PhUSE 2006
Paper PD01

Save Time: Program your Output Specs

Hansjörg Frenzel, PRA International, Mannheim, Germany
James Witt, PRA International, Mannheim, Germany

ABSTRACT
Your project has tight delivery timelines, but neither the statistical analysis plan nor the output specifications have been completed? In order to meet those timelines, you realize that it would be better if you could begin programming right away, but you don't have output specifications yet, and perhaps not even study data? One solution to help you meet those timelines is to begin programming anyway, and that is by using your eventual output programs to generate the output specifications, creating 'pre result versions' of the required outputs. This way, you can tackle and solve tough formatting challenges at the outset, even before derivation and aggregation of the data has been completed.
This paper discusses the pros and cons of preparing pre-result output files. The code introduced here is compatible with SAS® 8.2 onwards and only requires familiarity with SAS|BASE® and the ODS LISTINGS output destination.

INTRODUCTION
Within the CRO business shortened timelines and rescue projects lead to reductions in time for preparing and creating clinical study reports. Statisticians, SAS programmers and medical writers are the core members of the reporting team, and the main functions involved in this part of clinical study work. Optimizing the interfaces between these functions can help save both time and resources. This article focuses on the interface between statisticians and SAS programmers. When creating a statistical analysis plan (SAP) the statistician usually does a lot of work that is later repeated by SAS programmers. In a non-optimized environment the SAP and all output specifications are created using text processing software as a 'stand alone' bit of work that cannot later be accessed by the programs used to implement them. After the SAP has been finalized a great deal of redundant work is done if the programmer has to re-type into his or her programming environment all the information contained in the TFL-shells In addition, when creating the output specifications, the statistician may not be aware of SAS-inherent formatting constraints which may make it difficult to implement what he's specified, but which is relatively easy to present in a mock table created using word-processing software. The full costs of creating SAS generated tables, figures, and listings are then detected rather late during programming.

TASKS IN THE GRAY AREA BETWEEN STATISTICS AND SAS PROGRAMMING
Besides pure statistical (like input to protocol or SAP, or the statistical part of the clinical study report) and programming tasks (deriving data, generating output, QC) the work of a reporting team naturally entails many organizational and preparation tasks. Typically the lead programmer needs to set up specifications for the analysis database as well as for statistical output generation, which are not always part of the SAP. Results of these activities are the Data Definition Table (DDT) for the analysis database and the Table of Programs (TOP) for the statistical output. The TOP presents, among other things, the interdependence between the SAS programs and the output files these programs produce. Both statistician and lead programmer need to have a detailed knowledge about the study and how the data flow is organized. Usually output specifications, which provide detailed instructions about titles and footnotes, labeling of columns and a description of formats, are generated independently from TOP and DDT. Programming specs requires that the lead programmer and statistician work together on all three documents and use their knowledge to provide more detailed and better specifications.

JUSTIFICATION OF THE USE OF SAS PROGRAMS TO GENERATE OUTPUT SPECS
During the compilation of the specifications both statistician and lead programmer develop a deep understanding about the study and the requirements for the output. It's not unusual that, after the statistician and/or lead programmer have developed mock tables containing appropriate titles and footnotes, the programmer has to go back and re-enter these titles and footnotes in a program or spreadsheet (for example the TOP) to make them available for programming the actual outputs. After the SAP specifications have been agreed on, there are rougly three types of programming tasks that need to be done: 1) On the basis of the clinical database, an analysis (or "derived") database has to be programmed which contains all major and complex derivations and aggregations of the clinical data that are later to be displayed in the SAS output; 2) Statistical procedures are applied and algorithms are implemented for analyzing the data; and 3) The resulting procedure output is formatted for better legibility. Depending on the complexity of the output specifications, which are by and large determined by client requirements and which can vary greatly from client to client, the time required for this last formatting step can sometimes burn up to fifty percent of the entire programming effort. Complex formatting tasks can even bear a comparably high risk of incorrect display of data.

One approach to tackling the three programming tasks mentioned above is the following. After the output specs have been finalized, one writes the programs to generate the analysis database first, and then afterwards those programs for output programs. In the output programs both analysis and formatting algorithms are implemented. One advantage of this method is that the lead programmer does not need to describe a detailed interface between analysis output and formatting algorithms. The down side of this method is that, say, the junior programmer who has to implement the specifications is at a disadvantage compared to the lead programmer who wrote them: having spent some time on writing the specs and familiarizing himself with the clinical data, the lead programmer will usually have a clearer understanding of how the data should look that are to be displayed. One way to take advantage of this understanding would be for the lead programmer or statistician to *program* the specifications from the outset.

For a statistician who is proficient at using text processing software, at first glance it might seem awkward and inefficient to use SAS instead. Because it slows down the SAP writing process, this argument might be a stumbling block. However, programming the specifications could substantially reduce the amount of time needed for complex output formatting, and therefore, the costs as well. In addition, the influence of formatting requirements on programming time becomes obvious quite early in the study. Especially in the CRO business the leads as well as the programming team gain a more detailed understanding of the impact of lay-out on the costs early on when discussing the specifications with clients.

