Paper CC-11-2013

Computing Counts for CONSORT Diagrams: Three Alternatives

David H. Abbott, Veterans Affairs Health Services Research

ABSTRACT

It is standard practice to document how the sample used for a study was determined – first showing all subjects considered and then the numbers of same that were successively removed from the cohort for various reasons. SAS® does not provide a specific facility for generating the tables/diagrams commonly used to present cohort determination information. Recently, Art Carpenter and Dennis Fisher published a paper showing how to create such diagrams given a template RTF file and the counts that pertain to each box in the template. However, their work did not address the question of how to derive the counts. We examine two commonly used approaches and provide a SAS macro to automate a third approach. This third approach has superior reproducibility, eliminates transcription errors, and makes changes to the diagram easier to effect.

The macro we present uses SAS data sets prepared by the user to convey the user's specifications for the counts to be produced and the logical expressions associated with each count. Thus, the macro is able to compute counts for almost any consort diagram. The technique of enabling the user to supply SAS expressions via a data set and thereby control program flow is a little used macro programming strategy with potentially broad application.

SETTING THE STAGE

Diagrams that show how an initial pool of experimental subjects (e.g. patients, customers, …) gets pared down to the smaller set of subjects that are randomized and/or analyzed are used extensively. These are referred to variously as: CONSORT diagrams, cohort selection diagrams, disposition flow diagrams, cohort determination diagrams, …. Here's an example diagram:

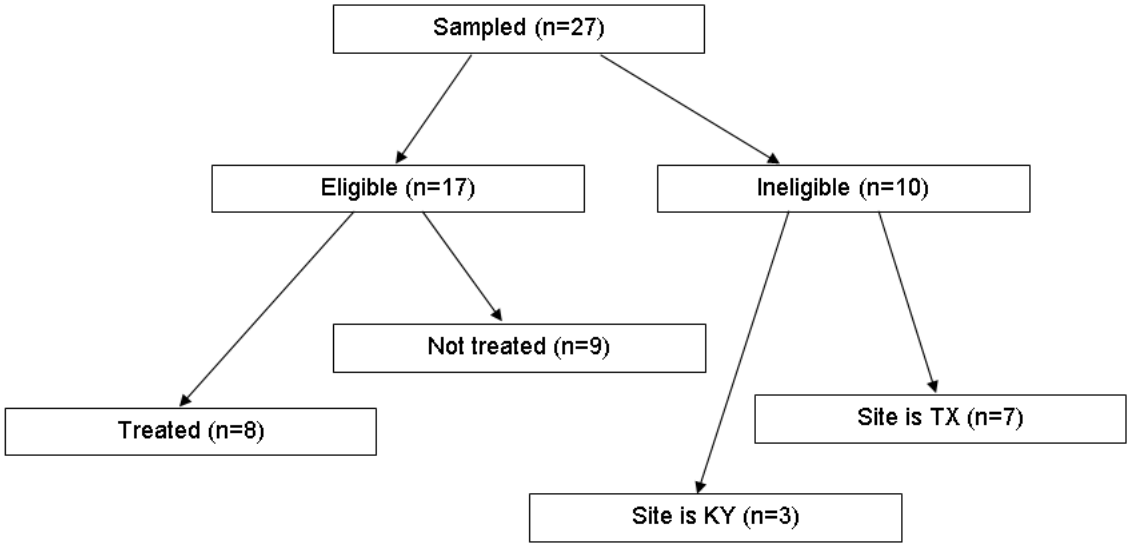

Figure 1. Example Cohort Selection Diagram

The information in these diagrams can also be represented in a table format and so can be referred to as CONSORT table, cohort selection table, etc. Whether diagram or table, the facts conveyed are the counts of subjects arranged in tree-like structure. A string variable called a "placeholder" serves to encode the route through the tree to each node. Here's a table that corresponds to the diagram above:

Obs	Node with Placeholder	Count
1	Sampled(n=_)	27
2	Ineligible(n=_0)	10
3	Site is TX(n=_00)	3
4	Site is KY(n=_01)	7
5	Eligible(n=_1)	0
6	Not treated(n=_10)	9
7	Treated(n=_11)	8

Table 2. CONSORT/Cohort Selection Table

The placeholder strings (called placeholders because they hold the place in the diagram for the count before it is calculated) start with an underscore to signify the root and the digits that follow indicate the branch (true/false) taken to reach each successor node leading to the given node.

