```
%macro mr_pack
( in_data=
, out_data=MR_PACK
, var=col1
, nested_vars=
, nested_var_level = nested_var_level
, subnested_var =
, packvar_label =
, page_width_chars = &_default_page_width_chars
, col1_width_pcnt = &_default_col1_width_pcnt
, indent=_indent_
, offset=1
, cont_text = &_default_continued_text
, prcode_split = &_default_prcode_split
, escapechar = &_default_escapechar
, total_line_text =
, total_line_levels =
, total_line_indents =
, total_line_offset =
, add_total_line_if =
, add_total_line_indent = 0
, table_type = posttext
, intext_total_line_if =
, font_size = 9
, debug =
, help =
) /store  source des='V1.0.0.11';

%*************************************************************************
FILENAME:    MR_PACK.SAS
DEVELOPER:   (b) (6)
PLATFORM:    SAS 9.1.3, 9.2 on PC
MACROS USED: mu_wordscan packtext mu_setall
ASSUMPTIONS: _INDENT_ has been set for each record to be indented

DESCRIPTION:
This macro will
  1. run packtext against a variable in a nested counting data set
  2. create concatenated versions of upper level variables that can be
     used later in reporting as (continued) lines
  3. create a label for text column in the proc report


USAGE NOTES:

PARAMETERS:
IN_DATA         = the input data set (REQUIRED)
OUT_DATA        = the output data set.  (DEFAULT = &SYSMACRONAME) (CONDITIONAL)
VAR             = the text variable to run through packtext (REQUIRED)
(default=COL1))
```

```
NESTED_VARS     = the nested variables to run through packtext, eg, SOC PT
(CONDITIONAL - either VAR or NESTED_VARS must be provided)
NESTED_VAR_LEVEL = the variable in the input data indicating level of
summarization.
                  eg, in a table using SOC and PT, SOC is level 1 and PT is level 2
                  (CONDITIONAL - based on whether NESTED_VARS is provided)
SUBNESTED_VAR   = a categorical variable to be sub nested within each level of
summatization
                  eg, in an AE table with severity appearing as rows beneath each
level
                  (OPTIONAL)
PACKVAR_LABEL   = the text to use as the column header in the proc report
                  if left blank, the macro will build the label with the
                  labels of all the nested vars with the subnested var in the last
position
                  (OPTIONAL)
PAGE_WIDTH_CHARS= the width of the page in characters (REQUIRED)
COL1_WIDTH_PCNT = the width of the column where VAR will be displayed in percent
INDENT          = variable in input data set containing the number of spaces to
indent the value of VAR
OFFSET          = the indentation of wrapped lines - in the packtext call, this
                   would be INDENT2 where INDENT2 = INDENT1+OFFSET
CONT_TEXT       = the text to concatenate onto the column text to indicate
                   a continued level
PRCODE_SPLIT    = the split character in the proc report
ESCAPECHAR      = the ODS escape character
TOTAL_LINE_TEXT = the text for a Total row
                   eg, -Total, Any Event...
TOTAL_LINE_LEVELS = the levels (see NEST_VAR_LEVEL) which will recieve a separate
                  row for Total - the macro will concatenate &escapechar.n and the
                  TOAL_LINE_TEXT onto the levels specified.
TOTAL_LINE_INDENTS= if supplied then TOTAL lines will be indented these number of
spaces
TOTAL_LINE_OFFSET= the indentatin of the wrapped TOTAL lines
ADD_TOTAL_LINE_IF = Conditonal clause used to identify observations not otherwise
identified by NESTED_VAR_LEVEL
                        eg, in an AE tables with a top line count that is not part
of the nested level, the user
                        could identify the record(s) with COL1=: 'NUMBER OF
SUBJECTS'
                        (OPTIONAL) (NO DEFAULT)
ADD_TOTAL_LINE_INDENT = the number of spaces to indent the added total line
TABLE_TYPE      = Flag to indicate whether the output will be used in an in-ext or
posttext table.
                  If intext, no hard line feeds will be inserted via PACKTEXT.
Instead, RTF code for hanging
                  indents will be inserted at the beginning of each value of &VAR.