Consider also the situation, where time between finalization of the SAP and first draft of the SAS output is tight. Generally speaking, there are two ways to solve the associated resource problems: Either the programming team works weekends and overtime, or additional resources have to be added to the team. The former solution results in tired, and thus error-prone programmers, as well as in higher costs for the company as that overtime needs to be compensated. Besides the increase of costs, the latter solution of adding more people to the project team also increases the amount of effort needed for communication, which can also result in misunderstandings and again in errors. The break between planning on the one hand and implementation on the other also puts pressure on both the statistician to the complete the SAP and on the lead programmer to complete the TOP and DDT earlier. It is not unlikely that the specifications will contain a couple of inaccuracies which lead to discussions and re-work during programming.

It should also be mentioned that the order in which specifications are created changes if your output specifications are programmed: While the TOP and the DDT are created after the output specifications, if these are ‘Paper specifications’, all three parts are developed simultaneously when the output specs are programmed. It is expected that this leads to more accuracy of all three specification documents.

There are factors that do need to be taken into consideration before you make the decision to program the specifications. If for example there are already programs or specifications available which require only little adaptation, the upfront effort involved in programming the specs will probably not be worthwhile. It may also be too risky to do so if for whatever reason the probability that the project could be stopped is higher than normal. Additional factors that need taking into account are how often the shells are expected to be amended, how familiar one is with the client team’s expectations, to what extent the limitations for the output formatting are well understood by the client, and just how complex the output formatting is anticipated to be. Complex output it may be awkward to program ahead of time, but the team would at least have the advantage of developing a clear understanding of the complexity and costs.

ASPECTS OF WORK THAT CAN BE SHIFTED TOWARDS THE PLANNING PHASE

Once the decision has been made to program the output specs, a couple of tasks can be accomplished before “real” data is available.

PROGRAM NAMES AND SAS PROGRAM TEMPLATES

Within our programming group the part of the TOP that contains titles, footnotes, output file names and SAS program names is usually available within a SAS dataset. This dataset is extracted from a Microsoft® Excel® workbook each time the information is updated. Because the names of all programs in the project are stored in this TOP dataset, it can be used to generate program templates or frames for SAS programs (containing a header and supporting information and macro calls) so that the implementing programmer doesn’t have to start from scratch. In situations where it’s desirable to use a particular procedure or standardized macro, we enter this in the TOP and use this information to create in the program template an appropriate call which the programmer then need only modify.

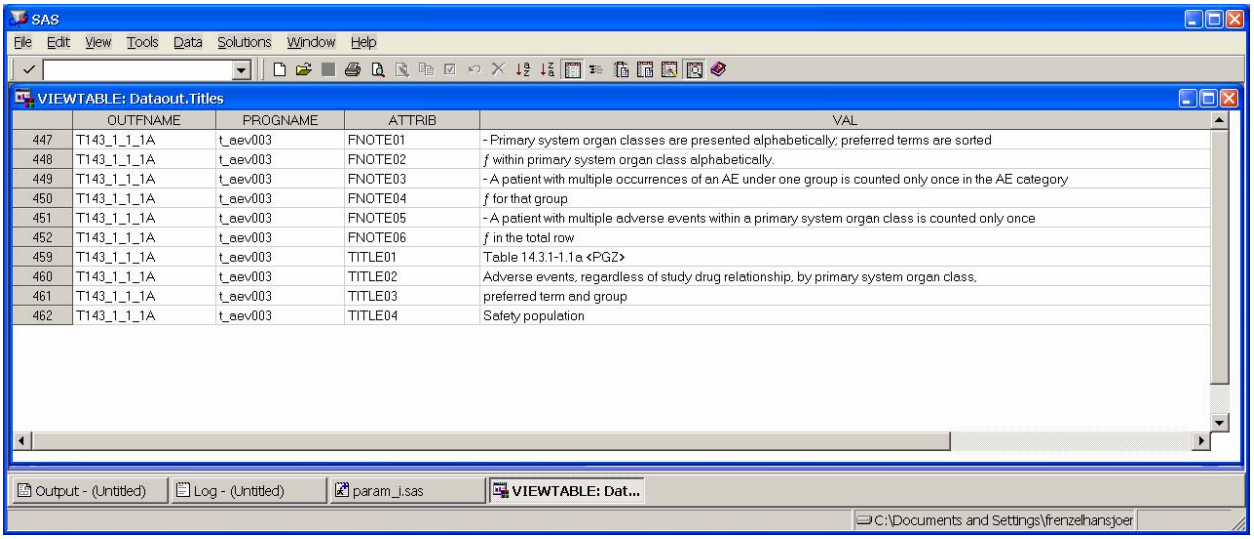
REUSING TITLES AND FOOTNOTES

Because titles and footnotes are included in the TOP dataset they can be retrieved within each program during run time, and do not need to be coded in the output programs themselves. This makes for easier maintenance and less QC effort. The TOP dataset also contains the name of the file which is to be written to, and this info is also retrieved during run-time and not hard-coded in the program, allowing for more flexibility if naming conventions or output order should change.

