Paper TS07

Opportunities for efficiency created by the ODS

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

ABSTRACT

The ODS can greatly simplify the creation of attractive, report-ready output from the SAS System. This talk describes the findings of a feasibility study into the use of ODS tables in a regulatory reporting environment. Colossal savings in coding effort can be achieved. In one case, a program of 14500 lines was reduced to one of seventy. However, these savings may be limited by a rigid adherence to templates defined without reference to ODS functionality. For maximum benefit to the organisation, an entrepreneurial approach from programming managers is required, together with flexibility from clients from both within the organisation and without.

INTRODUCTION

This paper is in two parts. In the first, I describe the results of a study conducted on behalf of one of my clients, Amgen Ltd, in 2002, which aimed to examine the feasibility of introducing the ODS into a mature pharmaceutical regulatory reporting environment. The project compared existing output to that which could be achieved using the ODS, the closeness of the match and the effort required to achieve it. In the second part, I explain solutions to some of the technical solutions we encountered during the project.

PROJECT BACKGROUND

Amgen describes itself as "a leading human therapeutics company in the biotechnology industry". In the last twenty-five years, it has received regulatory approval for at least seven major products and has annual sales in excess of two billion US dollars. Its reporting process is mature and well-established. Tables and listings for inclusion in regulatory dossiers are produced as RTF files using the `%print` drivers described by Wehr (1996). These macros use `data _null_` programming to write raw RTF to ASCII files. Properly used, they provide very high quality output that can be directly included in publication quality documents without further modification. However, even though the analysis programs are heavily modularised by the use of macros, they are long, complex and somewhat fragile. Considerable effort is required to validate even minor changes from project to project, or even study to study within project.

In 1996, Wehr's `%print` drivers were state-of-the-art. Bu the advent of a reliable ODS RTF destination in SAS Version 8.2 clearly changed the situation: even if `%print` remained the most flexible option, it was also the most complex and unwieldy. ODS was at least worth examining to see if it could provide the same control of output format without the complexity inherent with `%print`.

The scope of the project was therefore to attempt to produce tables and listings that matched current company standards. The starting point for each table was the "ready for reporting" dataset: there was no attempt to assess the impact on ODS on data manipulation or analysis.

It is worth noting that the company standards had been developed before the ODS was available. As a result, they were designed without any reference to what the ODS could do easily, or even if it could be done at all. In this respect, the challenge was potentially a tough one.

The ideal solution to each problem, whether ODS-based or not, was one that was:

1. Generic: that is, could be ported from study to study or project to project with little or no modification.
2. Data driven: independent of variable names, variable formats, variable types, number of observations data values and so on.
3. "One PROC away": given a subject-level dataset as input, the procedure that calculates the summary statistics should also create the output table.
4. Platform independent: Amgen work on a dual Windows/Unix network, so solutions should ideally work on both platforms
5. Destination independent: Although currently Amgen required only RTF files, the ability to create PDF files, or files in any other format, at a later time would be useful.
6. Simple: to reduce the costs of staff-retraining and code maintenance.

The success of each solution was to be assessed by:

1. Their proximity to the current layout standards
2. The coding effort required. Both in terms of the lines of code in the driver program and the number of lines generated by any macro calls. In calculating the number of lines of code required, set up code, such as that used to define permanent format catalogs and template stores could be ignored. In addition, the complexity of the code was relevant. A solution which used standard SAS statements was clearly preferable to one which demanded the user to be familiar with detailed parameter settings required for a macro call.

EXAMPLE 1: AN ADVERSE EVENT LISTING WITH A MASTER/DETAIL FORMAT

The screenshot below shows one page of a quite sophisticated listing of adverse event data.

Adverse Events by Subject

Center	Subject	Age (years)	Sex	Race	First dose	Last dose
29	748029301	30	Male	Caucasian	17MAY2000	18OCT2000

Event Start Date	Event Stop Date	Day	Week	Duration (days)	Adverse event	Preferred Term	Severity [a]	Serious	Study Drug Related?	Action [b]
17MAY2000	17MAY2000	1	1	1	Vascular Access Thrombosed	Thrombosis Vascular Access	01	No	No	88
07JUN2000	21JUN2000	22	4	15	Bilateral Leg Pain When Walking	Pain Limb	01	No	No	01
04AUG2000	07AUG2000	80	12	4	Hypertension	Hypertension	01	No	No	07
22SEP2000	13OCT2000	129	19	22	Hypertension	Hypertension	01	No	Yes	07 88
06OCT2000	06OCT2000	143	21	1	Cephalea	Headache	01	No	No	01