                  Also, no continued text will be generated.  Valid values are
P(OSTTEXT) and I(NTEXT)
                  (default = POSTTEXT)
```

```
INTEXT_TOTAL_LINE_IF = Conditional clause used to identify observartions to add the
TOTAL line to when producing intext tables
FONT_SIZE       = If generating indents for intext output, the font size is needed
in the calculation of the
                  number of twips in the indent. (DEFAULT=9)

*********************************************************************************
************;
%*-------------------------------------------------------------------------------
--*;
  %PUT ------------------------------------------------ ;
  %PUT INFO: (&SYSMACRONAME) ;
  %PUT INFO: Version 1.0 ;
  %PUT -START------------------------------------------ ;

%global &sysmacroname._rc;
%let &sysmacroname._rc = 0;

%** RETURN CODES
0= ran without error
1= output data set name not provided and defaulted
2= output data set already exists and will be deleted/overwritten
3= no split character supplied, therefore no splits in label
4= number of elements in TOTAL_LINE_LEVELS exceeds number of nested variables
5= number of elements in TOTAL_LINE_LEVELS not equal to the number of elements in
TOTAL_LINE_INDENTS
6= OFFSET missing or invalid and defaulted to 1
7= ADD_TOTAL_LINE_INDENT is missing, but ADD_TOTAL_LINE_IF is not, macro will set
indent to 0 spaces
8= ADD_TOTAL_LINE_IF did not find any records to add text to
9 =
10= FONT_SIZE is not provided, default to 9
;



%**** SECTION 1 - Setup and Preliminary Checks;

%global g_packvar_label g_packvar_dslabel ;

%* set help and debug;
%mu_help_debug *;
%let abort = NO;

%* get a list of the datasets in the WORK library before starting to enable cleanup
at
   the end of this macro;

%md_workinfo(
      debug        = &debug
```

```
    ,_workdata     = WORK_DATASETS_DATA
    ,_workview     = WORK_DATASETS_VIEW
    ,_mprinttoggle = mprint_setting
   ) *;



%****************** REQUIRED PARAMETERS***************;
%mu_check_req_parameters(
 parameters_to_check = IN_DATA VAR PAGE_WIDTH_CHARS COL1_WIDTH_PCNT INDENT
ESCAPECHAR TABLE_TYPE
,help = no
)

          data _null_;
            call symput("escapechar", compress("&escapechar", '.'));
          run;

        %*make col1_width_pcnt into a global macro variable so that it is available
to subsequent macros;
        %global g_col1_width_pcnt;
        %let g_col1_width_pcnt = &col1_width_pcnt;

        %*standardize the table_type parameter and check for valid values;
        %if %quote(&table_type) ne %str() %then
        %let table_type = %upcase(%substr(&TABLE_TYPE,1,1));
        %if &table_type ne P and &table_type ne I %then %do;
                %put ALERT_I: TABLE_TYPE (&TABLE_TYPE) not P(OSTTEXT) or I(NTEXT).;
                %put ALERT_I:   &sysmacroname will assume POSTTEXT;
                %let table_type = P;
        %end;

%***************** IN_DATA AND REQUIRED VARIABLES EXIST ************;
%mu_check_data_and_var_exist(
 data_to_check = &in_data
,vars_to_check_in_all_data = &var &nested_vars &subnested_var &indent
,vars_to_check_in_respective_data =
,abort_if_does_not_exist = yes
,help = no
)

%if "&nested_vars" ne "" %then %do;
        %mu_check_req_parameters(
         parameters_to_check = NESTED_VAR_LEVEL
        ,help = no)

        %mu_check_data_and_var_exist(
         data_to_check = &in_data
        ,vars_to_check_in_all_data = &nested_var_level
```

```
        ,vars_to_check_in_respective_data =
        ,abort_if_does_not_exist = yes
        ,help = no
)
%end;

%****************** CONDITIONAL PARAMETERS ************;
%if &OUT_DATA = %str() %then %do;
        %put ALERT_I: Missing parameter OUT_DATA.  Default to &SYSMACRONAME;
        %let out_data = &SYSMACRONAME;
        %let &sysmacroname._RC = 1;
%end;
%else %if %sysfunc(exist(&out_data)) %then %do;
        %put ALERT_I: OUT_DATA=&OUT_DATA.  Data set already exists and will be
deleted/overwritten;
        %let &sysmacroname._RC = 2;
        proc datasets mt=data lib=work;
          delete &out_data;
          run;
        quit;
%end;
%if %sysfunc(exist(packed_in_data)) %then %do;
        %put ALERT_C: &sysmacroname appears to have been run previously.;
        %put ALERT_C:   Interim dataset PACKED_IN_DATA created during ;
        %put ALERT_C:   that run will be deleted;
        proc datasets mt=data lib=work;
        delete packed_in_data;
        run;
        quit;
%end;




%** before doing anything else, get the sort order of the input data;
        %mu_get_sort_order(&in_data);

%** calculate the number of characters in column 1 that will be passed through
packtext;
        data _null_;
                char_width = ceil(&page_width_chars * (&col1_width_pcnt/100));
                call symput("char_width", trim(left(put(char_width, 8.))));
        run;

%** parse NESTED_VARS into macro variables;
        %mu_wordscan(string=&nested_vars , root=nest, numw=numnest)
        %if "&nested_vars" = "" %then %do;
         %let numnest = 0;
```

```
        %end;



%** parse TOTAL_LINE_LEVELS -- each nested level listed will get a line feed and
properly indented marker for addional `Total` text;
        %mu_wordscan(string=&total_line_levels, root=total, numw=numtotal)
        %put numtotal = &numtotal;

%** if offset is missing, default to 1;
        %if &offset = %str() %then %do;
          %put ALERT_I: OFFSET is missing.  &sysmacroname will default to 1 space;
          %let offset = 1;
          %let &sysmacroname._rc = 6;
        %end;
        %else %do;
          %* only accept integer values for offset;
           %let RE1=%sysfunc(prxparse(#\D#)) ;

            %if %sysfunc(prxmatch(&re1.,&offset))>0 %then %do ;
                %put ALERT_I: OFFSET (&offset) is non-integer.  &sysmacroname will
default to 1 space;
                %let offset = 1;
                %let &sysmacroname._rc = 6;
            %end ;
        %end;

%** the PRCODE_SPLIT is what is embedded into the column header and must match what
is
    used in the PROC REPORT as the split variable.  This will force the different
    labels of the nested variables onto separate lines in the column header;
        %if "&prcode_split" = "" %then %do;
                %put ALERT_I:  Parameter PRCODE_SPLIT is missing. ;
                %put ALERT_I:  No split codes will be embedded into the column