The TOP dataset contains four variables:

- OUTFNAME: Name of the output file
- PROGNAME: Name of the SAS program that generates the output
- ATTRIB: Contains identifies e.g. for footnote (FNOTExx) or title (TITLExx) rows
- VAL: Contains the values associated to output file and attribute.

Key variables are OUTFNAME and ATTRIB. Nevertheless the PROGNAME information is necessary to determine the relationship between SAS program and output files. Below a screenshot that helps to understand what kind of information the TOP dataset contains (note that titles and footnotes information are only two kinds of several attributes that are stored here and used to maintain the output; additional ones are discussed below).



	OUTFNAME	PROGNAME	ATTRIB	VAL
447	T143_1_1_1A	t_aev003	FNOTE01	- Primary system organ classes are presented alphabetically; preferred terms are sorted
448	T143_1_1_1A	t_aev003	FNOTE02	ƒ within primary system organ class alphabetically.
449	T143_1_1_1A	t_aev003	FNOTE03	- A patient with multiple occurrences of an AE under one group is counted only once in the AE category
450	T143_1_1_1A	t_aev003	FNOTE04	ƒ for that group
451	T143_1_1_1A	t_aev003	FNOTE05	- A patient with multiple adverse events within a primary system organ class is counted only once
452	T143_1_1_1A	t_aev003	FNOTE06	ƒ in the total row
459	T143_1_1_1A	t_aev003	TITLE01	Table 14.3.1-1.1a <PGZ>
460	T143_1_1_1A	t_aev003	TITLE02	Adverse events, regardless of study drug relationship, by primary system organ class,
461	T143_1_1_1A	t_aev003	TITLE03	preferred term and group
462	T143_1_1_1A	t_aev003	TITLE04	Safety population

The macro that retrieves the titles and footnotes information uses the output file name to select the records and also selects all rows that contain the string “TITLE” in the ATTRIB variable. The title number is derived from the numbers provided in the TOP dataset, but there is an option available which signals that title or footnote number should begin incrementing at a certain number (&starttit), in case there are common, output-independent titles or footnotes for each kind of output (tables, figures, and listings).

```
/* titles */
data inds (drop=_charno  titleno val attrib);
  length _charno $3 ;
  set &inds (where=(upcase(outfname)="%upcase(&selfile)" and
                   index(ATTRIB,'TITLE')>0)) end=fin;
     _charno=substr(trim(left(ATTRIB)),6);
     titleno=input(_charno,best.);
     _titleno= &starttit. + titleno ;
     titletxt=trim(left(val));
     output;
run;
```

A simple CALL EXECUTE is used to generate titles statements:

```
data _null_ ;
  set inds end=eof;
  call execute('title' || compress(put( _titleno, best.)) ||
               ' "' || trim(left(titletxt )) || '";');
  if eof then put "SETTING TITLES FOR &SELFILE. ....";
run;
```

which creates the following statements:

```
title1 "Table 14.3.1-1.1a <PGZ>" ;
title2 "Adverse Events, regardless of study drug relationship, by …" ;
title3 "preferred term and group" ;
title4 "Safety population" ;
```

Footnotes are retrieved in a similar way.

