PharmaSUG 2019 — Paper AP–214

Minimally Invasive Surgery to Merge Data

Paul Hamilton

ABSTRACT

Many professional papers on the topic of combining data sources have been presented at SAS User conferences over the decades. Most focus on the issue of combining data through the Data Step (set, merge, update) or on joining in SQL. However, there is another, completely different technique using SAS Component Language functions that is optimal for a small range of applications. This paper will present the technique (used in a Data Step and in macro language), explain how it differs from traditional data combination techniques, and offer specific examples illustrating where it might be useful. The technique can also be used to extract either metadata or actual data from SAS data sources. Former developers of SAS/AF will recognize the technique; Base and macro programmers will be amazed!

INTRODUCTION

The need to merge two different SAS® datasets together by a set of key(s) is one of the most common and universal tasks all SAS coders face. There are many techniques for performing this, and myriad papers discussing them have been presented at SAS user conferences over the decades. However, there is one technique developed during the days when SAS/AF® was bright new shiny technology. It is a niche technique useful in relatively small number of applications, but one of them is quite common in clinical trials listings: the concomitant medication listing. It can also be useful in many macro applications, although it can also be used outside a macro.

DIFFERENCES FROM STANDARD MERGING

When merging two or more data sources, using either the Data Step or PROC SQL, each dataset or table used in the merge must be opened, and each record in each input data source must be considered as to whether it should be selected and output as part of the data selection process. Depending on the size of the data sources in question, and the capabilities of the computing environment, this can be trivial or can be an enormous drain on resources. One might assume that if the AE domain is very large (large number of subjects on trial, very ill patient population…) then the CM domain might reasonably be expected to be large as well (patients taking many medications to seek relief from the many AEs). Even though it is possible that no patients took any medications on trial due to a current AE, still both data sources must be processed through each and every record.

Consider the following, very simple Data Step match-merge example:

```
data cm_ae;
  merge
    cm(in = C)
    ae(in = A);
  by usubjid subkey;
  if (C);
run;
```

In the example above, both source datasets must be opened, and each of their rows must be read and examined to see if they meet the conditions of the merge. Utilizing the technique demonstrated below, it is possible that only one dataset will be opened and processed, or (rarely) neither!

MINIMAL TECHNIQUE

Consider the Concomitant Medication CRF on a clinical trial. Depending on the sponsor, this can be quite sparse or could be more voluminous. Since the topic under discussion is included in CDASH V1-1.1 [CM.CMAENO], one hopes that over time all sponsors will be using this field and thus this technique has potentially broad utility.

If following the CDASH standard (or just historic usage), then when the site fills out the CM form, the patient should also be asked whether the CM was taken because of a particular AE. If so, when the entries are entered into the Clinical Trial Data Base, then the AESEQ on the record from the associated AE will be entered as the CMAENO, representing a foreign key. In the Data Step above, this allows a match-merge to bring in a description of the AE (Preferred Term). If the medication in question was not taken associated with an AE, then the field will be left blank. Several interesting situations can then arise:

- It could well be that although there are many CM records (and presumably many AE records as well), that only a small percentage have the CMAENO field populated. In this case, the large (possibly vast) majority of the CM rows need only a Read-Write operation, with no need to do a lookup in the AE source.
- It is probably a rare occurrence, but it is within the realm of possibility that no CM records have a non-missing CMAENO field; whether because the patients truly did not take any medications in response to their adverse events, or because the site personnel did not populate the CTDB properly. In this instance, no effort to combine the CM and AE domains is necessary.

As we read through the CM records, one of two conditions will occur:

1. The CMAENO field is blank, indicating that no AE information needs to be associated with this medication. Simply read and write it. Note that SAS is very swift in performing these operations.
2. The CMAENO field is populated, indicating the need to retrieve the relevant AE information. And what identifies that information? The current USUBJID and the AE Sequence number, which should match the number stored in CMAENO. In this case, open the AE file, pointing to solely the USUBJID and AE# of interest, and retrieve the relevant AE text field (Preferred Term).

CONCEPTS

Since the technique under examination arose from the introduction of SAS SCL (first, *Screen Control Language*, later rebranded as *SAS Component Language*), there are several new concepts to consider:

- File Handle: There is some precedence for this concept in Base SAS: the FILEREF or File Reference. When using a Data Step to read raw (non-SAS) data into SAS, the INFILE statement points to the location of the data source on your file system. Current syntax allows one to simply put the drive, folder location, and file name in quotes (e.g. 'Q:\proj1\rawdata.dat'). However, the original implementation of SAS required the use of a FILENAME statement, which creates a symbolic link (FILEREF) to point to the input data source. The FILEREF in this case is a typical SAS token, 1-8 characters, must start with letter or underscore, etc.
  The concept here is similar except the file handle is not a text string but a numeric variable which will contain an integer.
- Return Code. This occurs quite frequently in ordinary SAS usage, but we are not used to dealing with the codes themselves. If a Data or Proc Step with illegal syntax is submitted, it will fail and the offending code will be clearly pointed out in the SASLOG. This is usually accompanied with a numeric code but most of us will focus on the plain text error message. In SCL however, syntax very often requires that calling a function will also return a numeric code, and SCL can quickly become verbose as one examines and deals with the return code. Of course the outcome of that effort is a more robust and user-friendly set of code.
- Identifying Variables. In Base SAS we are accustomed to identifying variables in datasets by the Name. However, in SCL things are rather more primitive. Instead of referring to variables by Name, one must use their Number (as reported in Proc Contents). If the data source being processed is static in nature (as many CDISC data source0s will tend to be) then this is a slight inconvenience. If the data source is more dynamic in its metadata, then additional effort will be required to identify the correct variable number. Examples of this are out of scope for this paper.