The task of developing a CONSORT diagram can be divided into three steps:

1. Draw up a template for the diagram desired. The template is a diagram that looks just like the diagram desired except that it shows the placeholder rather than the count. Conceptually this means identifying the nodes of interest, their hierarchical layout, and the criteria to be applied at each node to generate successor nodes (usually two – the pass and fail branches of the criterion).
2. Compute the counts of subjects that apply to each node of interest and record these in a table like shown above with placeholders and counts.
3. Using the table and the template as inputs, replace the placeholders from step 1 with the counts computed counts from step 2.

Recently, Art Carpenter and Dennis Fisher discussed steps 1 & 3 in a well-received paper (Carpenter & Fisher, 2012). Other authors have likewise discussed how to develop an esthetically pleasing diagram given a template and a table of counts (Tran, 2008, Fairfield-Carter, 2011). However, the question of how best to compute the counts has received less attention. This paper examines three approaches to computing the counts needed for a CONSORT diagram:

1. The multiple DATA/PROC step approach – Uses one DATA/PROC step at each node of the diagram. This is usually the approach first tried by a programmer inexperienced with this task.

2. The single DATA step approach – all required counts are generated in a single DATA step whose nested IF/THEN/ELSE structure follows the tree structure of the diagram.

3. The specification-driven macro approach – The user prepares a data set that gives the logical expressions (i.e. selection criteria) used in computing the counts and a data set that lists the

nodes in the diagram and relates them to the logical expressions. The macro processes these specifications into the required table of counts.

MULTIPLE DATA STEP APPROACH

The notion with this approach is to start from the bottom of the tree and write a data step for each leaf in the diagram, and thereby generate temporary data set s that can be combined to form a temporary data set for the common parent node and so forth. The number of observations, the count of interest, is shown in the SAS log for each data step. We will use "StudySubjects" as the name of the input analytic data set describing the properties of each subject. Here's the code for our example:

```
DATA eligibleTreated;
  SET StudySubjects;
  WHERE eligible eq 1 AND treated eq 1;
RUN;
DATA eligibleNotTreated;
  SET StudySubjects;
  WHERE eligible eq 1 AND treated eq 0;
RUN;
DATA eligible;
  SET eligibleNotTreated eligibleTreated;
RUN;
DATA ineligibleKY;
  SET StudySubjects;
  WHERE eligible eq 0 AND site eq "KY";
RUN;
DATA ineligibleTX;
  SET StudySubjects;
  WHERE eligible eq 0 AND site eq "TX";
RUN;
DATA ineligible;
  SET ineligibleKY ineligibleTX;
RUN;
DATA sampled;
  SET ineligible eligible;
RUN;
```

This approach certainly works and is straight-forward. However, it does require the generation of numerous temporary data set s and generates a lot of code and DATA steps. Worse, the counts show up piecemeal in the SAS log rather than in a table of counts and must be transcribed manually into the consolidated table of results that is needed for implementing the replacement of the placeholders with the actual counts. Manual transcription is error-prone and complicates reproducing the results with certainty and ease. Finally, this approach produces code that is very specific to the details of the diagram template. If you decide you want to change the diagram template, e.g. use site as the first splitter (i.e. criterion) rather than the last, you pretty much have to re-implement from scratch.