ORGANIZING THE OUTPUT

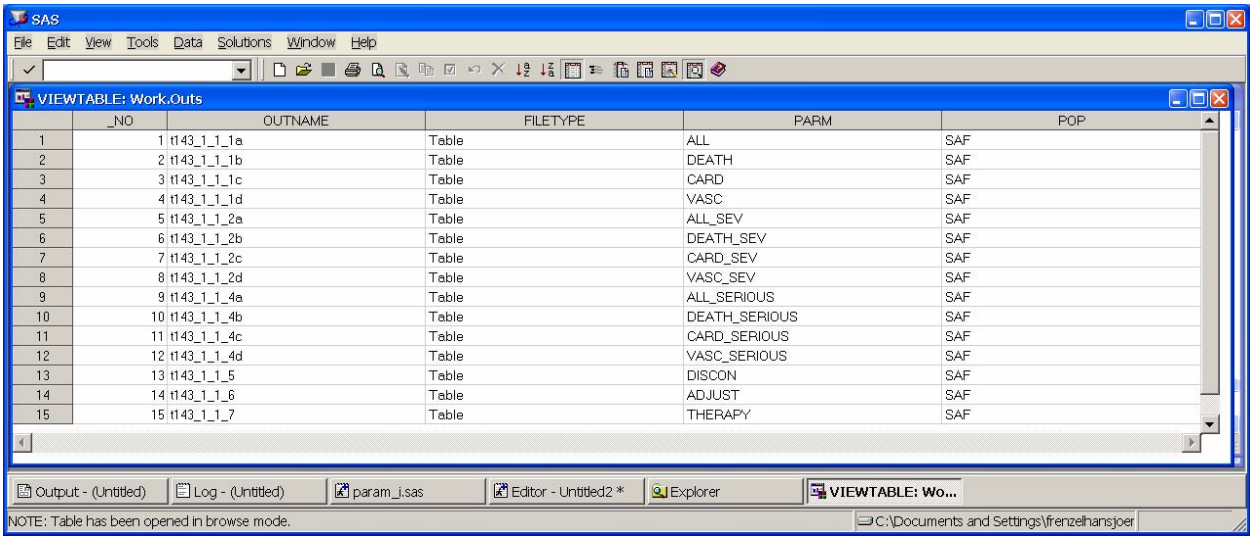
The TOP dataset also contains the name of the SAS program generating the output file; this information is also retrieved during run-time and not hard coded in the program. A macro solves this task by generating a dataset containing the attributes of all the outputs produced by the program that produces them: e.g. the output file name (OUTNAME), the population (POP) being presented, a parameter that can be assigned to identify an output file associated to a program (PARM – useful for identifying, e.g., repeated laboratory outputs), as well as a variable containing the type of the output (FILETYPE), i.e. whether the output is a table, listing, or figure. The variable SORTNO is an independent numeric key which is used for the transpose step.

```
proc sql;
  create table _&myoutds.1 as  /* e.g. outfiles1 */
        select val
               ,attrib
               ,sortno
               ,outfname
          from &mytitles. /* the TOP datasets */
         where compress(upcase(progname)) in ("&myprog.") /* e.g. t_aev003 */
               and compress(upcase(attrib)) in ('OUTNAME','POP','PARM','FILETYPE')
      order by sortno, attrib
   ;
quit;

/* generates a new dataset (e.g. outfiles2) with a transposed structure */
proc transpose data=_&myoutds.1 out=_&myoutds.2 (drop=_NAME_);
  var val;
  id attrib;
  by sortno;
run;

data &myoutds. (drop=sortno);  /* e.g. outfiles */
  retain _NO OUTNAME FILETYPE PARM POP;
  set _&myoutds.2 (where=(&mywhere.));
  _no=_N_;
run;
```

The structure of the output file of this macro is displayed below. If a table shall be produced for more than one population the variable POP can be used to select the appropriate output file name.



	_NO	OUTNAME	FILETYPE	PARM	POP
1	1	t143_1_1_1a	Table	ALL	SAF
2	2	t143_1_1_1b	Table	DEATH	SAF
3	3	t143_1_1_1c	Table	CARD	SAF
4	4	t143_1_1_1d	Table	VASC	SAF
5	5	t143_1_1_2a	Table	ALL_SEV	SAF
6	6	t143_1_1_2b	Table	DEATH_SEV	SAF
7	7	t143_1_1_2c	Table	CARD_SEV	SAF
8	8	t143_1_1_2d	Table	VASC_SEV	SAF
9	9	t143_1_1_4a	Table	ALL_SERIOUS	SAF
10	10	t143_1_1_4b	Table	DEATH_SERIOUS	SAF
11	11	t143_1_1_4c	Table	CARD_SERIOUS	SAF
12	12	t143_1_1_4d	Table	VASC_SERIOUS	SAF
13	13	t143_1_1_5	Table	DISCON	SAF
14	14	t143_1_1_6	Table	ADJUST	SAF
15	15	t143_1_1_7	Table	THERAPY	SAF

To retrieve the name of a specific output file it is sufficient to select the row with the correct PARM (and POP) value, e.g. if the output file name for the table containing “All Adverse Events” shall be retrieved the following code is implemented into the program:

```
/* %gettifo generates titles statement */
data _null_;
   set outs(where = (parm = "ALL "));
   call symput('outsnam', outname);
run;
```

Afterwards the output file name (“T143_1_1_1A”) is stored into macro variable &OUTSNAM and used in a proc printto (or corresponding ODS statement).

WHEN FORMATS CAN BE PREPARED

Obviously all formats that are needed to display data or summaries in listings and tables can already be created during the specification phase. When programming the specifications, the advantage is that these formats are already quality-controlled before programming begins.

PROVIDING MOCK DATA

When programming specifications, one critical point is the absence of real data. In order to facilitate the programming of the specifications, mock data are needed. It is crucial to have a good specification of the dataset that serves as input for he formatting part of the program. For clinical study output there are two types of data set needed: One type has the structure of data in the derived datasets, the other type has the structure of output datasets of SAS statistical procedures (or statistical macros that are available within the company). Whereas it is simpler to ‘manually’ provide mock data that can be displayed in listings, it is dangerous to provide mock data for statistical output manually, because this can lead to inconsistencies between the shell programs, which then lead to re-work when adapting the shell programs to ‘real-life’ programs. Obviously, a programmer should be readily familiar with the statistical procedures and macros involved when generating mock data for tables, because the naming of the variables which are handed over to the formatting part are key in this step. Having a macro with a clearly defined user interface makes it easier to generate mock data in a structured manner. In the following, two macros are sketched: one which creates analysis dataset-like data, and one to create statistical macro-like data.