Page 1 of 502

Note: . indicates missing

a - Severity: 01=Mild, 02=Moderate, 03=Severe, 04=Life Threatening, 05=Fatal

b - Action: 01=None, 04=Hospitalised, 05=Removed from study, 07=Medication taken, 08=Transfusion performed, 88=Other

Screenshot 1: A page from an adverse event listing

The features of the table that caused the greatest amount of thought were

1. Two tables on one page
2. An arbitrary number of records per subject, and an unknown number of lines required to display each record as a result of wrapping of the adverse event text
3. The requirement to have footnotes adjacent to, but not within, the main table.
4. Subscripts in the column headers

Although the table was complex, the ODS solution was able to match the in-house standard exactly whilst at the same time reducing the amount and complexity of the code required. The in-house solution required a 250 line driver program that generated approximately 14,500 lines of `DATA _NULL_` code. Using SAS version 8.2, the ODS version had 49 lines of code in the driver program generating 8,500 of `PROC REPORT` code. In order to obtain two tables on the same page, it was necessary to wrap two separate calls to `PROC REPORT` within a macro loop. This was the only reason for the large number of generated lines within the ODS solution. If the RTF tagset promised by SAS Institute performs as advertised, I believe a tagset-based solution in SAS 9.2 will require a driver program of around 10 or 15 statements and around 35 statements of generated macro code using a single call to `PROC REPORT`.

I'll discuss how each of the difficulties highlighted above were solved later on in this paper.

EXAMPLE 2: A NON-STANDARD DEMOGRAPHIC SUMMARY

One particular study was causing the group a considerable amount of difficulty as there was a reporting requirement to show summary statistics for one treatment both by dose and overall within the same table. This was a feature not supported by the standard reporting macros, so a study-specific modification was required for each table type. The summary of baseline demographics was typical of the sort of presentation required.

Baseline Demographics by Treatment Group
(Produced by %print)

	Treatment A		Treatment B						
	600 IU/kg (N=10)		6.5 µg/kg (N=10)		8 µg/kg (N=10)		All (N=20)		Total (N=30)	
Sex – n(%)										
Female	8	(80)	7	(70)	9	(90)	16	(80)	24	(80)
Male	2	(20)	3	(30)	1	(10)	4	(20)	6	(20)
Race – n (%)										
White or Caucasian	10	(100)	10	(100)	10	(100)	20	(100)	30	(100)

Page 1 of 1

N=Number of subjects randomised

Program: /statistics/xxx/new_ind/20010174/analysis/final/tables/ma_t5_demog.sas

Output: t5_demog.rtf (Date Generated: 02MAY02:12:09:39) Source data: dddin.c_keyvars

Screenshot 2: The standard demographic summary

It wasn't known exactly how long the modifications of the existing code took, but the general opinion was "several days" for coding and validation.

Starting from scratch, and using ODS, the following table was produced in an afternoon:

Demographics by treatment group
(Produced by ODS)

	Treatment A		Treatment B						
	600 IU/kg (N=10)		6.5µg/kg (N=10)		8µg/kg (N=10)		All (N=20)		Total (N=30)	
Sex - N (%)										
Female	8	(80)	7	(70)	9	(90)	16	(80)	24	(80)
Male	2	(20)	3	(30)	1	(10)	4	(20)	6	(20)
Race - N (%)										
White or Caucasian	10	(100)	10	(100)	10	(100)	20	(100)	30	(100)

Page 1 of 1

N: Number of subjects randomized

Program: /statistics/xxxx/new_ind/20010174/analysis/final/tables/ma_t5_demog.sas

Output: t5_demog.rtf (Date Generated: 02MAY02:12:09:39) Source Data: ddin.c_keyvars

Screenshot 3: The ODS demographic summary

Note: The partially missing separator between the header and the body of the table, and the uneven external border of the table are features of the rendition of the table in this PDF document. They are not features of the original RTF file.

This table is not an exact match to the standard. The major differences are the missing underlining of the spanning headers, the spacing of the column headers, and the gap between the "Treatment A" and "Treatment B" columns in the table. However, the informational content of the table is exactly correct.