ALGORITHIM

Now that these new concepts have been introduced, the entire technique as used in a Data Step (using in a macro, or macro statements in open code is virtually the same) appears below. To simplify things for

the paper, assume that the AE dataset has been copied to the work library with only USUBJID, AE #, and AEDECOD as fields. In the step below, we will read through the CM dataset, simply reading / writing records until we find a pointer to the AE domain. At this point the special processing begins:

- Build a dynamic <where> clause to use while reading the AE table. This will restrict the processing to the sole AE matching our USUBJID and foreign key. In the example code, this is done in two parts to simplify but this is for illustration only and not required.
- As mentioned above, when performing an explicit OPEN to the AE source, we must accept the file handle returned by the function. This will leave the file open for input, but so far a record has not been read into the Data Step.
- Issue a retrieve record command. Two varieties are available; since the input has been restricted to only retrieve one record, the <FETCH> function is appropriate. This loads the (sole) record where it is available to the Program Data Vector (PDV) but does not transfer data into the Data Step.
- The AE data have been pre-processed to the minimum fields needed for extraction. It is therefore trivial to determine that the desired field [AEDECOD] is the third field. The getVarC function (Get Variable Character) uses both the file handle and the variable # in the file as arguments. The value of [AEDECOD] is returned into the current PDV and the merge process is complete for this record.
- Close the AE data source. As in all things in SAS there must be symmetry: `<> () '' "" data-run proc-run / proc-quit open-close.`
- Data Step processing resumes as normal until the next AE non-missing pointer is found, then the AE data set is opened pointing only to that Subject / AE #. Extract the text string and continue…

MAIN CODE

The code below depends on some setup occurring. The original source CM data have been copied to the WORK library, and a small (3) subset of fields from the AE domain have been copied to WORK as well. When using this technique, sorting either data set is not a consideration.

```
data xtract;
  if (0) then set ae(keep = aedecod);
  set cm;
  length ostr $ 200 wclause $ 60;
  call missing(aedecod);
  if (aeno ne .) then do;
    wclause = 'where = (usubjid eq ' ||
              quote(strip(usubjid))  ||
              ' and aeseq eq '       ||
              put(aeno, 3.)          ||
              '))';
    ostr = 'work.ae(' || strip(wclause) || ')';
    dsid = open(ostr);
    rc = fetch(dsid);
    aedecod = getVarC(dsid, 3);
    rc = close(dsid);
  end;
  drop dsid rc wclause ostr;
run;
```

COMMENTS

The <length> statement allocates storage to hold the dynamic <open> string and the associated <where> clause. It is important to define sufficient room for all required text. By long-standing convention by SAS employees, book and paper authors, the variable containing the file handle is DSID (Data Set IDentification) but this is pure culture and not necessity; any valid SAS token may be used. Similarly, the variable used to capture a Return Code is often [RC] but this is also just convention. However, the

functions called {OPEN FETCH GETVARC and CLOSE} are all built into SAS and must be used as such, just like any other Data Step or Macro function call. The use of mixed case in [getVarC] is also not required; it is designed as a mnemonic device.

EFFICIENCIES

Usage of this technique in situations where a majority of the CM records do in fact point to AEs would be sub-optimal. Note that we open the CM data source once [SET] while we open-read-close the AE source once for each CM-AE pointer. This produces additional overhead which is reasonable and worthwhile for a small number of cases. And of course, if the CM data source has no pointers to AE, then only the CM source is read through, and the AE data source is never opened or read at all.

OTHER USAGE

The general technique of extracting information from a dataset (especially metadata) can be useful in many other situations. For example, inside a macro, one might decide to check a dataset designated as an input source to ensure that it has records to process, and that the metadata supports the intended usage. For example, there is no point in running a Proc Summary against a data source with zero records; or with all character fields. Using these functions, the macro coder can check to ensure that required conditions are met, and if not to gracefully give feedback to the macro caller and exist smoothly.

The old trick to “publish” the number of records in a SAS dataset (views and things like them will not apply) used a Data Step:

```
data _null_;
  if (0) then set old nobs = nobs;
  call symput('NOBS', put(nobs, best.));
  stop;
run;
```

The same effect using SCL would be:

```
%let dsid = %sysfunc(open(old));
%let nobs = %sysfunc(attrN(&dsid, NOBS));
%let rc = %sysfunc(close(&dsid));
```

A large number of SCL functions were ported to the Data Step. However, the SCL functions were never ported to the SAS Macro language; instead the [%sysfunc()] function allows one to reach out and use the vast majority of SCL functions inside a Macro, or as shown above macro statements in open code (parsing data or even metadata from in-built SAS Views can be quite time consuming)

CONCLUSION

In truth, there are not many applications for this particular style of merging. It so happens that in our Industry, an extremely common use case does exist; the Concomitant Medication Listing. Of course, the traditional techniques (Data Step merge, joining in Proc SQL) serve this need perfectly well. It also lends itself to both open-code and within-macro processing when some snippet of either data or metadata needs to be extracted with a minimum of resources.

CONTACT INFORMATION

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

Paul Hamilton
pdhamilton1@comcast.net
(425) 985-3252