PhUSE 2017

Paper CT01

Reducing SAS Dataset Merges with Data Driven Formats

Paul Grimsey, Roche Products Ltd, Welwyn Garden City, UK

ABSTRACT

Merging different data sources is necessary in the creation of analysis datasets. The downsides are that these increase dependencies in the code, can sometimes cause unexpected results (if not careful) and with large datasets could have a large processing time. Data driven formats provide a good strategy to reduce the number of merges required in analysis dataset creation, help to create cleaner, easier to read code and help programmers easily make additions to already existing complex programs. This paper will show a simple, ready to use macro incorporating proc format to create data driven formats and examples of how this can be used in dataset creation.

INTRODUCTION

PROC FORMAT allows programmers to create their own SAS formats and informats by either specifying the values, or a range of values directly in the code with the VALUE / INVALUE statements, or by reading in a dataset with the CNTLIN option. The resulting FORMAT / INFORMAT can then be used as an alternative to the MERGE statement in the DATA step for a ‘one to one’ or ‘one to many’ join. Combining data in this way has been shown to be quicker and take up less memory than by using the MERGE statement or SQL join [1] [2].

THE CNTLIN OPTION IN PROC FORMAT

PROC FORMAT can build a format or informat from a dataset using the CNTLIN option. The following is an example of the syntax where the dataset INDS is used to build the format or informat:

```
    proc format library=work.formats cntlin=inds;
    run;
```

In addition to the CNTLIN option there is also a CNTLOUT option. The following example will output all of the existing formats in the WORK.FORMATS catalogue to a dataset called OUTDS:

```
    proc format library=work.formats cntlout=outds;
    run;
```

The CNTLOUT dataset has a structure that can be copied to produce our own formats/informats. Although CNTLOUT contains 21 different columns, not all of these are required to make a basic format/informat. We will use 5 out of the 21 columns, as this provides enough functionality for our purposes. The 5 columns to be used are [3]:-

1. **FMTNAME -** specifies a character variable whose value is the format or informat name. This will be the name of our data driven format / informat.
2. **START -** specifies a character variable that gives the range's starting value. This will be the input value for the format / informat.
3. **LABEL -** specifies a character variable whose value is associated with a format or an informat. This will be the formatted value of START.
4. **TYPE -** specifies a character variable that indicates the type of format. This determines whether we create a format or informat and if this is character or numeric: C = a character format; N = a numeric format; I = a numeric informat; J = a character informat. P is also an available option and is used to specify a picture format.
5. **HLO -** specifies a character variable that contains range information about the format or informat. We will use this to provide a value for what to do if an unexpected value for START exists, using option ‘O’ (‘other’). Some other valid values include: H = a range's ending value is HIGH; I = a numeric informat range; L = a range's starting value is LOW; U = that the UPCASE option for an informat be used.

CREATING CNTLIN - EXAMPLE

Let’s imagine we would like to create a format for weight at baseline. We can use SDTM datasets VS and EX to get the last weight measurement in kilograms (VSSTRESN) before the first active treatment per patient (USUBJID) to make a dataset called WTBLPRE:-

USUBJID	VSSTRESN
12345-001-001	38.2
12345-001-002	48.2
12345-001-003	39.1
12345-001-004	78.5

This dataset could then be transformed as shown below into a dataset called WTBL, which may be used to create our character format. We will call the format WTBLF:-

```
data wtbl (keep = fmtname type start label);
  set wtblpre;
  fmtname = "WTBLF";
  type    = "C";
  start   = usubjid;
  rename vsstresn = label;
run;
```

The WTBL dataset has the following structure:

FMTNAME	TYPE	START	LABEL
WTBLF	C	12345-001-001	38.2
WTBLF	C	12345-001-002	48.2
WTBLF	C	12345-001-003	39.1
WTBLF	C	12345-001-004	78.5

These 4 columns would be enough for SAS to make a character format called WTBLF using the CNTLIN option in PROC FORMAT. However, if we apply this format to a value of START not found in the above dataset then the unformatted value of START will be used as the formatted value, which could be undesirable. To account for unexpected values we can use the HLO variable, give it a value of ‘O’ for ‘Other’ and set LABEL to be the desired value for HLO:-

```
data other;
  fmtname = "WTBLF";
  type    = "C";
  hlo     = "O";
run;
```

Note that in the above syntax we did not specify a value for LABEL. As we want to set LABEL to be missing / empty this can be done automatically during the next step, so we don’t need to concern ourselves with whether the LABEL is a numeric or character variable. We now combine the WTBL and OTHER datasets together to produce our final dataset called WTBLF:-

```
data wtblf;
  set wtbl
      other;
run;
```