SINGLE DATA STEP APPROACH

This approach uses a single DATA step to generate the counts required and encodes the structure of the diagram with a nested IF/THEN/ELSE structure rather than multiple DATA steps and DATA sets. It defines variables for each count required and converts the wide (lots of variables) data set thereby produced to a table where each row has a count and its corresponding label using PROC TRANSPOSE. We will use “SubjectCounts” as the name of the output data set in wide form and “SubjectsCountsTable” as the name of the output after being transposed. Here’s the code for our example:

```
DATA SubjectCounts;
  RETAIN fullSample ineligible ineligibleNotKY ineligibleKY
    eligible eligibleNotTreated eligibleTreated;
  SET work.StudySubjects  END=lastRec;
  fullSample+1;
  IF not eligible THEN DO;
    ineligible+1;
      IF site ne "KY" THEN ineligibleNotKY+1;
      ELSE ineligibleKY+1;
  END;
  ELSE DO;
    eligible+1;
      IF not treated THEN eligibleNotTreated+1;
      ELSE eligibleTreated+1;
  end;
  IF lastRec THEN OUTPUT;
  LABEL
    fullSample = "Sampled(n=_)"
    ineligible = "Ineligigle(n=_0)"
    ineligibleNotKY = "TX(n=_00)"
    ineligibleKY = "KY(n=_01)"
    eligible = "Eligible(n=_1)"
      eligibleNotTreated = "Not treated(n=_10)"
    eligibleTreated = "Treated(n=_11)";
  KEEP fullSample ineligible ineligibleNotKY ineligibleKY
    eligible eligibleNotTreated eligibleTreated;
RUN;
PROC transpose DATA=SubjectCounts OUT=SubjectCountsTable; RUN;
```

This approach, when combined with Carpenter and Fisher’s approach for replacing the placeholders, eliminates the transcription problem and doesn’t clutter up the WORK library with intermediate data set s. These are two big improvements! However, the DATA step code required is not trivial and can get tedious and confusing for more complicated diagrams. Again, this approach produces SAS code that is very specific to the details of the diagram template and requires a rewrite if you change the diagram template in consequential ways.

SPECIFICATION-DRIVEN MACRO APPROACH

The SAS macro facility is quite powerful and can be used to implement specification-driven algorithms that provide a general solution to certain problems. The problem at hand – generating the counts required by a CONSORT diagram – is solvable in this way. The key is to capture the defining specifications of a given CONSORT diagram in the form of two user-supplied SAS data sets – one for specifying the nodes of the diagram and another to specify the logic used in each node, i.e. the criteria for

splitting subjects to flow to one or the other of the descendent nodes. To describe it all in text gets a little complicated, so let's take a look at our worked example, starting with macro invocation:

```
%consortTable(
  subjectsDs=StudySubjects,
  predicatesDs=StudyCriteria,
  nodesListDs=DiagramNodes,
  tableOutDs=SubjectCountsTable,
);
```

The StudySubjects and SubjectCountsTable data sets are as described above. The StudyCriteria and the DiagramNodes data sets represent the specifications input to the macro. Here's a DATA step that creates the StudyCriteria data set:

```
DATA StudyCriteria;
  INFILE datalines DELIMITER='|';
  LENGTH predicate $ 32;
  INPUT predicate $ ;
  DATALINES;
eligible eq 1
treated eq 1
site eq 'KY'
RUN;
```

Although these three criteria are simple single-term SAS expressions, the consortTable macro will accept/process any valid SAS expression supplied via the "predicatesDs=" data set . For example, the expression, "`site in ('TX','GA','MS') or substring(site, 1, 1) eq 'N'`" would execute without issue and be true to the expressed logic.