header;
                %let &sysmacroname._RC = 3;
        %end;
        data _null_;
                call symput('unquoted_prcode_split', compress("&prcode_split",
"'"));
        run;

%if &numnest gt 1 %then %do;
%** since you never have to put a continued onto the last nested variable, use one
less
    as the number of iterations ;
        %let numnest_forcont = %eval(&numnest -1);
%end;
```

```
%if &numtotal ge 1 %then %do;

        %* if the total_line_indents is blank then use the indents from the nested
vars;
        %if "&total_line_indents" eq "" %then %do;
          proc sql noprint;
           select distinct(_indent_) into :_indents_ separated by ' '
           from &in_data
           order by _indent_
           ;
          quit;
          %if "&_indents_" ne "" %then %do;
            %put _indents_ = &_indents_;
            %mu_wordscan(string=&_indents_, root=_indent_, numw=num_indent_)

            %do i= 1 %to &numtotal;
              %put total&i = &&total&i;
              %let total_line_indents = &total_line_indents %str( )
%sysfunc(scan(&_indents_, 1 + &&total&i));
              %put &i total_line_indents = &total_line_indents;
            %end;
             %put total_line_indents = &total_line_indents;

          %end;
        %end;
        %mu_wordscan(string=&total_line_indents, root=totindent, numw=numtotindent)

        %if &numtotal gt &numnest %then %do;
          %put ALERT_C: Number of levels to add TOTAL to exceeds number of nested
vars.;
          %put ALERT_C: No totals lines will be added;
          %let &sysmacroname._rc = 4;
          %let total_line_levels = ;
          %mu_wordscan(string=&total_line_levels, root=total, numw=numtotal)
        %end;
        %if &numtotal ne &numtotindent %then %do;
          %put ALERT_C: Number of levels to add TOTAL to not equal to number of
indents for those levels.;
          %put ALERT_C: No totals lines will be indented;
          %let &sysmacroname._rc = 5;
          %let total_line_indents = ;
          %mu_wordscan(string=&total_line_indents, root=totindent,
numw=numtotindent)
        %end;

        %if &total_line_offset = %str() %then %do;
          %put ALERT_I: TOTAL_LINE_OFFSET is missing.  Program will default to
OFFSET;
          %let total_line_offset = &offset;
        %end;
```

```
%end;

%if &numnest ge 1 %then %do;
%** SECTION 2 - ONLY FOR NESTED TABLES -- Create labels for the nested column;
  %*  sec2.1 - get the minimum _indent_ value from each nested_var_level;
        proc sql noprint;
                create table nested_indents as
                select &nested_var_level, min(_indent_) as indent
                from &in_data
                where not missing(&nested_var_level)
                group by &nested_var_level
                ;
                %let num_nested_indents = &sqlobs;

                create table subnested_indents as
                select min(_indent_) as indent
                from &in_data
                where not missing(&nested_var_level) and _indent_ ne 0
                ;

                select count(*) into :num_order from dictionary.columns
                where upcase(libname) = 'WORK'
                and upcase(memname) = upcase("&in_data")
                and upcase(name) like "_ORDER%"
                ;
                select count(*) into :num_subord from dictionary.columns
                where upcase(libname) = 'WORK'
                and upcase(memname) = upcase("&in_data")
                and upcase(name) = "_SUBORD%"
                ;
        quit;

  %* sec2.2 - set up macro variables with the number of dots/tildes(to be
translated to spaces) for each indentation level;
   %if &num_nested_indents gt 0 %then %do;
        %mu_nobs(nested_indents)
        %if &nested_indents_nobs lt 2 %then %do;
                data nested_indents;
                   retain indent -2;
                   do nested_var_level = 1 to &numnest ;
                        indent + 2;
                        output;
                   end;
                 run;

        %end;


        data _null_;
                set nested_indents;
```

```
                last_indent = lag(indent);
                if _n_ ne 1 then dots = indent-last_indent;
                else dots = indent*2;
                if dots gt 0 then do;
                        call symput("dots" || trim(left(put(&nested_var_level,
8.))) , repeat('.', dots-1));
                        call symput("spaces" || trim(left(put(&nested_var_level,
8.))) , repeat("~", dots-1));
                end;
                else if dots eq 0 then do;
                        call symput("dots" || trim(left(put(&nested_var_level,
8.))) , '');
                        call symput("spaces" || trim(left(put(&nested_var_level,
8.))) , '');
                end;
        run;
        data _null_;
                set subnested_indents;
                if indent gt 0 then do;
                        call symput("subdots", repeat('.', indent-1));
                        call symput("subspaces", repeat("~", indent-1));
                end;
                else if indent eq 0 then do;
                        call symput("subdots" , '');
                        call symput("subspaces" , '');
                end;
                else if indent eq . then do;
                        call symput("subdots" , '.');
                        call symput("subspaces" , '~');
                end;
        run;
   %end;
   %else %if &num_nested_indents eq 0 %then %do;
        data _null_;
                do indent = 0 to &num_order ;
                  last_indent = lag(indent);
                  if indent ne 0 then dots = indent-last_indent;
                  else dots = indent;
                  if dots gt 0 then do;
                        call symput("dots" || trim(left(put(indent, 8.))) ,
repeat('.', dots-1));
                        call symput("spaces" || trim(left(put(indent, 8.))) ,
repeat("~", dots-1));
                  end;
                  else if dots eq 0 then do;
                        call symput("dots" || trim(left(put(indent, 8.))) , '');
                        call symput("spaces" || trim(left(put(indent, 8.))) , '');
                  end;
                end;
                %if &num_subord ge 1 %then %do;
```

```
                call symput("subdots", repeat('.', dots));