The standard solution required a driver program of 100 lines, which generated about 850 lines of `DATA _NULL_` code via a macro. The ODS solution required 22 statements in total, and used only `PROC TABULATE`, `TITLE` and `FOOTNOTE` statements. One has to ask if the extra effort is worthwhile.

It would have been possible to introduce the wider gap between the "Treatment A" and "Treatment B" parts of the table and remove the unwanted blank row in the header, but this would have required moving away from the "one proc away" objective that was set for the project, and so we decided not to proceed down this route.

Since carrying out this work, I have discovered a technique for creating the spanning underlines, although it is somewhat awkward to implement.

Thus, an exact match with the standard could have been achieved, although at the cost of some loss of generality and additional coding effort.

THE BENEFITS OF ODS TO THE ORGANISATION

No one would argue that the informational content of a table or listing must be accurate. But once the informational content of a presentation has been validated, there are an infinite number of ways in which that information can be presented. Clearly some layouts will be better than others, and so should be preferred. However, when the differences between two layouts relate to minor variations in format and layout that make no difference to the overall impact of the presentation, then surely the one that is adopted should be the one that is easiest to implement.

SPECIFICATION VERSUS IMPLEMENTATION: IT'S A TWO WAY STREET

For example, there was a need to emphasise the fact that different doses of Treatment B in the demography table were more closely related to each other than they were to "Treatment A". This was achieved in the standard table by introducing the spanning underline and slightly increasing the space between the columns for 600IU/kg Treatment A and 6.5µg/kg Treatment B compared to the other inter column spaces. But other methods of doing this are possible: a vertical separator running the whole height of the table would do the same thing, and be just as visually acceptable. Using the techniques I describe in my other technical solutions presentation at this conference (Kirkpatrick (2005)), the vertical separator can be implemented by simply modifying an option to a single statement. There is no need to worry about creating table-specific raw RTF code to create the underlines.

The vertical separator option was not an option for this particular exercise, as Amgen had no mechanism for feedback between specification and implementation: the specifiers had unintentionally specified a cost-intensive layout, and the implementers had no choice but to implement it.

Now, this example is a trivial one, but other, more serious options are easy to construct. Whilst modern computer systems will handle pretty much any layout of a given presentation with little noticeable increase in execution time, the effort of developing and maintaining complex code is significantly greater than that of simple code. Coding costs can be considerable: estimates of the cost per line of debugged code range between USD3.10 for correcting Y2K bugs (Newsweek (1998)) to USD125 (Poulin et al (1993)) for writing validated, compiled executables. Whatever figure is chosen, the efficiency introduced by efficient programming is considerable: even at the low end of the scale, for the adverse event listing, the cost is reduced from just under USD45000 to just over USD150. And that is just one table: multiply by the number of tables and listings in a study report and the number of study reports in a dossier and the figures become very impressive.

To benefit from the efficiencies that ODS and other new technologies give us, some form of collaboration between those who specify a standard and those who implement it is required. In the pharmaceutical industry, specifiers are usually medical writers or clinicians, implementers are statisticians, data managers and programmers. Whilst it's true that these groups do often talk to the others, the nature of the relationship is that, generally, the clinicians specify and the programmers find a way to implement. Indeed, this was the case with Amgen: there was no mechanism for the programmers to provide feedback on the relative programming effort required for an exact match to standard and an option that was close, but not perfect. It seems to me that this feedback mechanism, in some form or another, is essential if the benefits of new technology are to be felt by the entire organisation rather than only by those who actually use the new tools.

THE ADVANTAGES RUN DEEP

It's also not an option for programmers and departmental managers simply to moan about the way of the world and complain that the culture of their organisation makes this impossible. I believe a far more entrepreneurial attitude is needed: an attitude in which programmers and their managers advertise the new technologies that are available but unused, and demonstrate the advantages – not just in quality, but most definitely in cost – that they bring to the entire enterprise.

The advantages of ODS aren't restricted to simple efficiency savings and reduced programming costs. ODS can move the production of presentations out of a company specific methodology, such as the customised `%print` environment used by Amgen, into a generic system. True, templates and styles are likely to remain unique, but the methodology is part of core SAS. As such, far more people are likely to have the essential skill sets. This means that staff can be recruited from a wider pool and that less training is needed before a new recruit becomes productive. Moreover, as part of base SAS, ODS is supported by SAS Institute. So if something goes wrong, it is supported. The problem can be passed to SAS technical support at no cost. Thus, training and maintenance are also reduced.

