PharmaSUG China 2017 - Paper 79

The Encapsulation of Hash Module Macro

Mijun Hu, Novartis, Shanghai

Eason Yang, Novartis, Shanghai

ABSTRACT

The hash object has been introduced a lot in recent year SAS® presentations. Most of them were giving examples and explanations on how to apply them. Somehow people may still struggle on the weird syntax. Statements such as if _n_=1, length and call missing etc. are frequently used to support the hash merge and clear the log issue (uninitialized...). This leads to the motivation of encapsulation the hash merge process. Inputting only three parameters: 1. the dataset you want to merge with; 2. the key variables; 3. the new variables you would like to drag into the datasets. That's all we need to complete a merge without bothering the complex hash syntax.

INTRODUCTION

When we merge large data sets (e.g. developing ADLB data set), we more and more replace the old MERGE statement or PROC SQL with the hash object. Because the hash object provides an efficient, convenient mechanism for quick data storage and retrieval. Despite the hash concept was brought in and even repeated in every year's Pharma SUG, its syntax still looks too odd be memorized. A common way of using it is to store the template code of hash merge, and edit some key parameters after pasting the template into the program. We may add some extra statements like CALL MISSING just for the sake of avoiding unwanted messages in the log such as 'NOTE: Variable xx is uninitialized.'. This is sometimes upsetting for SAS® programmers not familiar with hash syntax. The encapsulation of the hash merge process into a macro can relieve people from this pain.

THE PARAMETERS IN THE MACRO

The macro is in the form of %merge(hash, ds, key, data, wh), i.e. 5 parameters. It looks long but pretty self-explanatory.

1. Hash: refers the hash name you would like to give in the hash object, it follows the same rule as SAS® variable name. Different hash names should be provided if multiple hash objects are created in the same DATA step.
2. Ds: refers the data set name you would like to merge. The larger it is, the more efficiency hash brings to you.
3. Key: refers to the key variables identified to merge the data set. It also supports renaming feature which will be introduced in the later sections.
4. Data: refers to the additional variables you would like to drag from the merging data set to the main data set. It also supports renaming feature as Key.
5. Wh: is the optional condition which you want to apply before the merge.

Note: if any symbol such as equal sign(=) is included in the parameter, it is required to put the whole parameter text in the quotation function %str(), otherwise the parameters can be corrected parsed by SAS®.

One practical example is given below:

```
data dlb;
    set data_a.dlb;
    %merge(h,data_a.dlb, lbtestcd usubjid, %str(lbstresn=base lbstresc=basec),
           %str(lbblfl='Y'));
run;
```

We can assign the baseline values of LBSTRESN/LBSTRESC across the visits for the same subject, and rename them as BASE/BASEC. Therefore we give a random hash name 'h', data set name DATA_A.DLB, use LBTESTCD and USUBJID as the key to merge. We also need to apply the filter LBBLFL='Y' in order to select only the baseline records. To complement the description of renaming feature, if you want to rename any variables before putting into the hash, you can do it inside the parameter texts. This functionality is enabled both in Key and Data parameters.

WHAT HAPPENS IN THE MACRO

If we use OPTIONS MPRINT to observe how SAS® parses the macro, we can see the following code in the log:

```
        if 0 then set data_a.dlb(keep=lbstresn lbstresc
              rename=(lbstresn=base lbstresc=basec));
        if _n_ =1 then do;
        declare hash h(dataset: "data_a.dlb(where=(lbblfl='Y')
              rename=( lbstresn=base lbstresc=basec))" );
        h.defineKey("lbtestcd", "usubjid");
        h.defineData("base", "basec");
        h.defineDone();
        end;
        rc=h.find();
        drop rc;
```

The algorithm of the macro is obvious:

1. Set a fake condition first to initialize the new variables we would like to merge in, i.e. LBSTRESN and LBSTRESC, renamed to BASE and BASEC respectively. An old way of doing this is to use the LENGTH statement along with CALL MISSING routine to avoid the messages like 'NOTE: Variable xx is uninitialized.', but you need set a length long enough to cover the longest merging values. While the fake condition way is more dynamic, it ensures the lengths of the new variables are accurately conveyed to the main data set, in addition, no need to add the CALL MISSING routine.
2. Commence the hash steps: a) Declare the hash object. b) Create an instance of (instantiate) the hash object. c) Initialize lookup keys and data. The renaming process again occurs within the instantiation step.
3. If there are multiple calls of the macro within the same DATA STEP, please pick a different hash name in the first parameter.

CONCLUSION

Hash has be more and more used instead of MERGE statement and SQL procedure. But the bizarre structure puzzles a lot of new users. With the capsulation of the hash module, we can realize multiple large merge within one single DATA step only by providing some key parameters. Getting rid of knowing re-constructing the hash module for each merge.

CONTACT INFORMATION

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

Name: Mijun Hu
Enterprise: Novartis
Address: 42178 Jinke Road
City, State ZIP: Shanghai
E-mail: mijun.hu@novartis.com

Macro Full Code

```
   %macro merge(hash,ds,key,data,wh);
        %local  i temp key_right key_renam data_left data_right data_renam ;
        %if %length(&wh)=0 %then %let wh=1;
        %let i=1;
        %do %while(%length(%scan(&key,&i)));
              %let temp=%scan(&key,&i);
              %if &i=1 %then %let key_right=%sysfunc(quote(%scan(&temp,-1,=)));
              %else %let key_right=&key_right, %sysfunc(quote(%scan(&temp,-1,=)));
              %if %index(&temp,=) %then %let key_renam=&key_renam &temp;
              %let i=%eval(&i+1);
        %end;
        %put &=key_right;
        %put &=key_renam;
        %let i=1;
```

```
    %do %while(%length(%scan(&data,&i)));
           %let temp=%scan(&data,&i);
           %if &i=1 %then %let data_right=%sysfunc(quote(%scan(&temp,-1,=)));
           %else %let data_right=&data_right, %sysfunc(quote(%scan(&temp,-1,=)));
           %if %index(&temp,=) %then %let data_renam=&data_renam &temp;
           %let data_left=&data_left %scan(&temp,1,=);
           %let i=%eval(&i+1);

    %end;
    %put &=data_left;
    %put &=data_right;
    %put &=data_renam;
    if 0 then set &ds(keep=&data_left %if %length(&data_renam) %then
rename=(&data_renam););
    if _n_ =1 then do;
    declare hash &hash(dataset: %if %length(&key_renam) or %length(&data_renam)
%then "&ds(where=(&wh) rename=(&key_renam &data_renam))";
                                                %else "&ds(where=(&wh))";
                              );
           &hash..defineKey(&key_right);
           &hash..defineData(&data_right);
           &hash..defineDone();
    end;
    rc=&hash..find();
    drop rc;
%mend merge;
```