                call symput("subspaces", repeat("~", dots));
                %end;
        run;
   %end;

  * sec2.3 -- open up the input data set to get its metadata;
        %let dsid_lib_dsin = %sysfunc(open(&in_data.));

        *get the attributes of the NESTED_VARS;
        %put numnest = &numnest;
        %do i = 1 %to &numnest;
            %let labelnest&i = %sysfunc(varlabel(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &&nest&i))));
            %if %quote(&&labelnest&i) = %str() %then %do;
                %put ALERT_I: NESTED_VAR &&nest&i is not labeled.  Macro will
assume name of variable;
                %let labelnest&i = &&nest&i;
            %end;
            %put labelnest&i = &&labelnest&i;
            %let lengthnest&i = %sysfunc(varlen(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &&nest&i))));
            %let typenest&i = %sysfunc(vartype(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &&nest&i))));
        %end;

        *get the attributes of the SUBNESTED_VAR ;
        %if %quote(&subnested_var) ne %str() %then %do;
            %let labelsubnest = %sysfunc(varlabel(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &subnested_var))));
            %if %quote(&labelsubnest) = %str() %then %do;
                %put ALERT_I: SUBNESTED_VAR &subnested_var is not labeled.  Macro
will assume name of variable;
                %let labelsubnest = &subnested_var;
            %end;
            %let lengthsubnest = %sysfunc(varlength(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &subnested_var))));
            %let typesubnest = %sysfunc(vartype(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &subnested_var))));
        %end;

        %let length_&var = %eval(%sysfunc(varlen(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &var)))) + 50);
        %put length_&var = &&length_&var;

        %let close_dsid_dsin = %sysfunc(close(&dsid_lib_dsin));


  * sec2.4 -- build macro variables consisting of the labels of each of the nested
variables and split characters;
```

```
        %* if packvar_label is missing then use default variable labels;
        %let packvar_dslabel = &packvar_label;
        %if "&packvar_label" = "" %then %do;
                %let indentation=;
                %let indentation_ds=;
                %do i = 1 %to &numnest;

                        %if &i gt 1 %then %let
indentation=&&escapechar..S={FOREGROUND=white}&&dots&i.&&escapechar..S={FOREGROUND=
black}&indentation;
                        %let packvar_label&i=&indentation.&&labelnest&i;

                        %let
packvar_label=&packvar_label.&&packvar_label&i.&unquoted_prcode_split.;
                        %put i=&i packvar_label=&packvar_label;

                        %if &i gt 1 %then %let
indentation_ds=&&spaces&i..&indentation_ds;
                        %let packvar_dslabel&i=&indentation_ds.&&labelnest&i;

                        %let
packvar_dslabel=&packvar_dslabel.&&packvar_dslabel&i.&unquoted_prcode_split;
                        %put i=&i packvar_dslabel=&packvar_dslabel;
                %end;
                %if &subnested_var ne %str( ) %then %do;
                        %let
indentation=&&escapechar..S={FOREGROUND=white}&subdots.&&escapechar..S={FOREGROUND=
black}&indentation;
                        %let indentation_ds=&subspaces.&indentation_ds;
                        %let packvar_label_subnest=&indentation.&labelSubNest;
                        %let packvar_dslabel_subnest=&indentation_ds.&labelSubNest;
                        %put packvar_label_subnest = &packvar_label_subnest;
                %end;
                %else %do;
                        %let packvar_label_subnest=;
                        %let packvar_dslabel_subnest=;
                %end;


                %let packvar_label=&packvar_label.&packvar_label_subnest;
                %let packvar_dslabel=&packvar_dslabel.&packvar_dslabel_subnest;

        %end;
        %else %if "%upcase(&packvar_label)" = "BLANK" %then %do;
                %let packvar_label = ;
                %let packvar_dslabel = ;
        %end;

        %* remove labels from the components of the packvar_label;
```

```
        %if "&packvar_label" ne "" or "&packvar_dslabel" ne "" %then %do;
                %do i = 1 %to &numnest;
                  proc datasets nolist;
                    modify &in_data;
                     label &&nest&i = ' ';
                    run;
                  quit;
                %end;
                %if %quote(&subnested_var) ne %str() %then %do;
                  proc datasets nolist;
                    modify &in_data;
                     label &subnested_var = ' ';
                    run;
                  quit;
                %end;

        %end;

        %* packvar label with RTF split codes;
        %let g_packvar_label = &packvar_label ;

        %* packvar label with proc report split codes;
        data _null_;
          length x $ 100;
          x = trim(left("&packvar_dslabel"));
          if substr(reverse(trim(left(x))), 1,1) = "&unquoted_prcode_split" then
          x = reverse(substr(reverse(trim(left(x))), 2));
          x = translate(x, ' ', '~');
          call symput("g_packvar_dslabel", trim(left(x)));
        run;
%end;



%else %if &numnest = 0 %then %do;


  *  sec2.1b - get the _indent_ values from VAR and SUBNESTED_VAR;;
        proc sql noprint;
                select min(_indent_), max(_indent_) into :minindent, :maxindent
from &in_data;          ;
        quit;

  * sec2.2b - set up macro variables with the number of dots/tildes(to be
translated to spaces) for each indentation level;
        data _null_;
                if &minindent gt 1 then do;
                        call symput("dots1" , repeat('.', &minindent.-1));
                        call symput("spaces1" , repeat("~", &minindent.-1));
                end;
```

```
                else if &minindent eq 0 then do;
                        call symput("dots1" , '');
                        call symput("spaces1" , '');
                end;
                if &maxindent gt 1 then do;
                        call symput("dots2" , repeat('.', &maxindent.-1));
                        call symput("spaces2" , repeat("~", &maxindent.-1));
                end;
                else if &maxindent eq 0 then do;