The structure of the dataset WTBLF looks like this:-

FMTNAME	TYPE	START	LABEL	HLO
WTBLF	C	12345-001-001	38.2	
WTBLF	C	12345-001-002	48.2	
WTBLF	C	12345-001-003	39.1	
WTBLF	C	12345-001-004	78.5	
WTBLF	C		.	O

The WTBLF dataset can be used to create a character format in the WORK.FORMATS library by using the CNTLIN option of PROC FORMAT:-

```
proc format library=work.formats cntlin=wtblf;
run;
```

We can apply the format in our code. Here is an example that could be used in a DATA step:-

```
weightbl = input(put(usubjid, wtblf.), best.);
```

Note that WEIGHTBL is created as a numeric variable and so we need to add an INPUT statement around the PUT.

A MACRO FOR CREATING THE DATA DRIVEN FORMATS

Our next step is to take the code shown previously and develop this into a macro which will create the format automatically for us from an input dataset. To make things simpler, we only need 2 variables in this dataset:-

1) The value we wish to match
2) The formatted value of the value we wish to match.

We will also make the dataset name and the variable holding the formatted value the same name as the format we wish to produce. This will simplify the inputs which are required for the macro. An example of this would be our WTBLPRE dataset containing USUBJID and VSSTRESN. We would change the dataset name to be WTBLF (to match our format name) and rename VSSTRESN as WTBLF (to also match our format name):-

Dataset name: WTBLF

USUBJID	WTBLF
12345-001-001	38.2
12345-001-002	48.2
12345-001-003	39.1
12345-001-004	78.5

We can then create the macro using 3 parameters:-

1. **FNAME** = the CNTLIN variable FMTNAME. This will be the name of the format, the input dataset name and the name of the column holding the formatted values
2. **FTYPE** = the CNTLIN variable TYPE
3. **FSTART** = the CNTLIN variable START

Adapting the code from the CNTLIN creation example we can use the following code blocks as the basis for our macro:-

```
data &fname.dpre (keep = fmtname type start label);
  set &fname;
  fmtname = "&fname";
  type    = "&ftype";
  start   = &fstart;
  rename &fname = label;
run;
```

Make the dataset which stores the values of the format.

```
data other;
  type    = "&ftype";
  hlo     = "O";
  fmtname = "&fname";
run;
```

Create ‘other’ values.

```
data &fname.d;
  set &fname.dpre
      other;
run;
```

Combine both datasets.

```
proc format library=work.formats cntlin=&fname.d;
run;
```

Create the format.

The full macro code is provided in the appendix with the following additions:-

1) Removal of the temporary datasets &FNAME.DPRE and OTHER.
2) Addition of an FDEBUG option to keep the &FNAME.D dataset for debugging purposes.
3) Creating some defaults for commonly used options so that: FTYPE is set to USUBJID, FTYPE is set to C and FDEBUG is set to Y.

The name of this macro is FMTMERGE.

USING THE FMTMERGE MACRO

Now we have the macro set up, it can be applied to our data.

1) USING ALL OF THE DEFAULTS

If we want to create a character format using USUBJID as the value and keep the input dataset for debugging purposes, the syntax to apply the macro is relatively simple:

```
%fmtmerge(fname = myfmt);
```

This example would take a dataset called MYFMT, containing the variables USUBJID and MYFMT, to create a character format called MYFMT, outputting a dataset called MYFMTD for debugging.

2) NOT USING THE DEFAULTS

Of course, the defaults may not be useful for every situation and these can easily be overwritten, for example:

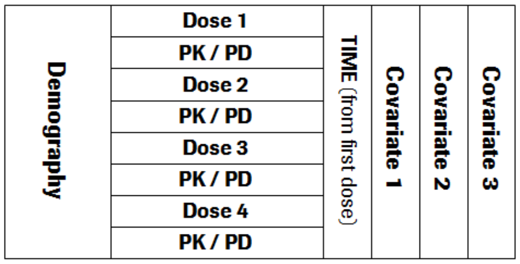
```
    %fmtmerge(fname = weight, fstart = id, ftype = N, fdebug = N);
```

Here we are asking the macro to find a dataset called WEIGHT with the variables WEIGHT and ID to create a numeric format called WEIGHT. The output dataset WEIGHTD would be removed from the work library.

In the above examples only formats are shown but informats could also be created using 'I' or 'J' as the FTYPE option. 'I' would give a numeric informat and 'J' would give a character informat. Providing the rules for creating formats are followed, the macro should behave accordingly.

