Paper 066-31

Save Those Eyes: A Quality-Control Utility for Checking RTF Output Immediately And Accurately

Michiel Hagendoorn, Amgen, Thousand Oaks, CA
Jonathan Squire, C2RA, Cambridge, MA
Johnny Tai, Comsys, Portage, MI

ABSTRACT

A common deliverable for SAS® programmers in biotech and other companies are summary tables and listings in a structured format, like Rich Text Format (RTF) generated by SAS® ODS. During quality control activities, these files are visually compared to independently generated output based on the same specifications. When a large number of output files or reruns are involved, this becomes time consuming and prone to human error.

We developed a utility that reads the data from an indicated RTF file into a SAS® data set. Quality control occurs by programmatically comparing this data set to the independently generated summary via the COMPARE procedure for instant and 100% accurate results, regardless of output size or quantity. We will present an overview of this utility.

The utility uses only the SAS/BASE® version 8.2 product and is geared to users with at least an intermediate level understanding of SAS/BASE® and SAS/MACRO® language.

INTRODUCTION

In the biotech and pharmaceutical industry, SAS® programmers usually produce 2 main types of deliverables:

- case report tabulations (CRTs: pre-specified permanent SAS® data sets containing raw data from the clinical trials database, and some derived data)
- tables and listings (TLs: summaries and displays of data in CRTs) and graphs. This paper discusses TLs produced by SAS® ODS in rich text format (RTF), although the underlying idea might be generalized to any structured markup language.

International and federal regulations require that we assure the quality of these deliverables. In this paper, activities to ensure and document that a CRT or TL is programmed to specification and accurately summarizes or displays its source data, are referenced as Quality Control (QC).

One commonly accepted method of QC is to have a second programmer write code based on the same specifications, independently of the original code, and compare the results. If they match, the output is said to have been QCed.

- For CRTs, reviewers can use PROC COMPARE to find differences between the original and the QC data set.
- For TLs, reviewers resort to a visual check of the original against the QC file (Figure 1).

The table below lists several advantages of PROC COMPARE versus a visual comparison. If the common visual check component in the QC of TLs could be replaced by PROC COMPARE, these efficiencies would be gained.

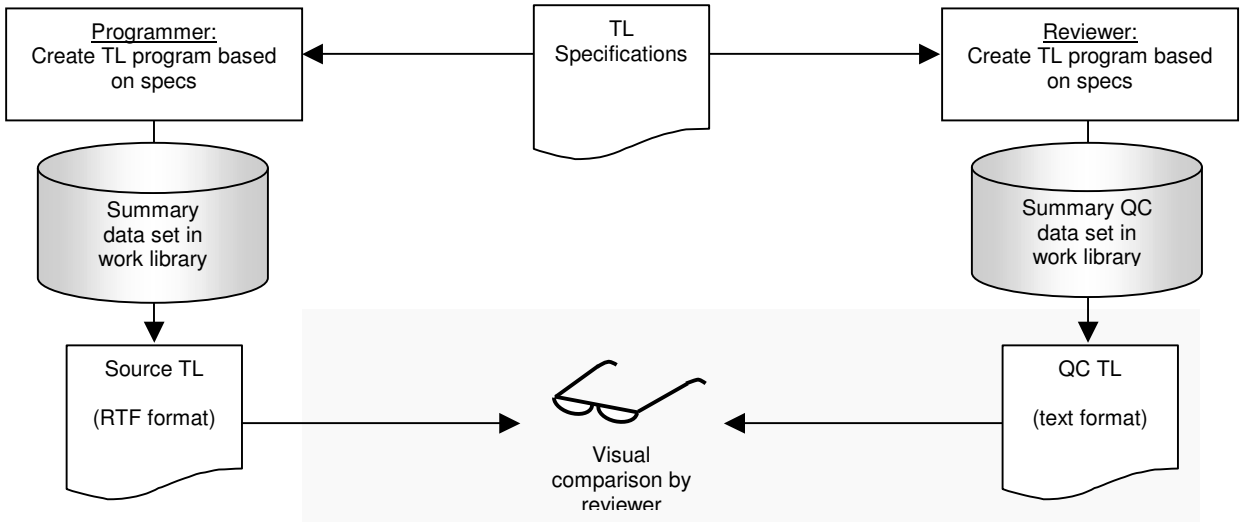
Disadvantage of Visual Comparison	Advantage of PROC COMPARE
Slower than a computerized check.	Faster than a visual check.
Reviewer might overlook an actual difference between source and QC instance.	All differences are picked up, big and small.
Does not provide a record that the source and QC instances were a match.	The .lst file provides documented, time-stamped evidence that no differences were found.
If an output file is repeated for different analysis sets, QCers usually would check 1 instance and if it passes, assume the repeats pass as well.	Checks all repeats with the same level of accuracy and attention.
If an output file passes QC at a very early stage, subsequent visual re-QC is needed when the source data changes down the line to ensure the program still processes the new data correctly. This takes as much time as the original QC.	If an output file passes QC at a very early stage, the PROC COMPARE can simply be rerun at the push of a button if the data changes down the line.