THE BENEFITS OF ODS TO THE INDIVIDUAL

Initially, I called this section "The benefits of ODS to the programmer", but I realised this wasn't correct. I'm a statistician, a statistician who programs it's true, but my main focus is still on statistics. But ODS has still been the biggest change for me and the way that I work in the nearly twenty years I've been working with SAS. Put simply, it lets me focus on information, not presentation. As a freelance, this has colossal benefits: I can easily customise the look and feel of standard outputs for an individual client: I simply define a new style that fits the client's corporate standards and all my existing reporting code produces output in the require format. True, I've had to put a fair amount of thought into exactly how to solve some of the technical issues that ODS throws up, and in this part of the talk I'll explain in some more detail some of the solutions I've come up with. These are the issues that I found easiest to solve, or are the ones that I'm most frequently asked about when I talk about ODS.

PAGINATION

To my mind, pagination is the single biggest area of ODS where SAS Institute needs to put more work. I go into the issues and solutions in far more detail elsewhere (Kirkpatrick (2005)), but put simply, the default behaviour is for ODS to avoid the issue: pagination is delegated to the application which is used to view the output file (usually Microsoft Word for RTF files, Acrobat Reader for PDF files, a web browser for HTML files). True, the default behaviour can be changed, but the fact that ODS knows little or nothing about vertical size, means that the information needed for precise control of pagination simply isn't available to the programmer.

There simple is no acceptable solution based purely on SAS that can solve the problem at the moment. My preferred solution is currently to use a Java class to provide information about vertical size and a COM based executable to provide fine tuning of titles and footnotes. Once it reaches production status, the data step `JAVAOBJ` command will provide tight integration of the Java class with SAS. In the mean time, communication is via XML scratch files. The X command similarly allows the executable to be tightly bound with the original SAS job. In the long term, converting the COM executable from C++ compiled for windows to Java will create a solution that is completely independent of operating system.

One aspect of pagination that SAS Institute have already improved with the arrival of SAS 9 is page numbering. The raw RTF code required to produce "Page x of y" has been posted widely on newsgroups and is now pretty well known. However, three inline formatting functions make the solution a lot neater: `\{thispage}` gives the current page number, `\{lastpage}` gives the total number of pages in the document and `\{pageof}` is equivalent to `\{thispage}` of `\{lastpage}`.

SPECIAL CHARACTERS

When solving a problem, a programmer often has to make the choice between a complex generic and a simple but specific one. This is the case when attempting to create ODS output that contains special characters, such as Greek letters or mathematical symbols. Take, for example, the need to write the Greek letter μ, perhaps as part of a scientific unit, to an ODS destination. Sometimes it's not even clear what the generic solution is. Is a solution that works across all ODS destinations required, or one that simply works for, say, RTF files? Or do we require a solution that will work for every font in a particular destination?

If an across-destinations solution is required, then the only option is to use the `S=` inline formatting command. First, one has to find a SAS font (not a system font) that contains a representation of the required character. In this case, and assuming that SAS/GRAPH is licensed, then the SYMBOL font meets our needs. So to get the value of the character variable `units` to appear as μg in any destination, we need to write

```
units = "^S={font_face=Symbol}m^S={}g";
```

Not forgetting to set the escape character to "^" before printing the data.

Whilst this solution works across all ODS destinations, it assumes that a font that contains the required character is available.

An RTF-specific solution that results in much smaller changes to the original data is to use the Unicode code for the required character. Continuing the example, the Unicode code for μ is 956 (decimal), so

```
units = "\uc2\u956 mcg";
```

will have the desired result provided that the value is rendered with the PROTECTSPECIALCHARS style attribute set to OFF.

This will work for all fonts in the RTF destination (provided that a glyph for the character is defined in the font). Whilst a Unicode code for virtually every conceivable symbol exists (see www.unicode.org/charts), the method relies on the RTF reader (usually Microsoft Word) being able to interpret the code. If it cannot, the alternate text, in this case mc, is displayed instead.

Both methods are easily turned into macros, so that the details can be hidden from the naïve user. A macro that implements the second approach can be freely downloaded from my website.

