THE TIME OF BREAKFAST.		No	1.3	3.8	3
Placebo	001-005	SCREENING	SUBJECT IS ALLERGIC TO METHOTREXATE.		No	0.6	3.2	1
	001-010	DAY 32	DAY 32 WAS PERFORMED 3 DAYS AFTER DAY 28, IT WAS IN AGREEMENT WITH PATIENT AND THE CRO.		No	1.3	1.9	3
	001-011	DAY 32	DAY 32 WAS PERFORMED 3 DAYS AFTER DAY 28, IT WAS IN AGREEMENT WITH PATIENT AND THE CRO.		No	1.3	0.6	3

Page 1 of 1

Program: violations.sas
Output: violations.rtf (18JUL2005 11:13)

If necessary, separators can be requested for the table header and table body separately.

AUTOMATING THE PROCESS

It is a relatively simple matter to write a wrapper macro that removes almost all of the details of the process from the user's sight. In the simplest case, the user need specify nothing more than the input dataset, the variables to be printed and widths of the corresponding columns, the name of the output and the ODS style to be used. One slight glitch is that, owing to the current limitations of the ODS, there is no built in method of interrogating a style to determine the font (and point size) that will be used to render a particular piece of output using a particular style element. This still has to be provided by the user and is a potential source of inconsistency and therefore error.

Whilst a wrapper macro can be used to automate the production of standard tables and listings, the direct use of the Paginator Java class allows custom tables to be built accurately and simply. It is possible to handle changes of font and point size mid-cell, and even raw RTF code can be handled or ignored intelligently.

A NOTE ON MEASUREMENT IN RTF FILES

All dimensions In RTF files are measured in whole numbers of twips. There are 1440 twips to the inch, or approximately 566.9 to the centimetre. This means that some dimensions will never be rendered accurately: the maximum resolution in and RTF file is 0.00694in or 0.0176cm. Although this resolution is sufficient for most purposes, it can sometimes have visible effects. In my experience, this is most often in tables with many narrow columns and horizontally merged cells. In this case, the width of the merged cells does not always exactly match the sum of the widths of spanned cells. This is not a fault of the ODS or of the paginator/reformatter system: it is an inherent feature of the RTF format. This means that when exact measurement is critical, dimensions should be specified in "round" fractions of an inch, rather than in centimetres.

In addition, Microsoft Word can itself introduce variation into the file as it appears on screen or in print. I have come across examples where a column width reported by Word differs by 0.1mm from that specified in the raw RTF. Even this small variation can have important consequences: if the Paginator reports that a text is 6.99cm long, and will therefore fit on one line in a cell with an available width of 7.00cm, but Word also thinks that the cell is only 6.99 wide, the text will wrap onto a second line when viewed or printed. An unexpected line of text is much more visible than a column width that is inaccurate by 0.1mm!

CONCLUSION

The control of page layout provided by ODS is adequate for most situations where precise knowledge of page layout and pagination is not required. But for more exacting applications, additional control is required. This paper has demonstrated that, even when SAS does not have the ability to provide the information required, tight and reliable integration of applications that can provide the necessary information is possible. These applications are flexible and accurate. The additional effort on the part of the user can be kept to an absolute minimum in all but the most complex of situations.

REFERENCES

PhUSE 2005 Conference (TU03: "Using tagsets to provide customised output" – John Kirkpatrick, ISC Ltd)

PhUSE 2005 Conference (TS07: "Opportunities for efficiency created by the ODS" – John Kirkpatrick, ISC Ltd)

McNeil S (2002) What's new in the output delivery system, Version 9.0. *Proceedings of the 27th Annual SAS Users Group International Conference*. SAS Institute. Cary, NC.

Wehr P (1996) %print drivers: teaching SAS to speak the many languages of document publication. *Proceedings of the 21st Annual SAS Users Group International Conference*. SAS Institute. Cary, NC.

CONTACT INFORMATION

You can download all the source data, SAS code and output files mentioned in this presentation from www.isc-ltd.co.uk/presentations.

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

John Kirkpatrick
Insight Statistical Consulting Ltd
Tile Barn
High Haden Road
Glatton
HUNTINGDON
Cambridgeshire PE 28 5RU
Great Britain

Work Phone: +44 (0) 1487 830838
Email: phuse2005@isc-ltd.co.uk (Please note: I get a lot of spam email. To maximise my chances of reading your email, please use an informative header and avoid free email accounts such as MSN and hotmail.)
Web: www.isc-ltd.co.uk