                        call symput("dots2" , '');
                        call symput("spaces2" , '');
                end;
        run;
  * sec2.3b -- open up the input data set to get its metadata;
        %let dsid_lib_dsin = %sysfunc(open(&in_data.));

        *get the attributes of VAR;
            %let labelnest1 = %sysfunc(varlabel(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &var))));
            %if %quote(&labelnest1) = %str() %then %do;
                %put ALERT_I: VAR (&var) is not labeled.  Macro will assume name of
variable;
                %let labelnest1 = &var;
            %end;
            %put labelnest1 = &labelnest1;
            %let lengthvar = %sysfunc(varlen(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &var))));
            %let length_&var = %eval(%sysfunc(varlen(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &var)))) + 50);
            %let typevar = %sysfunc(vartype(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &var))));


        *get the attributes of the SUBNESTED_VAR ;
        %if %quote(&subnested_var) ne %str() %then %do;
            %let labelnest2 = %sysfunc(varlabel(&dsid_lib_dsin,
%sysfunc(varnum(&dsid_lib_dsin, &subnested_var))));
            %if %quote(&labelnest2) = %str() %then %do;
                %put ALERT_I: SUBNESTED_VAR &subnested_var is not labeled.  Macro
will assume name of variable;
                %let labelnest2 = &subnested_var;
            %end;
        %end;

        %let close_dsid_dsin = %sysfunc(close(&dsid_lib_dsin));

  * sec2.4b -- build macro variables consisting of the labels and split characters;

        %* if packvar_label is missing then use default variable labels;
        %let packvar_dslabel = &packvar_label;
```

```
        %if "&packvar_label" = "" %then %do;
                %let indentation=;
                %let indentation_ds=;
                %if %quote(&subnested_var) eq %str() %then %do; %let x =  1; %end;
                %else %do; %let x =  2; %end;
                %do i = 1 %to &x ;

                        %if &i gt 1 %then %let
indentation=&&escapechar..S={FOREGROUND=white}&&dots&i.&&escapechar..S={FOREGROUND=
black}&indentation;
                        %let packvar_label&i=&indentation.&&labelnest&i;

                        %let
packvar_label=&packvar_label.&&packvar_label&i.&unquoted_prcode_split.;
                        %put i=&i packvar_label=&packvar_label;

                        %if &i gt 1 %then %let
indentation_ds=&&spaces&i..&indentation_ds;
                        %let packvar_dslabel&i=&indentation_ds.&&labelnest&i;

                        %let
packvar_dslabel=&packvar_dslabel.&&packvar_dslabel&i.&unquoted_prcode_split;
                        %put i=&i packvar_dslabel=&packvar_dslabel;
                %end;

        %end;
        %else %if "%upcase(&packvar_label)" = "BLANK" %then %do;
                %let packvar_label = ;
                %let packvar_dslabel = ;
        %end;

        %* remove labels from the components of the packvar_label;
        %if "&packvar_label" ne "" or "&packvar_dslabel" ne "" %then %do;
                  proc datasets nolist;
                    modify &in_data;
                     label &var = ' ';
                    run;
                  quit;
                %if %quote(&subnested_var) ne %str() %then %do;
                  proc datasets nolist;
                    modify &in_data;
                     label &subnested_var = ' ';
                    run;
                  quit;
                %end;

        %end;

        %* packvar label with RTF split codes;
        %let g_packvar_label = &packvar_label ;
```

```
        %* packvar label with proc report split codes;
        data _null_;
          length x $ 100;
          x = trim(left("&packvar_dslabel"));
          if substr(reverse(trim(left(x))), 1,1) = "&unquoted_prcode_split" then
          x = reverse(substr(reverse(trim(left(x))), 2));
          x = translate(x, ' ', '~');
          call symput("g_packvar_dslabel", trim(left(x)));
        run;




%end;
%* end sec2 ;


%* POSTTEXT processing;
%IF &TABLE_TYPE = P %then %do;


** SECTION 3 - create continued observations;
  * sec3.1 - determine legnths of each _order_ variable as if it were to be used in
continued;

   %if &numnest gt 1 %then %do;
        proc contents data=&in_data noprint out=contents(keep=name length );
        run;

        data _null_;
          set contents end=eof;
          call symput("length_"||trim(left(name)) , trim(left(put(length +
length("&cont_text") + 2, 8.))));

          %do i = 1 %to %eval(&numnest-1);
            if upcase(name) = upcase("&&nest&i") then newlength+length + 3;
          %end;
          if eof then do;
            newlength+length("&cont_text")+2;
            call symput("_newlength_", trim(left(put(newlength, 8.))));
            call symput("_dim_", trim(left(put(ceil(5 + newlength/&char_width),
8.))));
          end;
        run;
  %end;
  %else %if &numnest le 1 %then %do;
        proc contents data=&in_data noprint
             out=contents(keep=name length where=( upcase(name)=upcase("&var") ));
        run;
```

```
        proc sql noprint;
        select max(&indent) into :max_indent from &in_data;
        quit;

        data _null_;
          set contents end=eof;
          if upcase(name) = upcase("&var") then newlength+length + &max_indent;
          if eof then do;
            call symput("_newlength_", trim(left(put(newlength, 8.))));
            call symput("_dim_", trim(left(put(ceil(5 + newlength/&char_width),
8.))));
          end;
        run;
  %end;

%if &numnest gt 1 %then %do;
* sec3.2 - create duplicates of each _order_ var and append the "continued" as
needed;
        data in_data(drop=i marked n);
          set &in_data;
          by &sort_order;
          if &indent = . then &indent = 0;

           %* reset marked flag to 0;
          marked = 0;

          %* if the nested vars are equal to each other then
             add an RTF space to differentiate them;

               array nested(&numnest) &nested_vars;
                do n = 1 to dim(nested)-1;
                  if nested(n) ne '' and nested(n) = nested(n+1)
                    then nested(n) = trim(left(nested(n))) || "&escapechar.w";
                end;


           %* for each _order_ variable, compare to &VAR (the column1 variable).