We developed %readrtf, a SAS® macro that realizes the efficiencies, accuracy, and completeness of checking seen in QC of permanent data sets by converting RTF-formatted TLs back to a SAS® data set, which is then compared to the independently programmed QC data (figure 2). For instance, if TLs are run against a new transfer of clinical data, hundreds of thousands of data points represented in these TLs can be QCed in a matter of minutes. In some instances, listings spanning thousands of pages can be matched directly with the originating source data files.

TL QC USING TRADITIONAL METHODS

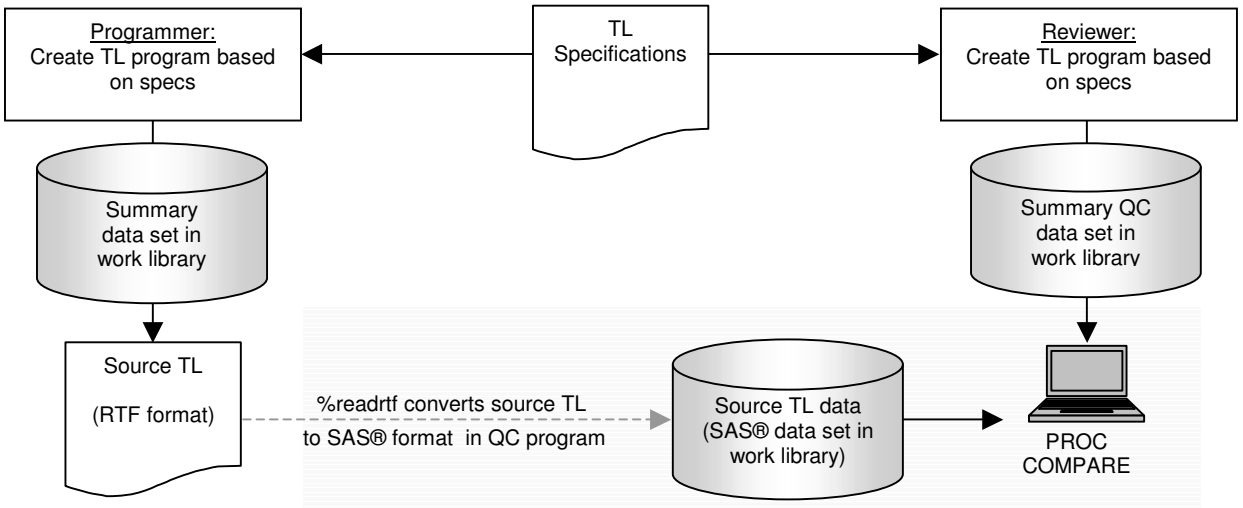
TLs are usually created on the basis of specifications which we shall reference as "table shells". A TL program processes the source data as needed and often creates a summary data set in the work library that is ready for printing to the output destination. In our case, the output destination is an RTF file. This is illustrated on the left side in the diagram below.

