ODS to RTF: Tips and Tricks

Paul Hamilton, Immunex Corporation, Seattle, WA

Abstract

Since the introduction of the ODS RTF destination, traditional character-based displays of tables and listings are passé. The new capabilities also bring new tribulations as the pharmaceutical community strives to produce high quality statistical reports, maximizing the amount of information on a page, while still conforming to FDA guidance. This paper examines some obscure ODS to RTF formatting tricks available by getting creative with SAS version 8.2. Topics covered include selective bolding and graying of data values, insertion of blank separator rows, selective graying of report rows, management of widow and orphan data lines, control of table cell widths and heights, font control, and selecting and displaying special characters (footnote notations, scientific and mathematical symbols, typography characters).

Introduction

Since the introduction of the Output Delivery System in version 7 of the SAS System, report writing has become much more complex. With the introduction of the Rich Text Format destination (RTF), the appearance of tables and listings has changed dramatically. However, the myriad possibilities are not yet clearly documented. This paper attempts to coalesce and clarify some of the techniques available, and how they are implemented. Comparisons to the traditional ASCII listing are noted, along with much more advanced techniques not previously available.

A brief caveat is in order at this point. The techniques described below were found to be appropriate and useful in our company, with our statisticians, and with our medical writers. Several of the techniques follow accepted practices in the print publishing community (something that has seldom been an option previously when producing reports from the SAS system), while others are very arbitrary and personal choices.

All these techniques were developed and tested under Version 8.2 of the SAS system. One bug and one limitation have been discovered so far, but otherwise the ODS RTF destination has proven to be flexible, powerful, and stable.

DIFFERENCES FROM TEXT LISTINGS

Large parts of a SAS program that produces a report rendered in RTF format are identical to a traditional program. All of the data cleanup, manipulation, summary and analyses remain the same, although there is the potential for some additional data manipulation just to aid readability. The primary differences occur at the very end of a program, where the summarized data are submitted to a report writing step.

Our goals and assumptions in researching the new RTF capabilities were:

- exclusive use of Proc Report for final report writing step
- mirror structure of existing reports as much as possible
- comply with FDA guidance as understood within our company
- produce RTF output for easy copy, editing, and integration with other documents
- utilize standard publishing technologies where practical
- expend a reasonable amount of effort to make the tables and listings easy to read and visually pleasing.

Although tremendous gains in report writing functionality, flexibility, and quality are obtained by using the ODS RTF destination, there of course has to be at least one drawback. This primarily affects listings with either one of two characteristics:

1. The listing contains multiple, variable number of records per report unit (e.g. patient). Some common examples of this type of listing are: Adverse Events, Infections, and Drug Administration.
2. The listing contains a long text field that is forced to wrap within its column over multiple lines of text. Any listing containing verbatim comments may exhibit this behavior, especially if the display is already crowded with other information (again, Adverse Event listings are a prime candidate).

One advantage of an RTF document is that it is easy to achieve the “Page X of Y” that has long been desired in traditional character listings. In RTF, this is accomplished by placing the following code in a suitable location (in this example, a Title statement):

```
title1 j=l "Some Text"
                 j=r "{Page} {\field{\*\fldinst
{ PAGE }}}\~{of}\~{\field{\*\fldinst {
NUMPAGES }}}";
```

When opened in a suitable RTF reader, this produces the desired result (forcing paging by “Print Preview” may be necessary.) However, there is an unfortunate side effect of this feature: Proc Report is no longer controlling, or even aware of, how full the page is becoming during report generation. Page management is turned over to the program that renders the RTF into a document. This means that the traditional effect of the Order definition does not behave as before.

For example, if an Adverse Event listing uses Patient Number as an Order variable, Proc Report will print only the first occurrence of each patient, and leave the report lines just below blank in that column. In the past, Proc Report would know when a new page started, and could print the Patient Number on the top of the new page, even though it had not changed value from the last record. This feature is a causality of progress, and will force the report reader to flip back to the previous page to determine which patient is being examined. Although there is no automatic way around this new limitation, a possible alternative will be presented below. However, because of this limitation, the author has adopted several new rules for RTF reports:

1. use only *Define* / *display* statements in Proc Report (no *order*), which implies…
2. data ordering must be accomplished prior to Proc Report, using Proc Sort.
3. once the data are ready to display in a report, one additional Data step might be required to optimize the final outcome.

Consider specifying at least the column size and justification for each variable in the report. This is accomplished with new *style* commands that appear on the *Define* statement:

```
Define … / display ‘Label’
  style(column) = [cellwidth = XX in
           just = {left | center | right}];
```

Specifying just these extra controls makes an enormous difference in the quality of the resulting report. Text automatically wraps, so use of the *flow* keyword is now redundant.