USING THE FMTMERGE MACRO TO CREATE AN ANALYSIS DATASET

The following illustrates a high level programming strategy for creating a modelling dataset using data from a number of different sources. The FMTMERGE macro will be used in the construction of the dataset. Below shows a graphical representation of the final dataset:

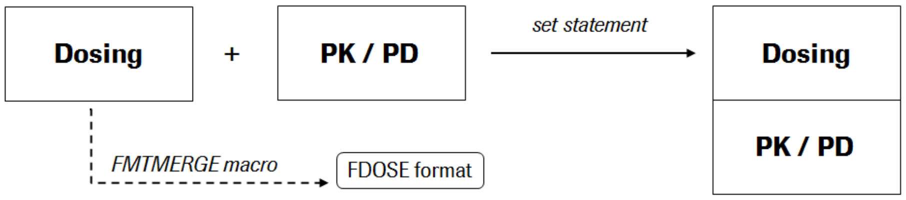


In the above diagram we see the data types used to produce this modelling dataset:-

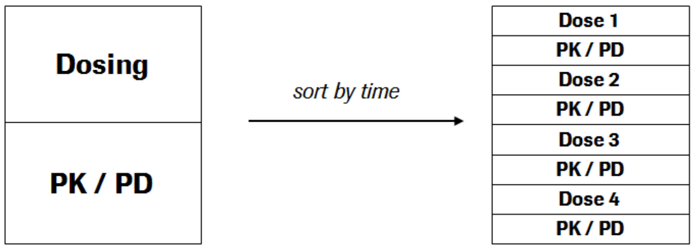
1. **Dosing** – required variables from SDTM.EX
2. **PK/PD** – required variables from pharmacokinetic and/or pharmacodynamic data which may come from any of the following sources: SDTM.PC, SDTM.LB, SDTM.Zx, etc
3. **Demography** – required variables from SDTM.DM
4. **Covariates** – patient level variables derived from various data sources e.g. SDTM.AE, SDTM.LB, SDTM.PC, etc

Step 1 – Dosing and PK/PD

The dosing and PK/PD datasets are subset and processed appropriately before combining using a SET statement. During the processing of EX we can find the date of the first treatment dose, output this as a dataset (called FDOSE here) and use FMTMERGE to create a format:

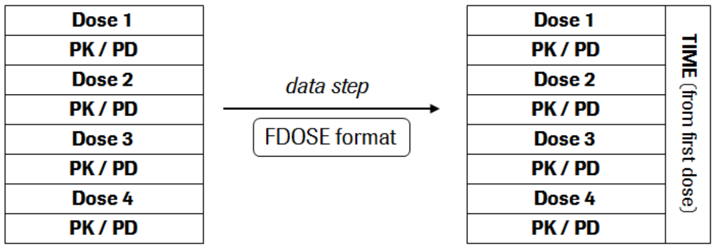


The resulting dataset is then sort by time (e.g. visit date):



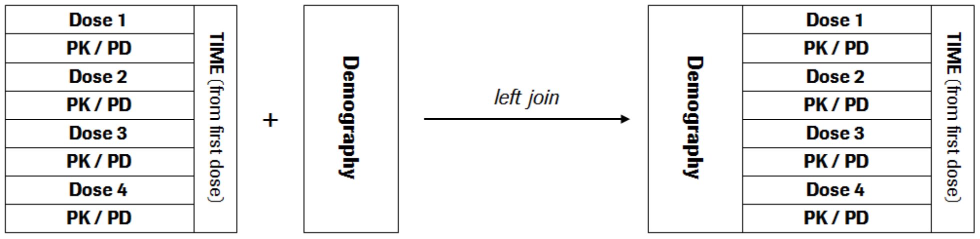
Step 2 – Calculating time from first dose

The FDOSE format can be applied to the combined dosing and PK/PD dataset. Using the visit date and the date of first dose (from the FDOSE format) we can calculate the time from first dose (TIME):