The QCer also bases his or her QC program on the table shells. (S)he could prepare one or more summary data sets in the work library which are then ready for printing to the output destination, which is usually the SAS® .lst file. This is illustrated on the right side in the diagram below. In the traditional TL QC process, the reviewer would then visually compare the source output to the QC output and confirm there are no differences.



TL QC USING %READRTF

The diagram below illustrates the process flow of TL QC using %readrtf. The source programming side is identical to the traditional process. In the QC program, however, 2 small steps are added: a conversion of the RTF source TL to a temporary SAS® data set, and a PROC COMPARE of that data set against the temporary QC data set. The computer compares all data in the 2 files and produces the standard PROC COMPARE report indicating whether or not any differences are found, and if so, where they are.



OVERVIEW OF THE PROGRAM

SAMPLE TABLE

A commonly presented output file in a clinical study report is an adverse event (AE) table. An example AE table in RTF format, which will be referenced throughout the rest of this paper, is provided below. Appendix 1 presents the same table as shown when opened in Notepad (modifications were made to remove redundant control words).

A Title for Our Sample RTF Table

	All Subjects (N = 19)		
HIGH-LEVEL TERM		Worst Grade [a]	
Preferred Term	Any	3	4
Any AEs - n(%)	19 (100)	6 (32)	9 (47)
HIGH-LEVEL TERM	17 (89)	3 (16)	2 (11)
Event	17 (89)	3 (16)	2 (11)
Severity 3	16 (84)	2 (11)	1 (5)
Severity 4	5 (26)	0 (0)	1 (5)

Page 1 of 1

[a] This is a footnote.

Program: /path/path/path/t_ae.sas, output: t_ae.rtf

SAMPLE MACRO CALL AND OUTPUT

The macro call below demonstrates how %readrtf would be invoked to convert the sample table to a SAS® data set; the PRINT procedure was used to display the SAS® data set as it appears when %readrtf has finished. Keyword parameter `f=`, the only required parameter for the macro, identifies the RTF file to be processed. Keyword parameter `indent_c1=`, which is optional, tells the macro to mimic the indentation in the RTF file by left-padding with blanks in the output data set. This option is not normally used and is only invoked here for demonstration purposes.

```
%readrtf(f=/path/path/path/t_ae.rtf, indent_c1=y);

proc print data=readrtf;
run;
```

```
seg-       sub- row-
ment level item num   c1              c2        c3      c4      num2  num3  num4  pct2  pct3 pct4
---- ----- ---- ----  --------------  --------  ------  ------  ----  ----  ----  ----  ---- ----
  1     1     1    1   Any AEs - n(%)  19 (100)  6 (32)  9 (47)   19     6     9   100    32   47
  2     1     1    2   HIGH-LEVEL TERM 17 (89)   3 (16)  2 (11)   17     3     2    89    16   11
  2     2     1    3     Event         17 (89)   3 (16)  2 (11)   17     3     2    89    16   11
  2     3     1    4       Severity 3  16 (84)   2 (11)  1 (5)    16     2     1    84    11    5
  2     3     2    5       Severity 4   5 (26)   0 (0)   1 (5)     5     0     1    26     0    5
```

REQUIREMENTS

A tool like %readrtf can only be developed if all TLs share a standardized structure. In this case, that means the presence of fixed elements that allow for programmatic identification of logical components in the file. For instance, in the sample table, we want to be able to identify the title, the table header, the body of the table, and the footnotes. For multi-page tables, we want to be able to do this for each page. Enforcing a standardized structure throughout all TLs is usually done at the department level within a company.

In our setting, most TLs follow a structure similar to the example table, and the enforced TL output format is RTF. Like HTML or any other structured text format, the RTF specification includes control words that are translated by other software, such as Microsoft Word®, into formatted documents. The control words are visible when the RTF file is viewed in a text editor like Notepad on Windows, or vi in UNIX. Because RTF files are really text files, they are accessible via INPUT statements in a DATA step, which made development of %readrtf feasible. Some of the commonly used RTF control words that are key to the operation of %readrtf are shown below.