GENERATING DATA FOR LISTINGS

Generating mock data as a stand-in for real analysis datasets requires the programmer to specify every variable that is necessary to set up the formatting part of a data listing program. In order to do this, the programmer and the statistician need to work together on the DDT at this point in the workflow. This work is made easier if a tool is available that provides a structured interface for the generation of mock derived datasets:

```
%mklist( dsname = demog ,
         patid = %str(length patid $ 10 ; patid="XXX/XXXXX" ; ) ,
         nrec  = 18 ,
         visvar = vis_var ,
         visits = 1 ,
         visfmt = visit ,
         vislabel = Visits ,
         lisspec = @var = treat
                   @type = cat
                   @form = trt
                   @label = Treatment code |
                   ...
                   @var = age
                   @type = num
                   @range = 18 - 75
                   @form  = 2
                   @label = Age of Patient |
                   @var = comment
                   @type = text
                   @form = $40
                   @range = 3 - 12
                   @label = Investigator´s Comment ) ;
```

The macro creates three types of SAS variables: continuous numerical variables (@type=NUM), categorical numeric variables (@type=CAT), and text variables (@type=TEXT). All three types of variables are generated within one data step. Within the dataset &nrec records * number of elements specified in the &visits variable are generated. The current value of a variable is generated using a random number function:

- For continuous variables the following algorithm is used: multiply the random number by the range specified in the @range parameter and add the lower limit. The format specified in the @form option is used to determine the precision of the number.
- For categorical variables the number of categories is multiplied by the random number, afterwards the category values are assigned using an array.
- For text variables: Random character strings are generated. The length of these strings is specified with the @range option. The length of the variable is controlled with the format provided to the @form option.

All these calculations are done using separate macros.

Continuous variables:

```
%macro mkcont(seed=,range=,form=,name=,label=) ;
  *** initialize variables *** ;
  length &name 8 ;
  randovar = abs( ranuni ( &seed )) ;
  formvar = compress( "&&form&i" ) ;
  *** calculate the precision *** ;
  exponent = 0 ;
  pointpos = index( formvar , '.' ) ;
  formlength = length( formvar ) ;
  if 0 < pointpos < formlength then
    exponent = input( scan( formvar , 2 , '.' ) , 8. ) ;
  *** generate the variable and the values *** ;
  &name = ( int( (10**exponent) * randovar *
              (%scan(&range,2,-) - %scan(&range,1,-) + 1) ) +
              (10**exponent * %scan(&range,1,-))) / 10**exponent ;
  label &name = "&label." ;
%mend mkcont ;
```

A format is used in order to determine the precision of the resulting data. The value to the right of the dot, which describes the precision, is loaded into the variable EXTENSION. For example, if values in format 8.2 are to be generated, the variable EXTENSION gets the value 2 assigned. Afterwards, the random number multiplied by the given range and also with 100 (10^2). Then the lower limit is added, again multiplied by 100. The integer is generated, and afterwards the result is divided by 100 again. Here an example: One wants to generate values for weight with format 8.1 within a range of 40 to 80 kg. First, a random number is generated, let's say 0.5555. Then 0.55 is multiplied with 40 (the range), which is equal to 22.22. This value is multiplied by 10 ($=10^1$), which equals 222.2. After the integer is calculated (222) the lower limit, again multiplied by 10, is added: result is 422. After everything is divided by 10, the resulting value is 42.2, which lies within the range and shows one digit after the comma.

Categorical variables:

```
%macro mkcat (form=,name=,label=);
    *** preliminary steps to drop the blanks from string combinations *** ;
    %local numform arname formlist ;
    %let arname = %sysfunc( compress( ar&form )) ;
    %let formlist = %sysfunc( compress( &form )) ;
    *** count the number of entries in the array *** ;
    *** and load the array *** ;
    retain numcat ;
    length catlist $ 200 ;
    catlist = "&&&formlist" ;
    numcat = 1 ;
    do until( scan( catlist , numcat , ',' ) eq '' ) ;
      if scan( catlist , numcat /*+ 1*/ , ',') eq '' then leave ;
      &arname(numcat) = input( compress(scan( catlist , numcat , ',')), best. ) ;
      numcat + 1 ;
    end ;
    if numcat > 1 then numcat = numcat - 1 ;

    drop numcat randovar catlist;

    randovar = ranuni( %sysfunc(datetime(),24.)) ;
    &name= &arname.( int( randovar*numcat ) + (randovar < 1)) ;

    label &name = "&label" ;
%mend mkcat ;
```