Here's a DATA step that creates the DiagramNodes data set:

```
DATA DiagramNodes;
  INFILE datalines DELIMITER='|' DSD;
  LENGTH nodeSpec nodeDescript $32;
  INPUT nodeSpec nodeDescript $;
  DATALINES;

  _xxx| Sampled
  _0xx| Ineligible
  _0x0| Site is TX
  _0x1| Site is KY
```

```
  _1xx| Eligible
  _10x| Not treated
  _11x| Treated
RUN;
```

With this data set, the user is saying that he/she wants counts for 7 nodes and that:

- The first node is the root, i.e. all subjects, hence we don't care (x) how the 3 criteria evaluate.
- The second node is all subjects that evaluate false (0) for the first criterion (i.e.,"eligible eq 1") and the 2nd and 3rd criteria are "don't care".
- The third node is all subjects that evaluate false for the 1st criterion and the 3rd criterion (2nd is "don't care")
- Etc.

The placeholders in this approach can include x's as well as 1's and 0's and that confers advantages relative to just using _, 0, and 1 (as for the Single DATA Step approach):

1. The placeholder, with reference to the criteria sequence, fully defines the meaning of each node without recourse to the diagram template.
2. With the definition/meaning of the nodes uncoupled from the diagram it becomes easier to prepare alternative versions of the counts table and perhaps compare these before preparing the diagram template.

Note also that with the introduction of the x's the meaning of the underscore ("_") changes from indicating the root to just serving as a non-numeric prefix. Finally, we should point out that the macro is coded to work fine with either the hierarchical placeholder rep (no x's, underscore means root) or the placeholders with the x's. The user has his/her choice.

Here is a PROC PRINT of the count table that results (i.e. SubjectCountsTable):

Placeholder	Description	Count
_xxx	Sampled	27
_0xx	Ineligible	10
_0x0	Site is TX	3
_0x1	Site is KY	7
_1xx	Eligible	17
_10x	Not treated	9
_11x	Treated	8

Table 2. PRINT of SubjectCountsTable

Notice how the placeholder value clarifies the meaning of the count. For example, the count described as

'Site is "KY"' might be misunderstood to mean among all subjects, but the 2nd character of the placeholder is a "0" so this count is restricted to ineligible subjects only. Of course, when the placeholder for this count is replaced with "7" in the example diagram the meaning of this count is made clear by its location in the tree.

The specification-driven macro approach does not eliminate the complexity for the user of initially preparing or changing a CONSORT diagram but it does decrease the complexity of preparing the required counts. The user does not need to rewrite a sizable chunk of SAS code. Rather the user needs only to modify the contents of the two specification data set s - arguments predicatesDs and nodeListDs realized as StudyCriteria and DiagramNodes in the example above.

SUPPORT FOR PLACEHOLDER REPLACEMENT

The final step in the process of generating a CONSORT diagram is replacing the placeholders in the template with the count values from the count table. Following the general approach illustrated by Carpenter and Fisher, but accounting for the placeholder and description being two variables rather than a single string, we prepared a macro %consortDiagram to accomplish this final step. Here' an example invocation:

```
%consortDiagram(
  cntTable=SubjectCountsTable, /* as produced by ConsortTable */
  rtfIn=RtfFileI, /* diagram with placeholders, as a file reference */
  rtfOut=RtfFileO, /* the intended product, as a file reference */
 );
```

Recall that FILE references are created using the FILENAME statement and serve as an operating-system-agnostic reference to a physical file. After %consortDiagram completes its work, the .rtf file pointed to by RtfFileO produces the diagram shown at the start of the paper.

IMPLEMENTATION OF THE MACROS

The SAS code for the two SAS macros is given in the Appendix. The implementation of consortDiagram is an adaptation of the fillFields macro given in (Carpenter & Fisher, 2012). The implementation of consortTable is more complex since the logic of this macro is driven by the two specification files predicatesDs and nodeListDs.