RESOURCES FOR CUSTOMIZATION

There are several ways to customize RTF reports. Proc Template allows the user to specify many elements of a custom document. Proc Report provides a wide array of formatting capabilities within its own domain. Finally, the traditional SAS tools are quite useful, especially Proc Sort and the Data step. We have chosen to start the process by avoiding Proc Template as much as possible, and exploring the capabilities within individual reports. The only usage of Proc Template is to create a style that provides a consistent look to all reports produced:

```
proc template;
  define style pharma;
    parent = styles.rtf;
    style body from body /
        leftmargin = 1.0in
       rightmargin = 1.25in
         topmargin = 1.3in
      bottommargin = 0.8in;
    style heading from heading /
      cellpadding = 0
      cellspacing = 0;
    style table from table /
            rules = groups
            frame = void
      cellpadding = 1.5pt
      cellspacing = .25pt;
    replace fonts /
  'TitleFont2' = ("Times New Roman",9pt,Bold)
  'TitleFont'  = ("Times New Roman",9pt,Bold)
  /* <snip – more fonts defined>*/ ;
    replace headersAndFooters from cell /
             font = fonts('HeadingFont')
       foreground = colors('headerfg')
       background = white;
  end;
run;
```

This template store may be shared among all programmers in a group, and invoked when creating an RTF file:

```
ODS RTF file = “dir\file-spec” style=pharma;
```

All other formatting is applied using the Data step, Proc Sort, and style commands directly in Proc Report. As our “standard reports” become accepted using these new technologies, Proc Template will undoubtedly play a larger role.

THE BASICS

Changing your Proc Report from character listing to basic ODS RTF does not take much work. In order to produce an RTF report, first inform ODS where the report should be placed:

```
ODS listing close;
ODS RTF file = ‘directory/file-information’;
proc report …;
  column …;
  define …;
run;
ODS RTF close;
ODS listing;
```

Document styles may be applied when specifying the output file location. Proc report will make fairly intelligent guesses as to how your data should be formatted, and one could stop there. However, a minimum of effort on each *Define* statement will provide much better control over the large-scale formatting issues.

TRICKS WITH DATA

Nearly every table and listing has at least one footnote. Footnotes require a placeholder to tie the text (often in a column heading) to point to the appropriate footnote. Traditionally, these footnotes markers have been simple characters unlikely to appear in the text, e.g. “+” and “*”. With RTF, more complex and interesting markers are available, such as the dagger (†), double dagger (‡), and other assorted glyphs (§ ® ©). Other symbols are useful for specialty purposes, such as laboratory displays (e.g. ± ° ² ³ µ). However, other symbols may be required on occasions that are not available in the primary font (e.g. ∞ α θ ≠ √). Interspersing symbols from different fonts requires a more advanced technique, covered below.

Execute the following steps to enter symbols directly into a SAS editor (even an FSEDIT screen!):

1. ensure the NUM LOCK key is on
2. determine the 4-digit ASCII code for the desired character (in the desired report font)
3. hold down the Alt key,
4. while typing the 4 numbers on the numeric key pad (not the sequential number keys at the top of the keyboard).

The character which appears in the SAS editor might look like the desired symbol, or completely different. However, when displayed in the report font, the desired symbols will appear properly when viewed on-screen or printed.

Another technique is to use simply assignment statements in the Data step to produce special characters. Perhaps your listing contains a “check if applies” variable (Adverse Events, hospitalization, infection, etc.) During the days of ASCII listings, the column heading might say “Check”, but the data column traditionally shows an “X” or some similar character. However, one can easily change this to a true check mark by doing 2 things:

```
if (checkvar eq 'X') then checkvar = 'Ö';
```

and, in the Proc Report *Define* statement,

```
define checkvar /
  display …
  style(column) = [font_face=“Symbol”];
```

The check mark in the Symbol font is “0214”, and can be entered in SAS as described above.

This technique works to shift any particular body column into a particular font. A related technique, of switching between multiple fonts within one column, is illustrated below.