FINISHING TOUCHES: INDENTATION, ALIGNMENT AND OTHER TWEAKS

My approach to producing a table or listing for a report is that, as far as possible, the underlying source data should not be manipulated to produce any cosmetic effects and that the final table should be produced directly by the reporting procedure, with no post processing. My reasoning is that this is much more likely to make the report independent of the input data and, since the data are "pure", validation is greatly simplified. Even within these constraints, sophisticated effects are possible. Here is a screen shot of one page of a demographic summary produced directly from PROC REPORT. It illustrates several useful techniques. Although some of them are features of PROC REPORT rather than of ODS, they are so closely related to ODS that they're worth repeating.

Summary of demographic data

	Treatment group							
	Placebo		Low		Medium		High	
Gender								
Male	7	(26.9%)	5	(29.4%)	4	(22.2%)	8	(21.6%)
Female	19	(73.0%)	12	(70.5%)	14	(77.7%)	29	(78.3%)
Ethnic origin								
Caucasian	26	(100.0%)	17	(100.0%)	18	(100.0%)	37	(100.0%)
Smoking status								
Smoker	6	(23.0%)	5	(29.4%)	7	(38.8%)	14	(37.8%)
Ex-smoker	4	(15.3%)	2	(11.7%)	3	(16.6%)	4	(10.8%)
Never smoked	16	(61.5%)	10	(58.8%)	8	(44.4%)	19	(51.3%)
Duration of disease (years)								
N	26		17		18		37	
Mean	10.42		9.14		10.53		10.88	
SD	7.576		7.936		7.593		7.861	
Min	0.1		0.8		1.7		0.5	
Max	29.1		27.8		29.6		29.6	
Median	9.51		6.97		8.73		8.53	

Page 1 of 4

Duration of disease is calculated as the number of years between first diagnosis and screening.
CRP values of less than 20µg/mL (the LLQ) have been assigned the value 0 when calculating summary statistics.
Program: demography.sas
Output: t_demog.rtf (09AUG2005 15:15)

Screenshot 4: another demographic summary table

The input data and source code for this table can be downloaded from my web site.

Before beginning, it's worth mentioning that I'm not using the REPORT procedure to calculate summary statistics from the subject level data. That would be too difficult. Instead, the input data consists of the pre-calculated summary statistics. I'm using it simply to create an appealing table.

The two most obvious features of the table are the spanning section headers in the body of the table and the indentation of the category labels. The spanning headers are pure PROC REPORT: an ORDER variable is defined as NOPRINT, and a COMPUTE BEFORE block prints the current value at the start of each section. The bold formatting is applied by an on-the-fly style definition:

```
DEFINE Class/ORDER NOPRINT ORDER=DATA;
COMPUTE BEFORE Class/STYLE=[JUST=LEFT FONT_WEIGHT=BOLD PROTECTSPECIALCHARS=OFF];
  LINE Class $CLSFMT.;
ENDCOMP;
```

The indentation of the row labels is trivial in SAS 9. The INDENT style attribute does the trick:

```
DEFINE Row/DISPLAY " " STYLE(COLUMN)=[INDENT=0.5cm CELLWIDTH=5cm];
```

In SAS Version 8.2, the INDENT style attribute isn't available, so a different approach is required. It's necessary to pass some raw RTF to define the indent. Fortunately, the PRETEXT style attribute allows the necessary code to be emitted without manipulating the underlying data values:

```
DEFINE Row/DISPLAY " " STYLE(COLUMN)=[PROTECTSPECIALCHARS=OFF
                                      PRETEXT="\tx283\tab" CELLWIDTH=5cm];
```

The RTF specification can be downloaded from the Microsoft website if you need more information on the syntax.

A small, but important, formatting tweak is to remove the missing value dots from the percentage columns when summary statistics (as opposed to frequency counts) are being presented. Actually, the dots aren't hidden, they're just printed in white. Here's how. First define a format that maps missing to white and all other values to black:

```
PROC FORMAT;
  VALUE ForeFmt .="WHITE" OTHER="BLACK";
RUN;
```

Now set the value of the FOREGROUND style attribute to the name of the format on the columns in which you want to hide the missing values. For example:

```
DEFINE Pct1/DISPLAY " STYLE(COLUMN)=[FOREGROUND=ForeFmt.];
```

The <attribute>=<format name> is an extremely powerful technique that can be used to supply values for any style attribute, both in DEFINE statements and, even more usefully, in COMPUTE blocks.