The key notion in processing the predicatedDs is to create a sequence of macro variables to hold each predicate, i.e., each line of the predicatesDs data set, and then use a macro looping construct to create a line of code for evaluating each predicate and then storing the results in a sequence of binary (well, actually valued 0-9) variables in an augmented version of the studySubjects data set . Given these variables, it is logically straight-forward to determine if each study subject meets the conditions specified for a given results table node and increment node-specific counters (xx) accordingly. The complicated part of testing each study subject against each node is retaining access to information about each and all nodes as each study subject is processed. This is accomplished by reading through all rows of the predicatesDs and storing the contents in arrays of retained variables before starting in on processing the rows of the study subjects. After the last study subject has been processed, the accumulated counts are written out to the results data set.

CONCLUSION

Some closing remarks regarding this topic are:

- There is no way to totally escape tedious detail when preparing a complicated CONSORT diagram. However, the %consortTable and %consortDiagram macros do, we believe, make preparing the code, reading the code, modifying the code, documenting the execution, and repeating the execution easier than the basic approach.
- The %consortTable macro illustrates that macros need not be restricted to operating with purely pre-established logic. It is possible and beneficial to allow SAS expressions as input from the user thereby allowing SAS macros to provide general solutions to certain problems.
- One tricky aspect of using an .rtf file for the diagram template is that it is possible to edit the template, say to change a placeholder name, and find that the substitution for the placeholder fails when it "should" surely succeed. This is easily corrected – backspace out the placeholder and its prefix (usually "n=") and retype the string without backspacing and the substitution will then succeed. (It seems that the .rtf file carries around traces of past edits and these can interfere with value substitution.)
- This approach and the macros are not restricted to only handling binary trees. The expressions provided in predicatesDs can actually evaluate to any of 0,1,2,3,4,5,6,7,8,9; so trees with up to 10 successors from a given node can be handled successfully.

REFERENCES

- Carpenter, Art and Fisher, Dennis, 2012. "Reading and Writing RTF documents as Data: Automatic Completion of CONSORT Flow Diagrams." Proceedings of PharmaSUG2012, http://www.pharmasug.org/proceedings/2012/TF/PharmaSUG-2012-TF16.pdf

- Tran, Duong, 2008, "A Novel Approach to Patient Profiling", http://www.tranz.co.uk/PSI2008_WD_Patient_Profiling.pdf

- Fairfield-Carter, Brian and Suzanne Humphreys, 2011, "Alternative Approaches to Creating Disposition Flow Diagrams", proceedings of the Pharmaceutical SAS Users Group Conference (PharmaSUG), 2011, Cary, NC: SAS Institute Inc. http://www.pharmasug.org/proceedings/2011/TT/PharmaSUG-2011-TT08.pdf

ACKNOWLEDGMENTS

The views expressed in this paper are those of the author and do not necessarily reflect the position or policy of the Department of Veterans Affairs or the United States government.

Without the leadership and encouragement of Dr. Dawn Provenzale, director of the Durham Epidemiologic Research and Information Center at the Durham VA Medical Center, this work could not have occurred. She takes a strong interest in fostering many dimensions of excellence in her employees.

CONTACT INFORMATION

Name	David H. Abbott
Enterprise	Center for Health Services Research in Primary Care
Address	Durham Veterans Affairs Medical Center
	HSR&D Service (152)
	508 Fulton St.
City, State ZIP	Durham, NC 27705
Work Phone:	919-286-0411
E-mail:	david.abbott@va.gov