BLANK SEPERATOR ROWS IN YOUR REPORT

The RTF specification does permit style commands to increase or decrease the height of any particular row in a report. After some experimentation with this technique, it was deemed to be unsuitable. Depending on the characteristics of the data being displayed (especially long text strings inside listings, such as comments), it was all too easy for the data to overrun the room allotted for it, resulting in partially hidden text. Consequently, we adopted the approach of adding blank lines to the report-ready SAS data set using the Data step. The blank lines are very easy to control, easy to detect, and cause minimal problems on output. One generic algorithm for this technique is:

```
data …;
  set … end = finish;
  by by_var1 by_var2…;
  /* keep an extra copy of patient id   */
  by_var1_copy = by_var1;
  /* Call a macro which defines all     */
  /* numeric and character variables in */
  /* standard array references.         */
  %AllArray;
  if (not first. by_var1) then do;
    /* See Changing Row Characteristics */
    by_var1 + .1;
    by_var2 + .1;
  end;
  output;
  if last.by_var1 then do;
    if (by_var1_copy in (list)) then return;
    if finish then return;
    /* Call a macro to set vars to missing */
    %ClearAll;
    output;
    if (by_var1_copy in (list2)) then output;
  end;
run;
```

This technique is generally more useful on listings than on reports. The code above assumes that at least one blank line is desired between each section of patient records. No assumptions about uniformity of records per patient are made. The first *output* statement above creates the actual lines visible in the report. After a section break (typically the last of a patient for listings), all variables are set to missing via a macro and then the new blank line is output.

The code that copies id variables and returns before the secondary output, or performs yet another blank line output, is designed to accomplish several goals:

1. avoid creating an extra blank line at the end of the report - on rare occasions, this can cause an additional blank page to print out.
2. avoid writing a blank line after a section when that blank line starts at the top of a new page.
3. try to avoid having only one line of a patient occur at the very bottom of a page.

Points 2 and 3 above require extra work, and may be judged as too laborious or time consuming in many cases. They are only appropriate when the report layout is fixed, and all data being displayed has been cleaned and locked. At the very final stage of the report generation, one can page through a listing after it has been rendered from RTF, looking for awkward placement of groups on pages. Then (in a very manual and tedious process) one can go back to the Data step and insert certain values of *by* variables (typically patient number) and either discard the extra output line, or add yet an additional line. This is a manual attempt to control for "widows and orphans", and will not be considered worthwhile in all circumstances. If the listings are not too long (less than 500 pages), this can add a small bit of visual quality, although at greater consumption of programmer time.

IN-STREAM RTF COMMANDS

RTF commands may be inserted into text streams, allowing a great deal of flexibility and power. RTF commands may be inserted into column headings in Proc Report, into character variables with the Data step, and into Proc Report Define statements. In order utilize RTF commands, two resources are necessary:

1. an understanding of how to distinguish RTF commands from normal text, and
2. a resource for discovering RTF commands and their effect.

Several effective methods exist for identifying RTF codes, but the next example is the author's preference due to simplicity of coding. The technique relies on the ability to designate a special character that ODS recognizes as an "escape" character—denoting that next characters are RTF commands and not normal text. Any character may be identified as the escape character, but practically speaking, one should select a character that has very little chance of occurring in the SAS data. The backslash (\) is a poor choice, since it is used by the RTF specification. Some SAS documentation illustrates this point using the caret (^), which in practice has show to be a good choice.

Assuming the caret has been chosen as the escape character, the following sequence of characters is used to issue RTF commands:

```
'^R"rtf-keywords-and-parameters"'
```

That is, a single quote, followed by the escape character, followed by "R", followed by a pair of double quotes containing RTF commands, and completed by a single quote. Examples below will illustrate how this technique can work in the Data step, Proc step, and macro language code.

A more complete example, using the Data step, follows:

```
data new_text;
  set whatever;
  length NewString $ 500;
  NewString = '^R"\b"'        ||
              trim(string1) ||
              '^R"\b0"'     ||
              trim(string2);
ODS escapechar='^';
```

The code above will combine two existing text strings together into one new string—but the first string will appear in bold font weight, while the second extant string will appear in normal text. The string "\b" turns "bolding" on, while the same command suffixed with a zero turns it off again (similar effects are available for italics and other typical character modifications.) The ODS command defining the escape character may appear anywhere before the Proc Report invocation which creates the desired report. The “R” following the escape character informs the ODS processor that raw RTF code follows, and should be passed along intact to the RTF reader.

TEXT “FOLDING”

The technique above may be extended more generally by a “folding” macro, show below. This is useful in a variety of reports, especially listings with structured long text strings such as:

- Adverse Events (with verbatim event text, and either COSTART or MedDRA coding to several levels)
- Infections
- Medications, etc.

Normal Comment listings may not take good advantage of such a technique, because there is usually not an inherent structure as in the previous examples.