Control word	Meaning
\trowd	start of table row and implicit start of first cell on row

Control word	Meaning
\li240	indentation of 240 (in twips: 1 twip is 1/20th of a point)

Control word	Meaning
\row	end of table row
\cell	end of table cell

Control word	Meaning
\sect	new section
\clbrdrb	cell border bottom
\brdrr	paragraph border right

By searching for these and other control words in relation to the standardized TL structure, the information in any TL can be processed and converted to a SAS® data set that represents the data points exactly as shown in the TL.

PROGRAM FLOW AND DETAILS

The program flow of the %readrtf macro can roughly be outlined as follows.

1. Read in an indicated RTF file as a plain text file of varying line length. Remove RTF control words for symbols like greater-than-or-equal signs, superscripts, etc.
2. Remove all control words for formatting of table borders, cells, special characters, etc. This leaves only the "true" data in the TL in each table row.
3. Determine where the data section (table body) of the TL begins. Delete observations representing information from title, column header, and footnote rows.
4. Determine level of indentation for each table body row to prepare for assignment of logical segments below.
5. Separate counts and %s into individual numeric variables, if applicable.
6. Based on the unique levels of indentation in the RTF file, assign ordering variables to each observation in the SAS® data set to track positioning within the TL.
7. Output SAS® data set `work.readrtf`.

The sections below provide a step-by-step review of the macro code. Presenting the entire code is beyond the scope of this paper. The code segments discussed here were selected to provide a sense of how the macro parses standardized RTF files into a representative SAS® data set that is ready for a PROC COMPARE.

The discussion assumes a layout like the sample table shown above, generated directly from SAS® 8.2 using the ODS RTF destination with the BODYTITLE option. If TLs are generated or post-processed by different software or follow a different standard layout, the code would need adjustment to work in that specific setting.

As a final note, in our company some TLs have a consistently different layout than the sample table used in this paper. To accommodate processing of such planned deviations, a `devtype=` keyword parameter is defined in the macro which lets users identify deviant layout styles as a numeric identifier. The macro will then conditionally execute additional steps, not shown here, to correctly process that particular layout.

1. READING THE RTF FILE INTO SAS

The first step is to read the entire RTF file, including RTF control words, into a SAS® data set for further processing. A row number (observation counter) is assigned to preserve the table structure after sort steps further down:

```
data readrtf_a;
   infile "&f" missover length = l end = lastobs lrecl = 2000;
   input string $varying2000. l;
   rownum = _n_;
```

Next, we convert frequently occurring RTF-formatted special characters to ASCI, or remove them altogether as they would not usually appear in a QC data set. In the examples below, the ≥ sign is reset to >= and superscript a (^{a}) is removed.

```
   string=tranwrd(string, '{\field{\*\fldinst SYMBOL 179 \\f "Symbol" }}', '>=');
   string=tranwrd(string, '{\super a} ' , '');
```

2. IDENTIFYING COLUMN TEXT

Now we get to the core of the macro, where RTF control words are stripped from actual text and numbers in the body of the table to reproduce a crude semblance of the original SAS® data set that was printed to RTF. Variables `c1-c99` are placeholders for table column information and initialized to blank. Since each table can have a different number of columns, we play it safe by reserving room for up to 99 columns. Variable `dropme` is a flag used to indicate we are in the RTF file. If set to 1, the record will be deleted at the end. This value is manipulated further below.

```
   retain c1-c99 dropme indent;
```

```
length c1-c2 $200 c3-c99 $50;
if _n_ = 1 then dropme = 1;
```

The RTF control word *\trowd* signals the beginning of a row in an RTF table. A table row will ultimately correspond to an observation in our output data set, hence this is a good time to reset all column variables to blank so they can be populated with only the values from this table row.

```
array c{99} $;
if index(string, '\trowd') then do;
   count    = 0;
   indent   = 0;
   do i=1 to dim(c);
      c{i} = '';
   end;
end;
```