The first action in this macro involves loading an array which is initialized at the beginning of the data step in which the macro is called. The task of this macro is to generate a variable which contains only values that are specified in a numeric format. For example, if a format SEX is specified, with values 1=Male and 4=Female, then the resulting variable SEX should contain only values 1 and 4 Outside the macro a macro variable is generated that contains the category values specified in the format that is used to generate the variable. These macro variables are named exactly like the format. For our example the macro variable is named &SEX and contains the value "1, 4". To assign the correct values to the variable SEX a random number is generated which is multiplied by the number of the categories specified within a format. Because the format SEX has two categories, the random variable (which is in the range from 0 to 1) can have values from 0 to 2. The integer is calculated, and afterwards a 1 is added if the random value is less than 1. To assign the expected values, the array ARSEX is used. This array has the first category value

("1") as in the first array element, and the second one ("4") as in the second array element. The statement SEX = ARSEX(INT(randovar * 2) + (randovar < 1)) assigns a "1" to SEX, if the randomly generated value is "1", and a "4" is assigned if the randomly generated value is "2".

Text variables:

```
%macro mktext(name=,form=,range=, label=);
  length &name $ &form. ;
  drop cntrl randovar;

  &name = ' ' ;
  cntrl = 0 ;

  do while( length( trim( left( &name ))) lt
            input( compress( "&form." , '$. ' ) , best. )) ;
    cntrl + 1 ;
    randovar = ranuni( %sysfunc(datetime(),16.)) ;
    &name = trim( left( &name )) || ' ' || repeat( 'X' ,
            int( randovar*(%scan(&range,2,-) - %scan(&range,1,-) + 1)) +
            %scan(&range,1,-)) ;
    if cntrl gt 200 then leave ;
  end ;

  label &name = "&label." ;
%mend mktext ;
```

The length of the variable is specified by the format assigned to the option @form. If the programmer specifies the format as $200., then the length of the variable will be 200. The option @range is used to specify the length of the elements of the text string. For example, if a programmer wants text strings having lengths varying between 3 and 12 characters length, he would specify @range=3 – 12. The variable is then filled with strings of the length between 0 + 3 and 9 + 3 (this is generated using a random variable).

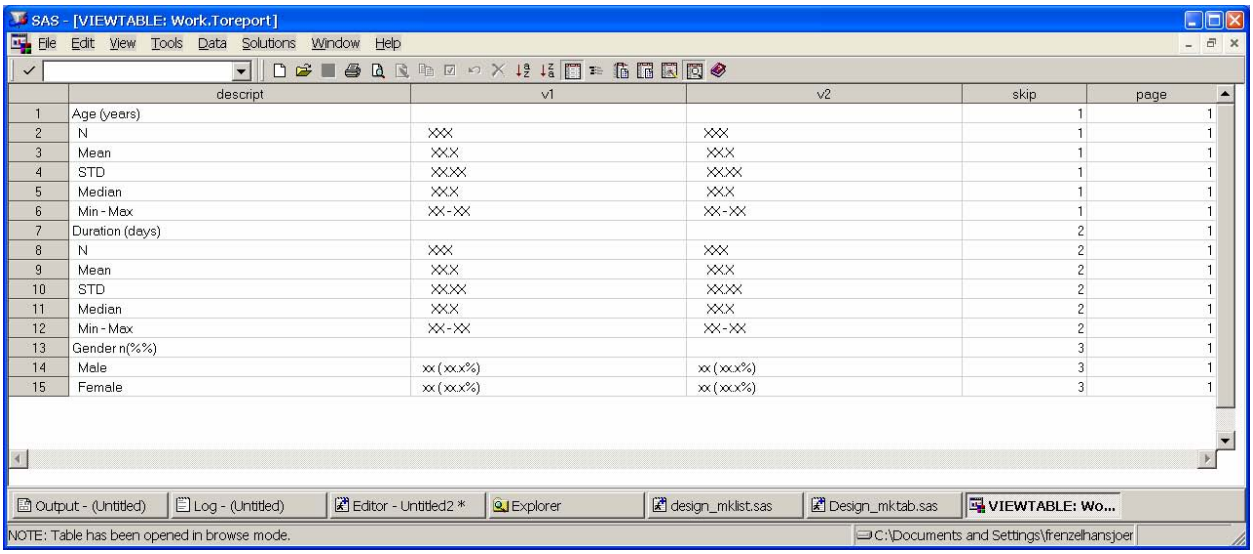
To generate datasets, the corresponding formats are first retrieved from the format dataset and loaded into a SAS dataset. Second, within a data step for each of the specified variables in the LISSPEC – parameter (using the @var= option), one of the three macros above is called within the data step. The macro %SPLITIT is used to process the info specified in the LISSPEC parameter. This macro is described in ESKEN, GROSS, LESKOPF, FRENZEL (2005).