APPENDIX — SOURCE CODE FOR CONSORT TABLE MACROS

```
%MACRO ConsortTable(
  subjectsDs=, /* data set  with 1 obs per experimental unit, e.g. patient */
  predicatesDs=, /* predicates used in selecting experimental units,
                    the order of the predicates IS significant*/
  nodesListDs=,  /* template for the selection table, i.e. the list of
                    nodes of interest in the form _[01][01]... etc. where
                    the 1st 0,1 spec applies to the 1st predicate,
                    the 2nd 0,1 sped applies to the 2nd predicate etc.*/
  tableOutDs=,   /* table of counts for each node of interest */
  sp=tmp, /* scratch prefix */
  cleanUp=1,     /* removes temporary data set s and the like */
  allowAbort=1  /* can use to prevent MACRO from aborting on name conflict */
);
/*
    Version 3, dated 05/02/2013
    Author: David H. Abbott
    Changes: added "." as a don't care character in the node spec
    Remaining Issues:  order of criteria is, in a sense, arbitrary
       the order just has to match up with node spec element order
       the order in output table is simply order in nodespec data set
*/
  %LOCAL dsid npreds nnodes rc localString;

  %* Count obs in predicatesDs;
  %LET dsid=%SYSFUNC(OPEN(&predicatesDs));
  %LET npreds=%SYSFUNC(ATTRN(&dsid,NLOBS));
  %LET rc=%SYSFUNC(CLOSE(&dsid));
  %PUT Predicate count is &npreds;

  %* Check for name conflicts and make indicator variables LOCAL;
  %IF %SYSFUNC(exist(&sp.withIndicator)) eq 1 %THEN  %DO;
    %PUT ERROR: data set  &sp.withIndicator already exists, aborting;
    %IF  &allowAbort %THEN %RETURN;
  %END;
  %LET dsid=%SYSFUNC(OPEN(&subjectsDs));
  %LET localString=;
  %DO i=1 %TO &npreds;
    %LET localString= &localString %quote( ) &sp.&i;
    %IF %SYSFUNC(varnum(&dsid,&sp.indicator&i)) ne 0 %THEN  %DO;
      %PUT ERROR: variable &sp.indicator&i already exists in &subjectsDs,
aborting;
      %LET rc=%SYSFUNC(CLOSE(&dsid));
      %IF &allowAbort %THEN  %RETURN;
    %END;
  %END;
  %LET rc=%SYSFUNC(CLOSE(&dsid));
  %LOCAL &localString;

  * Check the format of nodesListDs;
  DATA tmp;
    SET &nodesListDs;
    loc = PRXMATCH("/^_[01x]* *$/", nodeSpec);
    IF  loc eq 0 THEN DO;
```

```
      PUT "ERROR: entries in &nodesListDs are improper format";
      PUT "       first detected at line num=" _N_;
      ABORT CANCEL;
    END;
  RUN;

  * PUT the predicates into a run of MACRO variables;
  DATA _null_;
    SET &predicatesDs;
    mvName= CATS("&sp", TRIM(PUT(_N_,4.)));
    CALL SYMPUTX(mvName, predicate, "L");
  RUN;

  %* Add indicator variables for each predicate;
  %MACRO tmpMacro;
  DATA &sp.withIndicators;
    SET &subjectsDs;
    ARRAY indicator{&npreds} indicator1-indicator&npreds;
    %DO j=1 %TO &npreds;
      indicator(&j) = &&&sp.&j; /*evaluates the predicate*/
      label indicator&j = "&&&sp.&j";
    %END;
  RUN;
  %MEND;
  %tmpMacro;

  %* Count obs in nodesListDs;
  %LET dsid = %SYSFUNC(open(&nodesListDs));
  %LET nnodes =%SYSFUNC(attrn(&dsid,NLOBS));
  %LET rc=%SYSFUNC(CLOSE(&dsid));
  %IF  &rc ne 0 %THEN  %PUT ERROR in consortTable: close failed;
  %PUT Number nodes of interest is &nnodes;

  %* No need to check for variable overwrite below since
    only newly derived fields are kept in the result DS;

  %* Compute the hierarchical counts;