              Set the duplicate variable to missing if the record is for that level
              Otherwise, set the duplicate variable to the variable itself;


          %do i = 1 %to &numnest_forcont;
          length _cont_&&nest&i $ &&&&length_&&nest&i;
          if &var = &&nest&i then do;
                _cont_&&nest&i = '';
                found = 1;
          end;
          else do;
                %do j = 1 %to &i;
                if not found then _cont_&&nest&j = &&nest&j;
```

```
            %end;
          end;
          %end;

            %* cycle through the new duplicates in backwards order.
               The first non-missing duplicate will get the continued marker
               and the marked flag will be set.
               These duplicates will be used to construct the "stacked" type column
               header;
          array cont{*} $ _cont_:;
          do i= dim(cont) to 1 by -1;
            if marked = 0 and cont(i) ne '' then do;
                cont(i) = trim(left(cont(i))) || " &cont_text";
                marked = 1;
            end;
          end;


        run;


        proc sort data=in_data out=continued_indent(keep=&indent) nodupkey;
        where &var = &nest2;
        by &indent;
        run;

%end;

%else %if &numnest le 1 %then %do;
        data in_data;
          set &in_data;
          by &sort_order;
          if &indent = . then &indent = 0;
        run;
%end;

%* end sec3.2;


%** SECTION 4 - send data through packtext for INDENTATIONS;
%* sec4.1 - determine the number of distinct indentations that are needed;
        proc sql noprint;
          create table indents as
          select distinct &indent
          from in_data
          order by &indent
          ;
          %let numindents = &sqlobs;
          select &indent into :indent1-:indent&numindents
          from indents
```

```
          ;

        %if &numnest gt 1 %then %do;
          select &indent into :continued_indent from continued_indent;
        %end;

        quit;
        %put numindents = &numindents;
        %do i=1%to &numindents;
        %if &indent&i = %str() or &indent&i = . %then %do; %let &indent&i = 0;
%end;
        %put &indent&i = &&indent&i;
        %end;



%* sec4.2a - cycle through each indentation level as a separate data set
   and send data through packtext;
        %do i = 1 %to &numindents;
          %let x = %eval(&i + 1);
          %if &i = &numindents %then %do;
                %let indent&x = .;
                %let total_indent&x = 0;
          %end;
          %else %do;
                %let total_indent&x = &&indent&x;
          %end;
          %let _length_ = %eval(&char_width - &&indent&i);
          %let _total_length_ = %eval(&char_width - &&total_indent&x);
          %put indent&i = &&indent&i;
          %put indent&x = &&indent&x;
          %put _length_ = &_length_;
          %put total_indent&x = &&total_indent&x;
          %put _total_length_ = &_total_length_;

          data indent&i(drop=dummy:);
                set in_data;
                by &sort_order;
                where &indent = &&indent&i;

                %*initialize _subord_ to missing;
                if _subord_= . then _subord_ = .;

                *  create PACK_&VAR -- column 1 data;
                %packtext
                ( varlist = &var
                , vartx = pack_&var
                , length =  &_length_
                , dim=&_dim_
                , delim = ' '
```

```
                , flow = "&escapechar.n"
                , linect = linect
                , indent1 = 0
                , indent2 = %eval(&&indent&i+&offset)
                , linout = dummya
                , sp_lline = 'N'
                , out_type = 'RTF'
                )


                %* add leading indentations;
                if &indent gt 0 then
                pack_&var = repeat(' ', &indent -1) || trim(left(pack_&var));


                %* reduce multiple splits to one;
                flag = 0;
                do until (flag = 1);
                  if index(compress(pack_&var),  compress("&escapechar n
&escapechar n"))  = 0
                  and
                  index(compress(pack_&var),  compress("&escapechar n \line"))  = 0
                  and
                  index(compress(pack_&var),  compress("\line &escapechar n "))  =
0
                  then flag = 1;
                  else do;
                        pack_&var = tranwrd(pack_&var, compress("&escapechar n
&escapechar n"), compress("&escapechar n"));
                        pack_&var = tranwrd(pack_&var, compress("&escapechar n
&escapechar n"), compress("&escapechar n"));
                        pack_&var = tranwrd(pack_&var, "\line", "");

                  end;
                end;


          run;

%end;



%* sec4.2b - set the separated indentation data sets back together;
        %mu_setall
        (in_data_list=%do i = 1 %to &numindents; indent&i %end;
        , out_data=packed_in_data
        , sort_from_data = &in_data
        , help=no)
```

```
%if &numindents gt 1 %then %do;