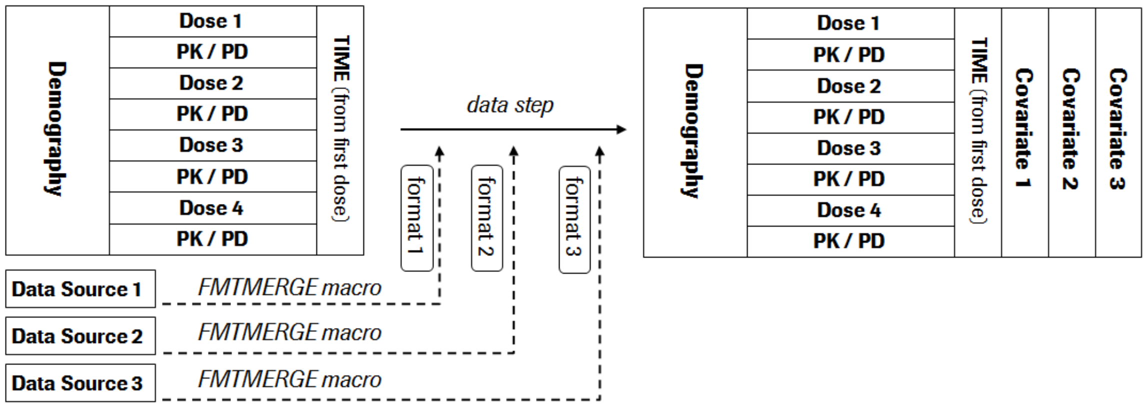
Step 3 – Demographics

The demography dataset is processed as appropriate and added to the dataset created in Step 2. We could take all of the variables demography variables and use FMTMERGE to create formats based upon USUBJID but this would be an inefficient use of the macro. Instead it would be best to use a left join keeping only the demography patients that occur in the dosing and PK/PD dataset:



Step 4 – Add covariates

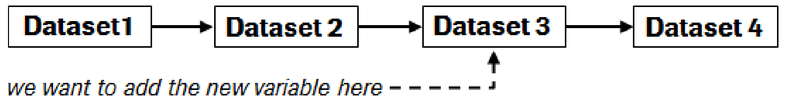
Additional patient level variables can be added by subsetting the required data in separate steps, using the FMTMERGE macro to create a format and then applying it to a DATA step. In the example below, 3 different data sources are used to add in 3 new variables to create our final dataset:



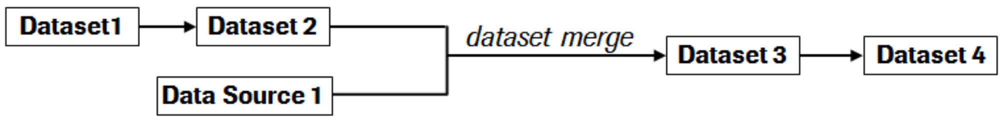
The above is just an example of the types of ways an analysis or modelling dataset can be created using the FMTMERGE macro to reduce the number of dataset merges required. The dataset can be made more complex by including other types of data, different variables for the relationship to time, as well as any number of patient level flags or variables.

USING THE FMTMERGE MACRO TO ADD A VARIABLE TO AN EXISTING ANALYSIS DATASET

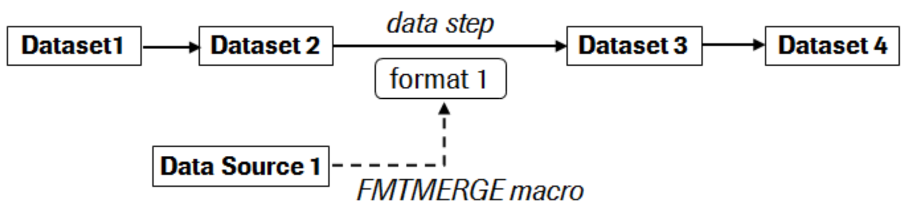
The FMTMERGE macro can also be used to easily add in a new variable to an already existing dataset without disrupting the original data flow. Imagine that you have been asked to add a patient level variable into the code and the most appropriate place is in the middle of a set of steps which are dependent upon each other. The original program flow may be something like this:-



We could just make a new dataset in the middle of the code which has our required new variable. In this example we will add the new variable to Dataset 3 by combining Dataset 2 with Data Source 1.



Using the FMTMERGE macro we can process the data required for the new variable outside of the original program flow, store the new variable in a format and then apply it in an already existing DATA step:-



BENEFITS AND LIMITATIONS

There are a number of benefits to combining data in this way, here are some of them:

- Datasets do not need to be sorted or indexed before merging
- Allows for conditional merging
- The number of observations stays the same e.g. no unexpected increase or decrease of observations
- The syntax for the merge is relatively simple e.g. the syntax for applying a format
- As the number of different data sources required to be merged increases, the complexity of the syntax does not increase substantially

Another benefit the author has found is that by thinking of the input data sources separately, it encourages the code to be structured and sectioned accordingly. This combined with good commenting, in the author's opinion, make the code easier to read, understand and maintain.

Whilst using formats to merge is an efficient way to combine data, it is best suited to one to one merges. For the creation of analysis / modelling datasets this will most often be for patient level data. We could use the same approach for one to many merges but this would require creating new variables containing a combination of the variables (e.g. patient and visit), which could become less efficient and harder to maintain. An alternative approach in this instance could be to use composite indexes, which has its own benefits and limitations.