Values in each cell on a table row are captured between angled brackets ({ and }). The RTF control word *\cell* signals the end of text printed in the cell. Thus, by capturing the characters between { and *\cell*, we obtain the text as it appears in that column, on that row, in the table:

```
if index(string, '{') and index(string, '\cell') then do;
   count + 1;
   prep     = substr(string, 1, index(string, '\cell')-1);
   prep     = scan(prep, 2, '{');
   c{count} = compress(prep, byte(13));
```

We're still processing the value of a single cell within the `if`-condition. Variable `count` is the column number. The leftmost column of a table, where `count=1`, usually provides a row label that identifies the info presented on that row. In our sample table, these are "Any AEs - n(%)", "HIGH-LEVEL TERM 1", and "Event 1-3". Note that event 1-3 are subcategories of the high-level term, which is visually identified by the indentation of these terms. In the RTF specification, indents are implemented as *\li240*, where *240* can be any number to indicate the desired magnitude of indentation. In the code below, we first capture the level of indentation of the first column by searching for the *\li* control word, and then remove the *\li* control word from the cell text.

```
   if count = 1 then do;
      sst = substr(string, index(string, '\li') + 3);
      if verify(sst, '-0123456789') > 1 then
         indent=input(substr(sst, 1, verify(sst, '-0123456789') - 1), best.);
   end;
   if index(c{count}, '\li') then do;
      c{count} = substr(c{count}, index(c{count}, '\li'));
      c{count} = substr(c{count}, index(c{count}, ' ')+1);
   end;
end;
```

3. IDENTIFYING DATA PRINTED IN THE TABLE BODY

The code above ensures each cell value is correctly read and formatted. This goal has now been achieved. In this same DATA step, we also determine whether this observation (eg, table row) is part of the title, table header, body, or footnote. We're only interested in data printed in the table body and want to discard the rest. This is accomplished in the code below:

- At the top of each page, or at "page x of y", set `dropme` to 1 to signal we're in the title or footnote area.
- At the start of the first row of the table, set `dropme` to 2 to signal the start of the table header. Per our sample table, this is where the top row begins – starting with the *\trowd* RTF control word.
- At the first occurrence of the leftmost cell being underlined per RTF control word *\clbrdrb*, set `dropme` to 3 to signal the last row in the table header. Our standard style guidelines ensure this first underline in column 1 always separates header from body.
- At the end of the last row in the table header (RTF control word *\row*), set `dropme` to 4 to signal we're passing the end of the table header and moving into the body of the table.
- At the first row underneath the table header, set `dropme` to 0 to signal that no observations are to be deleted until `dropme` changes value again when the cycle repeats at the bottom of the table on this page.

```
if dropme = 4 then dropme = 0;
if index(string, '\sect') or
```

```
      (index(compress(lowcase(string)), 'page') and
       index(         lowcase(string) , ' of '))     then dropme = 1;
   else if index(string, '\trowd'  ) and dropme = 1 then dropme = 2;
   else if index(string, '\clbrdrb') and dropme = 2 then dropme = 3;
   else if index(string, '\row'    ) and dropme = 3 then dropme = 4;
```

Once we hit a *\row* RTF control word, which signals the end of an entire row, we know all information on that row has been read. The code above should have populated all column variables (`c1-c[n]`) with the data from each cell on the row, and per the RETAIN statement, their values were kept along the way if cell values for this row appeared across multiple observations in the readrtf_a data set. We can now output this entire row as a single observation to the output data set.