        %do i = 1 %to %eval(&numnest-1);
          data packed_in_data(drop=dummy:);
                set packed_in_data;
                by &sort_order;


                *  create PACK_CONT_&VAR -- column 1 data for continued lines;
                %packtext
                ( varlist = _cont_&&nest&i
                , vartx = pack_cont_&&nest&i
                , length =  %eval(&char_width - &&indent&i)
                , dim=&_dim_
                , delim = ' '
                , flow = "&escapechar.n"
                , linect = linect_colhead_cont&i
                , indent1 = 0
                , indent2 = %eval(&&indent&i+&offset)
                , linout = dummy&i
                , sp_lline = 'N'
                , out_type = 'RTF'
                )



          run;
        %end;

          data packed_in_data;
                set packed_in_data;
                by &sort_order;

                linect_colhead_cont =0;
                %* add leading indentations;
        %do i = 1 %to %eval(&numnest-1);

                linect_colhead_cont + linect_colhead_cont&i;


                if &&indent&i gt 0 then
                pack_cont_&&nest&i = repeat(' ', &&indent&i -1) ||
trim(left(pack_cont_&&nest&i)) || "&escapechar.n";
                else if &&indent&i eq 0 then
                pack_cont_&&nest&i = trim(left(pack_cont_&&nest&i)) ||
"&escapechar.n";

                  %* if the continued line is the same as the column1 text,
                     then continued is not needed, blank it out;
                   if compress(tranwrd(tranwrd(pack_cont_&&nest&i, "&cont_text",
```

```
""),  compress("&escapechar n"), ""))
                        = compress(tranwrd(pack_&var,  compress("&escapechar n"),
"") )
                    then pack_cont_&&nest&i = '';
                    if compress(pack_cont_&&nest&i) =compress("&escapechar n") then
pack_cont_&&nest&i = '';

           %end;
           %if &numnest gt 1 %then %do;
             length pack_colhead_cont_text $ 2000;
             pack_colhead_cont_text = substr('' %do i = 1 %to %eval(&numnest-1); ||
trim(pack_cont_&&nest&i) %end; , 2);
           %end;
        run;


%end;
```

```
%* sec4.3 - cycle through each total line indentation level as a separate data set
and send data through packtext;
   %if &numtotal gt 0 %then %do;
        %do i = 1 %to &numtotal;
          %let _total_length_ = %eval(&char_width - &&totindent&i);


          data total_indent&i(drop=dummy:);
                set packed_in_data;
                by &sort_order;
                where &nested_var_level = &&total&i and missing(_subord_) ;


                * create TOTAL_LINE_TEXT;
                temp_total_line_text = "&total_line_text";
                %* send the total line text through packtext ;
                %packtext
                ( varlist =  temp_total_line_text
                , vartx = total_line_text
                , length = &_total_length_
                , dim=&_dim_
                , delim = ''
                , flow = "&escapechar.n"
                , linect = linect_total_line_text
                , indent1 = 0
                , indent2 = %eval(&&totindent&i + &total_line_offset)
```

```
                , linout = dummyt
                , sp_lline = 'N'
                , out_type = 'RTF'
                )


                %* add Total lines as needed;
                    if missing(_subord_) then do;
                        if &&totindent&i gt 0 then total_pack_&var =
trim(pack_&var) || compress("&escapechar n") || repeat(' ', &&totindent&i -1) ||
total_line_text;
                        else if &&totindent&i le 0 then total_pack_&var =
trim(pack_&var) || compress("&escapechar n")  || total_line_text;
                        total_linect = linect + linect_total_line_text;
                    end;


        run;

        data packed_in_data(drop=total_pack_&var total_linect);
                merge packed_in_data total_indent&i(keep=&sort_order
total_pack_&var total_linect);
                by &sort_order;
                if not missing(total_pack_&var) then do;
                        pack_&var = total_pack_&var;
                        linect = total_linect;
                end;
        run;

    %end;

  %end;

%* sec4.3b - add total line text to obs specified by IF condition;
  %if "&add_total_line_if" ne "" %then %do;


    %if &add_total_line_indent = %str() %then %do;
      %put ALERT_I: ADD_TOTAL_LINE_IF specified;
      %put ALERT_I:   But, ADD_TOTAL_LINE_INDENT is missing;
      %put ALERT_I:   Macro will set indent to 0 spaces;
      %let add_total_line_indent = 0;
      %let &sysmacroname._rc = 7;
    %end;


        %let _total_length_ = %eval(&char_width - &add_total_line_indent );


        data add_total(drop=dummy:);
```

```
                set packed_in_data;
                by &sort_order;
                where &add_total_line_if ;


                * create TOTAL_LINE_TEXT;
                temp_total_line_text = "&total_line_text";
                %* send the total line text through packtext ;