In the macro below, the first three parameters are the names of the character variables to manipulate: Original String # 1, Original String 2, and Result string. In this example, either string may have bold applied to the text style, and the second string may be indented by any desired amount (measured in twips). The macro *fold* must be called in a Data step, and the original strings must already exist. Also, the coder is responsible for ensuring that the result string is long enough to contain both the original strings, plus the RTF formatting characters inserted by the macro. Fortunately, this is very simple in Version 7 and 8 of the SAS system, with long character variables finally a reality.

```
%macro fold(os1=, os2=, res=,
            bold1=no, bold2=no, indent=0);
  %local istring;
  %let istring = %str(%')^R
                 %str(%")
                 \par\li&indent
                 %str(%")
                 %str(%')||;
  &res =
  %if &bold1 = yes %then %do;
            '{\b '              ||
  %end;
            trim(&os1)          ||
  %if &bold1 = yes %then %do;
            '}' ||
  %end;
            %unquote(&istring)
  %if &bold2 = yes %then %do;
            '{\b '              ||
  %end;
            left(trim(&os2))
  %if &bold2 = yes %then %do;
            || '}'
  %end;
  ;
  &res = compbl(&res);
%mend fold;
```

Spacing in the macro above is simply to facilitate readability in the short column layout, and is not vital to the performance of the code. The final line, which “squeezes” multiple blanks to one blank each, may or may not be considered desirable in other circumstances.

The table stub below illustrates the practical effect of this technique. In this example, the literal drug name as supplied on the CRF displays first, followed by an indented coded (“preferred”) drug name, displayed in bold. Other choices are easily obtained, such as varying the order, degree of indentation, and character formatting (e.g. bold, italics) applied to each part.

Drug Name   **Preferred Name**
BECONASE NASAL ADMINISTERED TWICE PER DAY   **BECLOMETASONE DIPROPIONATE**
ALBUTEROL   **SALBUTAMOL SULFATE**
VANCERIL   **BECLOMETASONE DIPROPIONATE**
VANCENASE NASAL   **BECLOMETASONE DIPROPIONATE**

Now the benefit of not simply inserting spaces becomes apparent. When a text string becomes too long to fit within the column dimension, it wraps to its own margin. This makes the combination of long text strings much easier to read and interpret. Note that the arrangement of column header exactly matches the location and character styling of the body data it refers to. This is accomplished by specifying in-stream RTF commands in the Column and Define statements, with parameters matching the previous call to the Fold macro in a Data step.

```
data …;
  %fold(os1=drugname, os2=prdrgnam,
        res=drug_prefdrug,
        bold2=yes, indent=150);
…
run;

proc report … ;
 column (… other variable names
       ('^R"\b0\ql "Drug Name' drug_prefdrug)
 define  drug_prefdrug / display
       '^R"\b\li150"Preferred Name' …;
```

The “\b” turns on the character bold attribute, while “\li150” specifies an indentation (see “Decimal Tabs” below for clues on the numbering schema).

Using these techniques, variables that were previously displayed in separate columns may not be “folded” intelligently into one column, with different styles applied to aid in readability. This can free up quite a bit of space, and make more intelligent use of the limited number of words available in a report.

ALIGNMENT IN TEXT AND HEADERS

There are three places one can focus on text alignment within the body of a report (not including the headers and footers as controlled by Title and Footnote statements):

- In the body
- In a column header
- In a spanning column header

The first two controls are quite uncomplicated, while the third requires a bit of intricacy. Controlling simple justification (*left*, *center*, or *right*) for body columns and one-column headers is accomplished with the appropriate Style command on the Define statement:

```
… / style(column) = [just=keyword]…;
… / style(header) = [just=keyword] …;
```

Using this technique, justifications of column body and column headers are independent of one another. This is handy when dealing with wide text column (in either tables or listings), and the column header might look more balanced aligned with the left cell margin, or centered over the column (the default behavior).

This technique may or may not work with spanning column headings. For example, consider the partial table header below:

Event Onset	
Date	Day

Using the *style(header)* technique described above, the single column header of "Date" may be left or center justified, as deemed appropriate. However, this has no effect on the spanning header "Event Onset", and it will be center justified by default. While this might be suitable in this case, it may look very strange in other cases.

Completion		Reason for Not
Date	Day	Completing Study

In the table stub above, the spanning column heading "Reason for Not" is only spanning one column. The text is being placed as a separate spanner to reduce the overall size of the column, while placing the upper header line adjacent to other text, to avoid "floating" above other column text. The *style(header)* technique will control the justification of the lower header, but not the upper one. This is another opportunity to use in-stream RTF commands, such as:

```
proc report …;
column (…
        ('Completion' date_var day_var)
        ('^R"\ql "Reason for Not' why_var)
```

In the *column* statement, the same method of entering RTF codes works flawlessly. In this case, the RTF code "ql" is used, forcing left justification when centered justification would normally apply. This forces the two column headers to match one another as well as the body column beneath.