The code below assumes a row was blank (`allblank=1`) unless at least 1 column had a value. If at least 1 column had a value, then `allblank` is overruled to 0 and the record is sent to the output data set. We are not interested in blank rows because those are merely inserted as visual aids to help organize a TL, hence we will not output those:

```
   if not dropme and index(string, '\row') then do;
      allblank = 1;
      do i=1 to dim(c);
         if compress(c{i}, ' \') ne '' then allblank = 0;
      end;
      if not allblank then output;
   end;
run;
```

Depending on the number of columns in the RTF table, many of the column variables (`c1-c99`) have not been populated. We want our macro to auto-detect such unused column variables and remove them from the `readrtf_a` data set before delivering it for further refinement. This code creates macro variable `&dropper` containing the names of all unused column variables, which is then used in a DROP= input data set option further below; it also counts the number of remaining column variables (RTF table columns) after entirely blank columns are dropped and stores that in macro variable `&numvars`:

```
proc transpose data=readrtf_a(drop=count) out=chk;
   var c:;
   by rownum;
run;

proc sql noprint;
   select distinct _name_ into: dropper separated by ' '
   from chk where _name_ not in (select _name_ from chk where col1 ne '');

   select distinct count(distinct _name_) into: numvars
   from chk where col1 ne '';
quit;
```

4. DETERMINE LEVEL OF TABLE ROW INDENTATION

We now have a data set with 1 observation per table row where all title, header, and footnote info has been removed. Special characters have been removed. To make the output data set more prepared for a PROC COMPARE against a QC data set, we want to make a few more tweaks to it.

First, we identify the relative level of indentation of column 1 for each row in the table (that is, observation in the output data set). We had stored each level of indentation in variable `indent` in step 2. The unique values in this variable can be ranked in magnitude and mapped to an ordinal scale. Level 1 would be no indent, level 2 would be the smallest indent, level 3 is the next-smallest indent, and so on:

```
proc sort data=readrtf_a(drop=count &dropper) out=readrtf_b;
   by indent;
run;

data readrtf_c(index=(rownum));
   set readrtf_b;
   by indent;
   if first.indent then level + 1;
run;
```

5. SEPARATE COUNTS AND %S

As a second enhancement in the output file, we split n (%) values into a numeric count (n) variable and a numeric % variable. That way, the QCer does not need to format the QC ns and %s into a precisely formatted character variables "n (%)", instead a direct compare can be done against rounded QC numbers.

Macro variable `numvars` was created in the SQL procedure step in section 3; it is used here as the array subscript for the number and % variables that are subsequently populated for each table column as appropriate:

```
data readrtf_d(drop=i num1 pct1 allblank);
   set readrtf_c;

   array c  (&numvars) c:;
   array num(&numvars)   ;
   array pct(&numvars)   ;
   do i=2 to dim(c);
      if      c(i) not in ('' '-') and
         verify(compress(c(i)), '-0123456789.')=0 then num(i) = input(c(i), best.);
      else if c(i) not in ('' '-') and
              verify(compress(c(i)), '-0123456789.()')=0 then do;
        num(i) = input(scan(         c(i)      , 1, '('), best.);
        pct(i) = input(scan(compress(c(i), ')'), 2, '('), best.);
      end;
   end;
run;
```

6. ASSIGN TABLE SEGMENT, ITEM, AND ROW IDENTIFIERS

Based on the relative levels of indentation marked in step 4 above, we can now identify table segments, subitems within each segment, and cumulative row number. Usually a QC data set will cover the entire table, but if needed, this setup allows a QCer to compare against specific sections in a table or troubleshoot QC results.

```
data readrtf(keep = segment--rownum c: num: pct:);
   length segment level subitem rownum 8.;
   retain segment 0 subitem;
   set readrtf_d;

   if level = 1 then segment + 1;
   if level ne lag(level) then subitem = 0;
   subitem + 1;
   rownum = _n_;
```

If the user requested an image be present in the output data set of the indentation seen in the RTF table, by setting parameter `indent_c1=Y` in the macro call, this code will take care of it. It uses the relative levels of indentation identified above and replaces them with the appropriate number of leading blanks:

```
   if compress(lowcase("&indent_c1")) = 'y' then do;
      if      level = 2 then c1 = '  '                  ||c1;
      else if level > 2 then c1 = repeat('  ', level-2)||c1;
   end;
run;
```

The output data set, `readrtf`, is now ready to be used in a PROC COMPARE against independent QC results.