                %packtext
                ( varlist =  temp_total_line_text
                , vartx = total_line_text
                , length = &_total_length_
                , dim=&_dim_
                , delim = ''
                , flow = "&escapechar.n"
                , linect = linect_total_line_text
                , indent1 = 0
                , indent2 = %eval(&add_total_line_indent  + &total_line_offset)
                , linout = dummytx
                , sp_lline = 'N'
                , out_type = 'RTF'
                )


                %* add Total lines as needed;
                    if missing(_subord_) then do;
                        if &add_total_line_indent gt 0 then total_pack_&var =
trim(pack_&var) || compress("&escapechar n") || repeat(' ', &add_total_line_indent
-1) || total_line_text;
                        else if &add_total_line_indent le 0 then total_pack_&var =
trim(pack_&var) || compress("&escapechar n") || total_line_text;
                        total_linect = linect + linect_total_line_text;
                    end;


        run;

        %mu_nobs(add_total)

        %if &add_total_nobs = 0 %then %do;
                %put ALERT_I:  ADD_TOTAL_LINE_IF (&add_total_line_if) specified ;
                %put ALERT_I:    did not find any records to add text to.;
                %let &sysmacroname._rc = 8;
        %end;
        %else %do;
          data packed_in_data(drop=total_pack_&var total_linect);
                merge packed_in_data add_total (keep=&sort_order total_pack_&var
total_linect);
                by &sort_order;
                if not missing(total_pack_&var) then do;
```

```
                        pack_&var = total_pack_&var;
                        linect = total_linect;
                end;
          run;
        %end;
 %end;


%* sec4.4 - remove trailing splits;

        data packed_in_data;
                set packed_in_data;

                %* remove trailing line feeds from pack_col1;

                if pack_&var = '' then pack_&var = '';
                if pack_colhead_cont_text = '' then pack_colhead_cont_text = '';

                if not missing(pack_&var) then do;
                  if reverse(compress(pack_&var)) =: compress("n &escapechar") then
do;
                    if length(compress(pack_&var)) gt 2 then pack_&var =
substr(trim(pack_&var), 1, length(trim(pack_&var))-2);
                    else if length(compress(pack_&var)) eq 2 then pack_&var = '';
                  end;
                end;
           %if &numnest gt 1 %then %do;
                if not missing(pack_colhead_cont_text) then do;
                  if reverse(compress(pack_colhead_cont_text)) =: compress("n
&escapechar") then do;
                    if length(compress(pack_colhead_cont_text)) gt 2 then
pack_colhead_cont_text = substr(trim(pack_colhead_cont_text), 1,
length(trim(pack_colhead_cont_text))-2);
                    else if length(compress(pack_colhead_cont_text)) eq 2 then
pack_colhead_cont_text = '';
                  end;
                end;
           %end;
        run;


%* sec5.0 - output to final dataset - maintain order from the input data and apply
label to packed variable;
        %mu_setall
        (in_data_list=packed_in_data
        , out_data=&out_data
        , sort_from_data = &in_data
        , help=no)

%if "&g_packvar_dslabel" ne "" %then %do;
```

```
        proc datasets mt=data lib=work;
                modify &out_data;
                label pack_&var = "&g_packvar_dslabel";
                run;
        quit;
%end;
%END; %*** end of posttext processing;

%ELSE %IF &TABLE_TYPE = I %THEN %DO; %*** start of intext processing;

%if &font_size= %str() %then %do;
        %put ALERT_I: FONT_SIZE is not specified to intext table.  Assume 9pt;
        %let font_size = 9;
        %let &sysmacroname._rc = 10;
%end;

data &out_data(sortedby=&sort_order);
        set &in_data;
        length pack_&var $ &&length_&var;
        if &indent gt 0 then
        pack_&var =
"&escapechar.R/RTF'\li"||compress(put(&font_size*12*(&indent+&offset), 8.)) ||
"\fi-" || compress(put(&font_size*12*(&offset), 8.)) ||"' " || &var;
        else
        pack_&var = "&escapechar.R/RTF'\li"||compress(put(&font_size*12*(&offset),
8.)) || "\fi-" || compress(put(&font_size*12*(&offset), 8.)) ||"' " || &var;


        %if %quote(&total_line_text) ne %str() and %quote(&intext_total_line_if) ne
%str() %then %do;
           if &intext_total_line_if then pack_&var = trim(pack_&var) ||
"&escapechar.n&total_line_text";
        %end;
        label pack_&var = "&g_packvar_dslabel";
run;


%end;  %*** end of in-text processing;



%* Sec6.0 - clean up;

%md_clean_and_reset
( debug      = &debug
,_workdata   = %str(&WORK_DATASETS_DATA &out_data)
,_workview   = %str(&WORK_DATASETS_VIEW)
,resetmprint = &mprint_setting
) *;
```

```
  %PUT ---------------------------------------------- ;
  %PUT INFO: (&SYSMACRONAME) ;
  %PUT INFO: Version 1.0 ;
  %put &SYSMACRONAME._RC = &&&SYSMACRONAME._RC;
  %PUT -END----------------------------------------- ;

%mend mr_pack;
```