DECIMAL TABS: TEXT BALANCING

One extremely useful technique is the use of decimal tabs. At present, two separate reasons for employing decimal tabs have come to mind:

1. aligning numeric values of widely different units of scale within the same column (e.g. low and high laboratory normal values)
2. more aesthetic alignment of body text with column headings (e.g. a small body text column such as age (XX) under a wider column heading such as "Age of Respondent at Registration").

Using this technique requires some convoluted syntax, and also the use of a rather esoteric measurement system. However, the syntax occurs as usual within the context of the Define statement in Proc Report, and the measurement system is easy to use once comprehend. Consider the code stub below:

```
… style(column) = […
   protectspecialchars = off
   pretext = "\tqdec\txNNN "]
```

The text "protectspecialchars=off" is required syntax for this technique. The "pretext" and following specifications are also required, and diverge from the in-stream RTF commands considered so far. There is no ODS escape character and no "R", and only one pair of double quotes. The familiar back slashes are present, as well as some strange commands. The pretext specifies a decimal tab ("\tqdec") followed by direction on where to place the tab ("\tx*NNN*", where the *NNN* is a number supplied by the SAS coder).

The metric for this number is expressed in *twips*, undeniably an unusual notion. Many measurements in word processors are expressed in terms of inches, i.e. requesting a column in a table to be 1.2 inches wide. Another measurement system in wide use is the point, defined as 1/72 of an inch. A twip is defined as 1/20 of a point, or 1/1440 of an inch. When requesting decimal tabs, the *NNN* is measured in these twips.

Duration (Days)
43
4
24

In the table stub above, notice that the values for day are neither left nor right justified. One might then guess that they are center justified, but then the numbers would not align with each other so nicely. This was accomplished by setting left justification, and specifying a decimal tab:

```
Column … ('Duration' aedur) … ;
define aedur /
  display '(Days)'
  style(column) = [… just=left
        protectspecialchars=off
        pretext="\tqdec\tx650" …];
```

In practice, one need not necessarily convert column inches to twips numerically. Often, one can take a blind guess at an appropriate value, observe the outcome in the RTF report, and quickly converge on a value that looks very aesthetic. Since twips represent small fractions of an inch, guesses may be made in intervals in the hundreds (e.g. a test sequence of 100, 300, 500, 800, 750, 720).

Decimal tabs are a little more complicated to understand, but provide a subtle sense of quality to a report by helping to balance text blocks.

DECIMAL TABS: ALIGNING NUMBERS

In the table below, laboratory normal ranges (low and high) are displayed (much extraneous text has been removed). Note that numbers with very different orders of magnitude and different levels of reporting accuracy are nicely aligned with one another.

RBC	4.1	5.3
SGPT	0	45
Sodium	135	146
Specific Gravity	1.001	1.035
Total Bilirubin	0	1.3
Total Protein	5.5	8

The table shows yet another usage for decimal tabs when aligning dissimilar but related types of numbers. A frequently used macro produces a text column of univariate summary measures. Actually, these often look satisfactory when displayed using center justification. However, as the scale of the numbers or their variation increases, the display sometimes winds up looking quite clumsy. Employing decimal tabs forces the decimal of the first number encountered In each cell to line up to the specified tab insertion point.

Number of Doses	Mean (SE)	41.4 (3.5)
	Std Dev	30.0
	Median	34.0
	Min – Max	-2.0 – 99.0
	N	75

By the way, there is a very subtle RTF trick in the display above. When reporting a range (e.g. Min and Max), it is possible for both the numbers reported to be negative. This leads to an awkward construction such as:

```
-2.34 - -1.97
```

The trouble here is that, using traditional character reporting techniques, there is no way to distinguish between a normal dash (meaning "negative number following") and a different type of dash (meaning "from-to".) However, there are provisions for such characters, and it requires just a simple ODS RTF trick. Notice the minimum number of doses reported above is negative 2 (not reported in a study, just for illustrating a point!) If you observe carefully, the negative sign in front of the "2" is shorter in length than the longer dash between the two numbers. It is easy to encode an "en-dash (0150)" or an "em-dash (0151)" instead of a "regular" dash. Usage of these dashes is governed by rules that may be new to many programmers. An excellent reference covering these topics is provided at the end of the paper.

CHANGING ROW STYLES

Some listings or reports have a complex inner structure, and additional tricks may help make that structure more explicit. In the listing below, the same person participated in several studies, and was assigned different patient numbers. Since the data refer to the same person, forcing a page eject or inserting a blank line might be confusing. With ODS RTF, it is simple to apply a style to a particular row, representing a change in study status.