As a reminder, due to space limitations, the code above is not the complete %readrtf program. The overview was intended to provide a sense of challenges we encountered and how they were addressed in our specific setting.

CONCLUSION

The output for a typical clinical trial can easily exceed 200 tables and listings. The %readrtf macro has proven a useful tool to increase the efficiency and accuracy of QCing each of these outputs in our specific setting. By generalizing the logic outlined in this paper, the %readrtf approach could be extended to other structured settings such as HTML or different TL standards to make QC more accurate and efficient.

The current macro is considered part of RTF TL QC, and not a one-in-all QC substitute. Reviewers should still visually check the TL titles, footnotes, special characters such as superscripts, and the log file for errors and warnings. In rare cases, a customized RTF file is to be QCed that %readrtf cannot process. In such cases, the reviewer should perform visual comparison of QC vs. source output and not use %readrtf.

On a final note, the %readrtf approach described in this paper requires the QCer to create their QC results in the same format as the work.readrtf output data set produced by %readrtf. For small tables, it might be more efficient to visually compare numbers straight from a MEANS or FREQ procedure.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. The SAS® code can be sent upon request. Contact the author at:

Michiel Hagendoorn
MS 24-2-C
Amgen, Inc.
One Amgen Center Drive
Thousand Oaks, CA 91320
Work Phone: (805) 447-4420
Email: mhagendo@amgen.com