  DATA &tableOutDs(keep=nodeNum theSpec theDescript theCnt);
    LENGTH descript1-descript&nnodes $32;
    RETAIN nodeSpec1-nodeSpec&nnodes descript1-descript&nnodes cnt1-
cnt&nnodes;
    SET    &sp.withIndicators(obs=max) END=lastRec;
    ARRAY indicVec{&npreds} indicator1-indicator&npreds;
    ARRAY nodeSpecVec{&nnodes} $32 nodeSpec1-nodeSpec&nnodes;
    ARRAY descriptVec{&nnodes} $32 descript1-descript&nnodes;
    ARRAY cntVec{&nnodes} cnt1-cnt&nnodes;
    IF _N_ eq 1 THEN DO;
      * remember nodes of interest;
      DO i=1 TO &nnodes;
        SET    &nodesListDs;
        nodeSpecVec(i) = nodeSpec;
        descriptVec(i) = nodeDescript;
        cntVec(i) = 0;
      END;
    END;
    /* Loop thru remembered stuff to find what is applicable to this obs */
```

```
    DO j=1 TO &nnodes;
      IF nodeSpecVec(j) eq "_" THEN DO;
        cntVec(j)+1;
      END;
      ELSE DO;
        countIt=1; *until evidence otherwise;
        * parse nodeSpec;
        partCnt= FLOOR(LENGTH(nodeSpecVec(j))-1);
        DO k=1 TO partCnt;
          part=SUBSTR(nodeSpecVec(j), k+1, 1);
          IF  part eq "x" THEN part = ".";
          specVal=INPUT(part,1.); /* 0 or 1, well might be 0...9 */
          IF specVal ne indicVec(k)and not MISSING(specVal) THEN countIt=0;
        END;
        IF countIt THEN cntVec(j)+1;
      END;
    END;
    IF  lastRec THEN DO;
      DO j=1 TO &nnodes;
        nodeNum=j;
        theSpec= nodeSpecVec(j);
        theDescript= descriptVec(j);
        theCnt= cntVec(j);
        OUTPUT &tableOutDs;
      END;
    END;
  RUN;

%IF  &cleanUp eq 1 %THEN  %DO;
  PROC DATASETS LIBRARY=work NOLIST;
    delete &sp.withIndicators  ;
  RUN;   QUIT;
%END;
%MEND;


/*
================================================================================
*/

%MACRO consortDiagram(
  cntTable=, /* as produced by ConsortTable */
  rtfIn=, /* has labels for substitution, as a file reference */
  rtfOut=, /* the intended product, as a file reference */
  prefix=n=,/*looks odd but &prefix resolves as desired to "n=" */
);
/*
  WARNING: the strings to be substituted with counts have the form
    &prefix.XYZ, e.g. n=XYZ
  due to peculiarities of RTF the string "n=XYZ" will only reliably
  occur in the RTF file IF  all the characters in the string are input
  as a single manual action. For example, it might not work to enter
  "n=XyZ" and THEN correct it to "n=XYZ".
*/
/*
```

```
    Version 2, dated 08/18/2013
    Author: David H. Abbott
*/
  %LOCAL theSpecList theCntList rowCnt i theSpec theCnt;

  * Load the values into MACRO variable lists;
  PROC SQL NOPRINT;
    SELECT theSpec, theCnt
    INTO :theSpecList separated by ',',
         :theCntList separated by ','
    FROM &cntTable;
    %LET rowCnt = &sqlobs;
  QUIT;

  DATA _null_;
    LENGTH tmp1 tmp2 $32;
    INFILE &rtfIn lrecl=3000;
    FILE &rtfOut lrecl=3000;
    INPUT;
    %DO i = 1 %TO &rowCnt;
      %LET theSpec =%SCAN(%BQUOTE(&theSpecList),&i,%STR(,));
      %LET theCnt =%SCAN(%BQUOTE(&theCntList),&i,%STR(,));
        tmp1 =CATS("&prefix","&theSpec","}");
        tmp2 =CATS("&prefix",COMPRESS("&theCnt",6.),"}");
        *IF  _N_ eq 1 THEN PUT "tmp values |" tmp1 "|" tmp2 "|";
      _INFILE_ = TRANSTRN(_INFILE_, TRIM(tmp1), TRIM(tmp2));
    %END;
    PUT _INFILE_;
  RUN;
%MEND;
```