Subject	Visit	Date	Day	Body System
43	Baseline	30Nov1994	-13	Asthma/Allergy
43	Baseline	30Nov1994	-13	Asthma/Allergy
43	Baseline	30Nov1994	-13	Musculoskeletal
43	Baseline	30Nov1994	-13	Gastrointestinal
56	Baseline	16Jan1995	-17	Pleuropulmonary
56	Baseline	16Jan1995	-17	Genitourinary
56	Baseline	16Jan1995	-17	HEENT
56	Baseline	16Jan1995	-17	HEENT

This was accomplished using two techniques:

1. Sort the data by "universal patient identifier", followed by the unique patient identifier in each protocol. In the Data step, define a dummy variable (fp for "First new Patient) which has the value of 1 for each row to be grayed, and 0 otherwise.

2. In the Proc Report, the following code will selectively apply a 25% gray background to a row on the report:

```
define fp / display ' ' noprint;
other define statements as needed …
compute fp;
  if (fp eq 1) then call define(_row_,
    "Style", "STYLE=[BACKGROUND = cxDDDDDD]")
  endcomp;
```

The technique is to apply another style, gray background, to the entire row (previous styles have been applied to individual columns). The notation "cxDDDDDD" specifies a 25% gray level.

This trick might also be useful in tables with sub-sections, where a visual differentiation between sections can make the table more readable. The extract from a Laboratory Toxicity table below shows how each new laboratory parameter has a gray background applied.

Laboratory Toxicity	Phase	Highest Grade	Counts	Percent
Albumin Low	PRE	Total	0/75	0.0
		1	0/75	0.0
		2	0/75	0.0
		3	0/75	0.0
		4	---N/A---	----
	DUR	Total	1/74	1.4
		1	0/74	0.0
		2	1/74	1.4
		3	0/74	0.0
		4	---N/A---	----

With a little extra work, one can use these same techniques to provide a work-around to the problem with the *Order* usage mentioned above. Since the paging control occur outside of SAS, after the RTF processor has completed writing a report, using the *Order* parameter on the Define statement becomes much less useful than in the past. Switching to a *Display* usage undoubtedly works, but makes a listing monotonous to read, and makes distinguishing sets of related records (e.g. patient) more difficult.

One approach that has proved successful is to print the repeating columns (which would normally be used with the Order usage) in bold type on their first occurrence, followed by gray when repeating the same values. This makes the listing much easier to scan, and eliminates the reader having to "flip back" a page when a block of records crosses over a page boundary. This can be accomplished by a combination of tricks in the Data step and in Proc Report.

Subject	Site	Visit	Date	Start Time	Drug (min)
100	**1**	**Day 0**	05Sep2000	09:05 AM	10
100	1	Day 0	05Sep2000	12:45 PM	12
100	1	**Week 1**	12Sep2000	05:36 AM	11
100	1	**Week 2**	19Sep2000	05:40 AM	8

This problem has to be solved in slightly different ways for numeric vs. character variables. Part of the following solution relies on the assumption that the numeric variables are by definition used for identification purposes, and so are nominal variables, and therefore integers.

Furthermore, it is assumed that these variables have an appropriate format assigned to them in the SAS data set (e.g. 5. for Patient, 3. for Site). This means that Data step code can manipulate these variables is useful but non-detectable ways. The trick for these identifying variables is to add a small constant to all records after the first one:

```
data …;
  set …;
  by patno site;
  if (first.patno eq 0) then do;
    patno + .1;
    site + .1;
  end;
  format patno 5. Site 3.;
```

Since the patient number and site variables have integer formats applied to them, they look identical on all lines of the report. However, Proc Report can distinguish the difference, and apply different text styles depending on the preference:

```
compute patno;
  if (patno ne .) then do;
    if patno ne int(patno) then
     call define(_col_, "STYLE",
         "STYLE=[FOREGROUND = cxDDDDDD]");
    if patno eq int(patno) then
     call define(_col_, "STYLE",
         "STYLE=[FONT_WEIGHT=BOLD]");
  end;
endcomp;
```

In practice, most listings have blank lines inserted between patient groups by the Data step. The first *if* condition stops the processing of these blank lines, thus avoiding many obnoxious and useless warning messages from appearing in the SAS log. The logic then continues to check whether the numeric variable is actually an integer (despite its appearance on the listing), and if not, a foreground style of gray is applied. This has the effect of graying out the text, as opposed to graying out the row itself we observed in the previous technique. Proc Report will require a section of code like that above for each numeric identifying variable processed.