The page numbering is achieved using the technique I've already described:

```
ODS ESCAPECHAR="^";
FOOTNOTE1 J=R "Page ^{pageof}";
```

The µ in the third footnote is created using the Unicode technique I mentioned previously, but there is a slight snag in that there is no way to change the style element that renders system footers on-the-fly. The only option is to redefine the entire style. Fortunately, this is simple:

```
PROC TEMPLATE;
  DEFINE STYLE TempRTF/STORE=WORK.Templates;
    PARENT=styles.LandscapeA4;
    STYLE SystemFooter FROM SystemFooter/
      PROTECTSPECIALCHARS=OFF;
  END;
RUN;
ODS PATH (PREPEND) WORK.Templates;
```

Now simply use the new style when opening the RTF destination:

```
ODS RTF FILE="t_demog.rtf" STYLE=TempRTF;
```

Achieving the varying precision in the summary statistics is a three stage process. First, create a dummy variable containing a concatenation of the variable name and the summary statistic (for example "WeightMean"). This variable is the left-most DISPLAY variable in the report definition, but is defined as NOPRINT:

```
COLUMN Page _FmtVar CLASS ROW
       ("Treatment group" ("Placebo" N1 PCT1) ("Low" N2 PCT2)
                          ("Medium" N3 PCT3) ("High" N4 PCT4));
DEFINE _FmtVar/DISPLAY NOPRINT;
```

Next, define a format that maps the values of this variable to the format required, for example:

```
VALUE $STATFMT "N"="F3."
               "WeightMean"="F6.2"   "WeightMin"="F5.1" "WeightMax"="F5.1"
               "WeightMedian"="F6.2" "WeightSTD"="F7.3"
```

and so on. Finally, use this format in a CALL DEFINE statement to define the format of each column that needs to be handled in this way. For example:

```
DEFINE N1/DISPLAY " " STYLE=[CELLWIDTH=1.75cm JUST=DEC];
COMPUTE N1;
  CALL DEFINE(_COL_, "FORMAT", PUT(_FmtVar, $STATFMT.));
ENDCOMP;
```

The decimal alignment is achieved by the JUST=DEC style attribute in SAS 9. As with the INDENT styyle attribute, equivalent raw RTF can be used to achieve the same effect in Version 8.2.

The table header and footers are defined by standard TITLE and FOOTNOTE statements.

```
ODS ESCAPECHAR="^";
TITLE "Summary of demographic data";
FOOTNOTE1 J=R "Page ^{pageof}";
FOOTNOTE2 J=L "Duration of disease is calculated as the number of years between
              first diagnosis and screening.";
FOOTNOTE3 J=L "CRP values of less than 20\uc2\u956 mcg/mL (the LLQ) have been
              assigned the value 0 when calculating summary statistics.";
FOOTNOTE4 J=L ITALIC H=8pt "Program: demography.sas";
FOOTNOTE5 J=L ITALIC H=8pt "Output: t_demog.rtf (&sysdate9 &systime)";
```

They are moved to their correct positions by the reformatting program that I describe ind etail elsewhere:

```
X "F:\Insight\cpp\srsformat\srsformat.exe
   in=F:\Insight\presentations\PhUSE2005\TS007\t_demog.rtf";
```

The separator lines are added at the same time. Separate specifications for the header and body of the table are needed because the table header contains spanning headers.

```
X "F:\Insight\cpp\srsformat\srsformat.exe
   in=F:\Insight\presentations\PhUSE2005\TS007\t_demog.rtf
   hsepcols=(1,2,3,4)
   bsepcols=(1,3,5,7)";
```

CONCLUSION

In this paper, I have demonstrated that sophisticated and elegant report formats can be created using standard SAS programming statements and with very little programming overhead. The ODS is the basic environment for these reports. There is little if any need for complex, bespoke layouts to be created by DATA _NULL_ programming or other techniques. However, if the benefits of the ODS to overall organisation are to be maximised, then programmers and programming managers need to develop an entrepreneurial streak to convince their colleagues of the advantages of ODS and their colleagues need to learn to develop standard templates with regard to the capabilities of the ODS.

REFERENCES

Newsweek (1998) July 20th Edition, page 14.
PhUSE 2005 Conference (TS03: "Pagination in the ODS" – John Kirkpatrick, ISC Ltd)

Poulin JS, Caruso JM and Hancock DR (1993) The business case for software reuse. *IBM Systems Journal* 32(4):590-594
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.

ACKNOWLEDGMENTS
I am grateful to Amgen Ltd for their permission to reproduce some of the example tables and listings in this paper.

CONTACT INFORMATION
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