GENERATING DATA FOR TABLES

For tables, the programmer needs a tool that creates datasets in the structure provided by statistical procedures or macros. Again a structured interface is used:

```
%mktab( dsname  = dem_tab ,
        trt     = trt ,
        trtsel  = %str( input( compress( start ) , best. ) < 99 ) ,
        tabspec = @type   = DESCR
                  @rows   = desc1_
                  @label  = Age (years)
                  @cols   = descs1_ |
                  @type   = DESCR
                  @rows   = desc1_
                  @label  = Duration (days)
                  @cols   = descs1_ |
                  @type   = CAT
                  @rows   = sex1_
                  @label  = Gender n(%%)
                  @cols   = cnt1_ | ) ;
```

So let's assume a statistical macro produces a dataset with the following structure: DESCRIPT is a character variable which contains labels to describe the variable that is displayed, variables V1 up to Vn contain the result statistics for each treatment group (the values for n are derived from the variable assigned as treatment variable). SKIP and PAGE are available to distinguish blocks of information and to control the presentation of the output. To fill the DESCRIPT numeric formats are used. The formats specify which categories are displayed. Again there is a difference between categorical and continuous variables. The following is an example generated for a demography table:



	descript	v1	v2	skip	page
1	Age (years)			1	1
2	N	XXX	XXX	1	1
3	Mean	XXX	XXX	1	1
4	STD	XXXX	XXXX	1	1
5	Median	XXX	XXX	1	1
6	Min - Max	XX-XX	XX-XX	1	1
7	Duration (days)			2	1
8	N	XXX	XXX	2	1
9	Mean	XXX	XXX	2	1
10	STD	XXXX	XXXX	2	1
11	Median	XXX	XXX	2	1
12	Min - Max	XX-XX	XX-XX	2	1
13	Gender n(%%)			3	1
14	Male	xx (xx.x%)	xx (xx.x%)	3	1
15	Female	xx (xx.x%)	xx (xx.x%)	3	1

Age and Duration are continuous variables, Gender is a categorical variable. For both types of variables different algorithms are used, which are implemented in different macros. For each variable in the output dataset the macro creates one dataset. These datasets are joined afterwards. The single datasets are derived using the format catalog rather then from artificial data. For the demography table shown above the following formats were specified:

```
value desc1_
    1 = 'N'
    2 = 'Mean'
    3 = 'STD'
    4 = 'Median'
    5 = 'Min - Max'
    ;
  /* study specific formats */
  value trt
    1 = 'Treatment A'
    2 = 'Treatment B'
   99 = 'Total'
   other = 'No Treatment assigned'
   value sex1_
     1 = 'Male'
     2 = 'Female'
     ;
```

The format trt. is used to generate the columns, the formats DESC1_ and SEX1_ are used to specify the rows for continuous variables (AGE and DURATION) and the categorical variable (SEX).

CONTINUOUS VARIABLES

```
    *** to generate dataset rows for continuous variables *** ;
    %macro mkdscr( rows= , numtrt= , cols= , label= ,
                   lcolblank=YES , dsname= , skip= , page=1) ;
      %local i ;
      *** get blanks out of MV parameters *** ;
      data _null_ ;
        length parvalue $ 100 ;
        parvalue = "&rows." ;
        call symput( 'rows' , compress( parvalue ) ) ;
        parvalue = "&cols." ;
        call symput( 'cols' , compress( parvalue ) ) ;
        parvalue = "&label." ;
        call symput( 'label' , trim( left( parvalue )) ) ;
        parvalue = "&dsname." ;
        call symput( 'dsname' , compress( parvalue ) ) ;
      run ;
```

```
  *** fill the cols with structure specified in format &COLS. *** ;
  data &dsname ;
    set col1fmt ;          /* dataset is downloaded from catalog work.formats */
    length descript $ 40 cols 8 ;
    descript = label ;     /* the LABEL var. of the COL1FMT dataset */
                           /* contains the description              */
    where upcase( fmtname ) = "%upcase(&rows)" ;
    /* categories specified in the format are loaded into variable COLS */
    cols=input( compress( start ) , best. ) ;
    %local colfmt ;
    %let colfmt = %scan( &cols , 1 , # ) ;
    %do i = 1 %to &numtrt ; /* treatment columns are generated here */
      %if %scan( &cols , &i , # ) ne %then
        %let colfmt = %scan( &cols , &i , # ) ;

      length v&i $ 20 ;
      v&i = put( cols , &colfmt.. ) ;
    %end ;
    keep descript cols %do i = 1 %to &numtrt ; v&i %end ; ;
  run ;

  *** add a label to the categories *** ;
  data &dsname ;
    set &dsname ;
    retain skip &skip page &page ;
    descript = '  ' || left( descript ) ;
    if _N_ = 1 then do ;
      output ;
      cols = 0 ;
      descript="&label" ;
      %if %upcase( &lcolblank ) = YES %then %do ;
        %do i = 1 %to &numtrt ;
          v&i = ' ' ;
        %end ;
      %end ;
      output ;
    end ;
    else output ;
  run ;

    proc sort data =&dsname out=&dsname (drop=cols ) ; by cols ; run ;
  %mend mkdscr ;
```