Character variables are dealt with in a similar fashion. The trick here is again to make a manipulation to the values that will not appear on the report, but will enable Proc Report to distinguish between the first and subsequent occurrences of repeating values.

```
data …;
  set …;
  by patno visit;
  if (not first.visit) then
    visit = ‘ ‘ || trim(visit);
…
proc report ..;
compute visit;
  if substr(visit, 1, 1) eq ' '
   then call define(_ col_, "STYLE",
            "STYLE=[FOREGROUND = cxDDDDDD]");
  if substr(visit, 1, 1) ne ' '
   then call define(_col_, "STYLE",
                 "STYLE=[FONT_WEIGHT=BOLD]");
endcomp;
```

The logic is the same as with numeric variables, and the same styles are applied in each case (first row bold, subsequent rows gray). The data manipulation in this case is to pre-append a blank character to the text string. Oddly enough, this does not display when examining the final report. This is because of the happy coincidence that RTF processing ignores leading blanks unless directed to respect them with the style *ASIS = YES*. Thus, the visits all begin on the left margin of their column cell, and the bold-gray trick works flawlessly.

If changing data values is too forbidding a notion, the same technique may be achieved by defining dummy binary variables that are tested in order to apply the desired style. However, using just one variable instead of two made the development of a generic RTF style defining macro more straightforward.

MULTIPLE HEADER FONTS

Previously, we examined how to change an entire report column to a different font, to support printing o special characters (e.g. the check mark). We also considered how to insert odd characters into a text stream, using the Alt-numeric keypad trick. However, this does not work if it is necessary to include a character that simply does not exist in the primary font. For example, perhaps all of your column headings are displayed in Times New Roman. Inserting a Greek beta symbol (code 0223) is straightforward. If a column heading requires a more rarely used character, such as an alpha character (α), or a right arrow (⇒), then things get more complicated. These characters simply do not appear anywhere in the Times New Roman font. They are readily available in the Symbol font, so the technique requires mixing different fonts in the same header. Fortunately, this is feasible using in-stream RTF commands. The example below illustrates how to insert a right-facing arrow in a column header, producing “Before ⇒ After”.

```
define … / display
  '^R"\f1"Before ^R"\f2"Þ ^R"\f1" After'
  style(column) = [cellwidth = 0.45 in
                   just = center
                 font_face="Times New Roman"]
  style(header) = [ just = center
                    font_face = "Symbol"];
```

The reason this technique works is that each RTF file contains a definition of all the fonts used in the document. The fonts are referred to by their font number, which is specified in the font definition section. Knowing that there are only two fonts defined in a report, one can freely switch back and forth between them using the

```
^R\fn
```

code sequence, where *n* refers to the desired font number. In the *define* statement above, the quoted text string following the *display* keyword may be interpreted as:

- escape character R – raw RTF commands following
- in a quoted string, the first command-switch to the first document font
- the literal text *Before*, which will appear in the heading
- escape character R – raw RTF commands following
- switch to the second document font
- print character number 0222 (does not appear as an arrow above because it is not displayed in Symbol font)
- escape character R – raw RTF commands following
- switch back to font definition # 1

- print the word *After* – appears in normal font.

Using this type of technique, multiple fonts may be freely intermixed anywhere text can occur in a report:

- titles
- footnotes
- single column headings
- spanning column headings
- body (row) text
- formats

Given the enormous variety in fonts available in the computer and publishing industry, virtually any desired display should be feasible with a modicum of exertion.

SWITCHING BETWEEN STYLES WITHIN A COLUMN

Some reports mix different types of information displays within the same column. In the table below, one part of column 2 contains univariate statistic names, while a subsequent section contains a frequency distribution of a numeric variable. In column 3, the summarized output of Proc Univariate appears, followed by counts and percentages. If one were to apply the same justification to the entire column, some parts of the report would look ideal, while others would be at least inelegant, and perhaps repulsive. Fortunately, the intrepid RTF programmer can accomplish almost anything.

The data set was organized so that a dummy numeric variable contained the panel numbers (1, 2, 3…) for each report section. This sorting variable is used as a trigger for applying different styles in (visible) columns 2 and 3.

# of Doses	Mean (SE)	41.4 (3.5)
	Std Dev	30.0
	Median	34.0
	Min – Max	2.0 – 99.0
	N	75
#f Doses per Patient	2	2/75 (3%)
	4	2/75 (3%)
	99	1/75 (1%)

In the Proc Report code fragment below, actual values of the sorting variable are examined in order to identify what type of column formatting is desired. Remember that these are not data values from CRF-driven SAS data sets, but codes determined by the report layout. The codes only change if the table structure is modified, and are not subject to the data quality issues inherent in clinical trials data.

```
compute name_of_col_2_;
  if (sortvar in (2, 4)) then
   call define(_col_, "style",
               "style = [just = left
               protectspecialchars = off
               pretext = '\tqdec\tx1000 ']");
  if (sortvar in (5, 7, 10)) then
   call define(_col_, "style",
               "style = [just = left ]");
endcomp;
compute name_of_col_3_;
  if (sortvar eq 1) then
   call define(_col_, "style",
               "style = [just = left
               protectspecialchars = off
               pretext = '\tqdec\tx450 ']");
endcomp;
```

The techniques used above are the same ones examined in previous examples (setting justification, use of decimal tabs). This example simply highlights the ability to apply different types of formatting to different sections of a report.