USES

Here are some uses for the macro:

- Creation of patient level covariates
- Creation of population flags (e.g. Safety, Efficacy, PK)
- Time of first dose of a given treatment (e.g. for creating time variables such as "Time after first dose")
- First planned or actual dose (e.g. for single dose studies where this information is not available in DM)
- Adding N to cohorts / treatment group in tables, listings and figures.

CONCLUSIONS

Data driven formats offer the programmer a clean way to perform one to one merges which can be easily applied in a number of ways to the creation and maintenance of datasets. This approach can be used to reduce dataset merges and allow new variables to easily be added to existing datasets. Use of the FMTMERGE macro simplifies the syntax required to create user-defined data driven formats, making this a very useful additional technique available to the SAS programmer.

REFERENCES

[1] http://support.sas.com/resources/papers/proceedings09/071-2009.pdf
[2] http://www.phusewiki.org/docs/Conference%202013%20CC%20Presentations/CC02.pdf
[3] Adapted from:
http://support.sas.com/documentation/cdl/en/proc/70377/HTML/default/viewer.htm#p0owa4ftikc2ekn1q0rmpulg86cx.htm#p05g1cdvff8h8tn1frtzxrnkzezo

ACKNOWLEDGMENTS

Vincent Buchheit, Johann Laurent, Tim Barnett, Philip Holland and Jim Elder (for the original idea).

CONTACT INFORMATION

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

Paul Grimsey
Roche Products Ltd
6 Falcon Way, Shire Park
Welwyn Garden City / AL7 1TW
+44 (0) 1707 36 5871
paul.grimsey@roche.com

APPENDIX I

```
/******************************************************************************
* fmtmerge.sas - macro to create data driven formats
* -----------------------------------------------------------------------------
* Author   : Paul Grimsey
* SAS version : 9.4
*******************************************************************************
* This macro is designed to help dataset building by creating data driven
* formats to reduce the number of merges / joins in the dataset creation code.
*******************************************************************************
* Keyword parameters:-
*   FNAME              - name of input dataset and variable containing the data
*                        value.
*   FSTART (optional) - variable used to match the data value (FNAME). Default
*                        value is USUBJID.
*   FTYPE  (optional) - Type of format: C - character formaat (default)
*                                        N - numeric format
*                                        I - numeric informat
*                                        J - character informat
*  FDEBUG  (optional) - Y - keeps the output dataset (default)
*                        N - deletes the output dataset
*
* Outputs :-
*   FNAME  - as the name of the format created by the macro
*   FNAMED - output dataset (for debugging purposes)
*
* Limitations
*   The same limitations apply to the usage of this macro as apply to creating
*   formats in SAS e.g.
*     - FNAME should be named in accordance with format naming conventions.
*     - there should be no overalapping values in the format.
*******************************************************************************
* Macro usage examples:-
*
* 1) Default usage:
*      %fmtmerge(fname = RACE);
*    The macro takes a dataset called RACE, with the race variable renamed as
*    RACE.  It then creates a Character format with RACE as the VALUE and
*    USUBJID as the LABEL. A dataset called RACED is also output.
*
* 2) User-defined usage:
*      %fmtmerge(fname = WEIGHT, fstart = PT, ftype = N);
*    The macro takes a dataset called WEIGHT, with the weight variable renamed
*    as WEIGHT.  It then creates a Numeric format with WEIGHT as the VALUE and
*    PT as the LABEL. A dataset called WEIGHTD is also output.
******************************************************************************/

options mprint mlogic;

%macro fmtmerge(fname  = ,
                ftype  = C,
                fstart = usubjid,
                fdebug = Y);

  %* make the dataset which stores the values for the format *;
  data &fname.dpre (keep = fmtname type start label);
    set &fname;
    fmtname = "&fname";
    type = "&ftype";
    start = &fstart;
    rename &fname = label;
  run;

  %* create 'other' values *;
  data other;
    type    = "&ftype";
    hlo     = "O";
    fmtname = "&fname";
  run;

  %* combine both datasets *;
  data &fname.d;
    set &fname.dpre
        other;
  run;

  %* create the format *;
  proc format library=work.formats cntlin=&fname.d;
  run;

  %* remove temporary dataset *;
  proc datasets nodetails nolist;
    delete &fname.dpre other;
  run;
  quit;

  %* remove format dataset - upon user request *;
  %let fdebugu = %upcase(&fdebug);
  %if &fdebugu=N %then %do;
    proc datasets nodetails nolist;
      delete &fname.d;
    run;
      quit;
  %end;

%mend fmtmerge;



********** END OF MACRO **********;
```