The algorithm is relatively simple. A dataset (WORK.COL1FMT) is retrieved from a format catalog and a format is selected to display the summary for a continuous variable. The format should contain the description of the statistics that are given by the statistics macro rather than labels that are associated with a variable itself. In our example the format DESC1_ contains the labels for 'Number of non-missing records', 'Mean', 'Standard Deviation', 'Median', 'Minimum', and 'Maximum'. The labels are stored in the variable LABEL, the category values (1-5) are stored in variables START and END. The contents of LABEL is assigned to variable DESCRIPT, the values in START are used to derive the contents of the treatment variables V1 – Vn, and are stored as numeric values in variable COLS. To derive these variables additionally the format DESCS1_ is used, which is displayed below.

```
  value descs1_
    1 = '     XXX     '
    2 = '      XX.X   '
    3 = '      XX.XX  '
    4 = '      XX.X   '
    5 = '     XX - XX'
    ;
```

Each treatment variable is filled using the following algorithm: Vn = put(cols, descs1_.) ;. If different formats are to be assigned to different columns they can be assigned to the @form option, separated by a '#'. The last format name is always retained, if only one format is assigned it will be used for all columns.

CATEGORICAL VARIABLES

```
*** to generate dataset rows for categorial variables *** ;
%macro mkcnt( rows= , numtrt= , cols= , label= , lcolblank=YES ,
              dsname= , skip= , page=1) ;
  %local i ;
  *** get blanks out of MV parameters *** ;
  … see above …

  *** generate lines with the structure described in MV COLS ***;
  data &dsname ;
    set col1fmt ;
    length descript $ 40 cols 8 catcol $ 40 ;
    retain catcol ;
    catcol = put( 1 , &cols.. ) ;
    descript = label ;
    where upcase( fmtname ) = "%upcase(&rows)" ;

    cols=input( compress( start ) , best. ) ;
    %do i = 1 %to &numtrt ;
      length v&i $ 20 ;
      v&i = catcol ;
    %end ;
    keep descript cols %do i = 1 %to &numtrt ; v&i %end ; ;
  run ;

  *** add a label to the categories and       *** ;
  *** wipe out the result cols - if requested *** ;
  data &dsname ;
    set &dsname ;
    retain skip &skip page &page ;
    descript = '  ' || left( descript ) ;
    if _N_ = 1 then do ;
      output ;
      cols = 0 ;
      descript="&label" ;
      %if %upcase( &lcolblank ) = YES %then %do ;
        %do i = 1 %to &numtrt ;
          v&i = ' ' ;
        %end ;
      %end ;
      output ;
    end ;
    else output ;
  run ;

  proc sort data =&dsname out=&dsname (drop=cols ) ; by cols ; run ;
%mend mkcnt ;
```

Summaries for categorical variables are generated similarly to continuous variable summaries. Again a data set is retrieved from the format catalog, but this time the format SEX1_ is used. The variable DESCRIPT contains the categories contained in the variable LABEL of the format dataset. But because an unlimited number of categories is possible with categorical variables, the treatment columns are generated differently. First, a different format is used to fill these columns. In our example it is format CNTL1_:

```
value cnt1_
  1 = '   xx ( xx.x%)'
 ;
```

For each column and each row a variable CATCOL is populated using the following statement: CATCOL = put(1 , CNT1_.) ;. Because CATCOL is a retain variable, each record in the output datasets will contain the same value. The treatment variables are then subsequently created by means of the CATCOL variable: Vn = CATCOL; .

CONCLUSION

Programming your output specifications might help you better meet tight timelines. Besides time constraints there may be other factors that influence your decision to do so, but one should not completely disregard the possibility as a viable means of increasing efficiency and quality. The advantages are obvious: More communication between statistician and programmer, more detailed (and hopefully better) specifications, a better knowledge of the influence of formatting challenges on time and costs, and the opportunity to QC the formatting parts of programs before “actual” programming on the data begins, only to mention a few. Clever supporting tools, such as a stringent programming framework that aligns the documentation (TOP and DDT) with programming and macros that allow to creation of mock data in a structured manner facilitate this attempt.

There are risks associated with this method: Updating the specifications requires more time than when composed using simple text processing software. Also, some effort may have been in vain if at the last minute the scope of the analysis reduced or the study itself cancelled.

Due to space constraints, the macros %MKTAB and %MKLIST are not completely presented in this paper, but can be received through the authors or from the common download sites.

REFERENCES

Esken, W., Leskopf, W., Gross, Ch., Frenzel, H (2005): PREFTAB – a macro to automatically create adverse events tables. Paper PO21, PharmaSUG 2005.

ACKNOWLEDGMENTS

CONTACT INFORMATION

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

Hansjörg A. Frenzel
PRA International
Dynamostr. 13-15
D-68165 Mannheim
Germany
Phone: +49 (0)621-8782-328

James E. Witt
PRA International
Dynamostr. 13-15
D-68165 Mannheim
Germany
Phone: +49 (0)621-8782-280