APPENDIX 1

The text below provides the RTF definition for our sample table.

```
{\rtf1\ansi\ansicpg1252\uc1\deff0\deflang1033\deflangfe1033
{\fonttbl {\f1\fswiss\fprq2\fcharset0 Arial;} }
{\colortbl; \red0\green0\blue0; }
\widowctrl\ftnbj\aenddoc\formshade\viewkind1\viewscale100\pgbrdrhead\pgbrdrfoot\fet0
\paperw12240\paperh15839\margl2160\margr1440\margt1440\margb1080
\sectd\linex0\endnhere\headery1800\footery1440\marglsxn2160\margrsxn1440\margtsxn1440\
margbsxn1080
\pard\plain \qc {\b\f1\fs22\cf1 A Title for Our Sample RTF Table \par} \pard{\par}

\trowd\trkeep\trhdr\trqc\trgaph20
\clbrdrt\brdrs\brdrw15\clbrdrl\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx4399
\clbrdrt\brdrs\brdrw15\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx8466
\pard\plain\intbl\keepn\sb20\sa20\ql\f1\fs20{\cell}
\pard\plain\intbl\keepn\sb20\sa20\qc\f1\fs20{\brdrb\brdrs All Subjects{\line}
(N\~=\~19)\cell}
{\row}

\trowd\trkeep\trhdr\trqc\trgaph20
\clbrdrl\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx5185
\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx8466
\pard\plain\intbl\keepn\sb20\sa20\ql\f1\fs20{HIGH-LEVEL\~TERM\cell}
\pard\plain\intbl\keepn\sb20\sa20\qc\f1\fs20{\brdrb\brdrs Worst Grade \super a
\nosupersub \cell}
{\row}

\trowd\trkeep\trhdr\trqc\trgaph20
\clbrdrb\brdrs\brdrw9\clbrdrl\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx4399
\clbrdrb\brdrs\brdrw9\cltxlrtb\clvertalb\clcbpat8\cellx5185
\clbrdrb\brdrs\brdrw9\cltxlrtb\clvertalb\clcbpat8\cellx6818
\clbrdrb\brdrs\brdrw9\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx8466
\pard\plain\intbl\keepn\sb20\sa20\ql\f1\fs20{   Preferred Term\cell}
\pard\plain\intbl\keepn\sb20\sa20\qc\f1\fs20{Any\cell}
\pard\plain\intbl\keepn\sb20\sa20\qc\f1\fs20{3\cell}
\pard\plain\intbl\keepn\sb20\sa20\qc\f1\fs20{4\cell}
{\row}

\trowd\trkeep\trqc\trgaph20
\clbrdrl\brdrs\brdrw15\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalt\clcbpat8\cellx8466
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{\cell}
{\row}

\trowd\trkeep\trqc\trgaph20
\clbrdrl\brdrs\brdrw15\cltxlrtb\clvertalt\clcbpat8\cellx4399
\cltxlrtb\clvertalb\clcbpat8\cellx5185
\cltxlrtb\clvertalb\clcbpat8\cellx6818
\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx8466
\pard\plain\intbl\sb20\sa20\ql\f1\fs20{Any AEs - n(%)\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{19 (100)\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{6 (32)\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{9 (47)\cell}
{\row}

\trowd\trkeep\trqc\trgaph20
\clbrdrl\brdrs\brdrw15\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalt\clcbpat8\cellx8466
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{\cell}
{\row}

\trowd\trkeep\trqc\trgaph20
\clbrdrl\brdrs\brdrw15\cltxlrtb\clvertalt\clcbpat8\cellx4399
\cltxlrtb\clvertalb\clcbpat8\cellx5185
\cltxlrtb\clvertalb\clcbpat8\cellx6818
```

```
\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx8466
\pard\plain\intbl\sb20\sa20\ql\f1\fs20{HIGH-LEVEL TERM\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{17 (89)\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{3 (16)\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{2 (11)\cell}
{\row}

\trowd\trkeep\trqc\trgaph20
\clbrdrl\brdrs\brdrw15\cltxlrtb\clvertalt\clcbpat8\cellx4399
\cltxlrtb\clvertalb\clcbpat8\cellx5185
\cltxlrtb\clvertalb\clcbpat8\cellx6818
\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx8466
\pard\li240\lin240\plain\intbl\sb20\sa20\ql\f1\fs20{Event\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{17 (89)\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{3 (16)\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{2 (11)\cell}
{\row}

\trowd\trkeep\trqc\trgaph20
\clbrdrl\brdrs\brdrw15\cltxlrtb\clvertalt\clcbpat8\cellx4399
\cltxlrtb\clvertalb\clcbpat8\cellx5185
\cltxlrtb\clvertalb\clcbpat8\cellx6818
\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx8466
\pard\li480\lin480\plain\intbl\keepn\sb20\sa20\ql\f1\fs20{Severity 3\cell}
\pard\plain\intbl\keepn\sb20\sa20\qc\f1\fs20{16 (84)\cell}
\pard\plain\intbl\keepn\sb20\sa20\qc\f1\fs20{2 (11)\cell}
\pard\plain\intbl\keepn\sb20\sa20\qc\f1\fs20{1 (5)\cell}
{\row}

\trowd\trkeep\trqc\trgaph20
\clbrdrb\brdrs\brdrw15\clbrdrl\brdrs\brdrw15\cltxlrtb\clvertalt\clcbpat8\cellx4399
\clbrdrb\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx5185
\clbrdrb\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx6818
\clbrdrb\brdrs\brdrw15\clbrdrr\brdrs\brdrw15\cltxlrtb\clvertalb\clcbpat8\cellx8466
\pard\li480\lin480\plain\intbl\sb20\sa20\ql\f1\fs20{Severity 4\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{5 (26)\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{0 (0)\cell}
\pard\plain\intbl\sb20\sa20\qc\f1\fs20{1 (5)\cell}
{\row}

{\trowd\trkeep\trqc\trgaph10
\cltxlrtb\clvertalt\clcbpat8\cellx8466
\pard\plain\intbl\sb10\sa10\qr {\f1\fs20 Page 1 of 1\cell}
{\row}

\plain\intbl\sb10\sa10\ql {\f1\fs18 {\super a} This is a footnote. \cell} {\row}
\plain\intbl\sb10\sa10\ql {\f1\i\fs18 \cell} {\row}
\plain\intbl\sb10\sa10\ql {\f1\i\fs18 Program: /path/path/path/t_ae.sas, output:
t_ae.rtf \cell} {\row}
\pard\par}}
```