PROBLEMS AND LIMITATIONS

Hopefully you will agree that, with a little creativity, ODS RTF can accomplish an astonishing amount of high quality formatting. ODS RTF under Version 8.2 of the SAS system (under Windows/NT) has proven to be powerful and reliable. However, the world of software being what it is, there has to be at least one fly in the ointment, and some limitations. ODS RTF did have significant problems when writing out a longer Drug Administration listing, which ran to around 1000 pages in length. The symptoms of the problem were:

- the computer started slowing down
- screen refreshes became glacial
- eventually the whole Windows session seemed to grind quietly to a halt.

SAS Institute Technical Support confirmed this as a know problem with Version 8.2. A hot fix is available that addresses this problem, but the accompanying documentation indicates that it is not yet fixed in version 8, but will be under Version 9. Re-running the listing confirmed this caveat: the machine died slower and in a different way, but the report was never created successfully. Alternate strategies were to use Proc Print instead (which did function properly, but produced such an unattractive report that it was discarded), and breaking the listing into three separate parts (the method chosen). Hopefully this problem will be addressed in Version 9.

Another problem has cropped while rendering the RTF code. In Word 97, occasionally the last line of a report lands exactly at the end of a page. Word 97 then builds an extra new page, containing the titles, footnotes, column headings, but no row data. This is unwieldy to cope with, but has nothing to do with SAS and it's RTF code (the same problem occurs when building tables directly in Word). So far, then only solutions to this problem have been:

- add in extra blank lines between sections in the Data step, forcing the last group of rows to move to a new last page
- remove a blank line between groups of rows, letting two patients run together.
- manipulate the line spacing on several rows to be more shallow, taking care not to hide text when doing so.

We hope to discover a way to handle this more elegantly, or that later versions of Word do not exhibit this behavior.

The only constraint so far in our adventures with RTF reports is the inability to place one cell border under a spanning column heading. This would serve to aid the eye in discerning which lower columns the spanning column header is referencing.

Previous Study Participation	
Protocol	Subject

The table fragment above might be better displayed like the one just below:

Previous Study Participation	
Protocol	Subject

The RTF specification certainly allows for this functionality, and the effect can be accomplished manually in a word processor in a very short amount of time. However, it is not yet apparent how to accomplish this effect directly in Proc Report code.

CONCLUSION

The advent of the RTF destination in the Output Delivery System has vastly increased the power and flexibility of SAS based reporting. An entire new world of capabilities is opening up. Hopefully SAS programmers in industry can learn to adapt and deploy these techniques.

REFERENCES

Art Carpenter, *Carpenter's Complete Guide to the SAS® Macro Language*, Cary, NC. SAS Institute, Inc., 1998, 242pp. Useful for tips on how to encode multiple quote marks while inserting RTF commands into a macro.

Haworth, Lauren E., *Output Delivery System: The Basics*, Cary, NC: SAS Institute Inc., 2001. Any attempt to understand ODS styles should include this reference early and often.

Robin Williams, *The PC Is Not a Typewriter : A Style Manual for Creating Professional-Level Type on Your Personal Computer*, Portland, Or. Book News, Inc. Invaluable and easily digestible lessons how to look great in print by following the rules you never even heard of. The *Mac* version of the book came first.

Several useful FAQ are available at:
http://www.sas.com/service/techsup/faq/ods.html

One of above FAQ points to a URL for the Rich Text Format specification documentation:
http://www.cena.dgac.fr/~sagnier/info/formats/rtf/rtfspe15.htm

ACKNOWLEDGMENTS

Several colleagues shared in the exploration of these techniques. I would like to acknowledge and thank John Enzwiler and Mathew Harmon for their hard work in researching and developing these techniques, and their generosity in sharing the subsequent discoveries. I also greatly appreciate the support and feedback of my manager, Jennifer Lester.

CONTACT INFORMATION

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

Paul Hamilton
Immunex
51 University
Seattle WA 98101
Work Phone: (206)587-0430 x3936
Fax: (206)262-9978
Email: hamiltonp@immunex.com
Web: http://www.immunex.com/