PharmaSUG 2013 - Paper BB06

Doing the hashwork: Using the DATA step hash object to perform common clinical programming chores.

Miles Dunn, Alexion Pharmaceuticals Inc., Cheshire CT

ABSTRACT

The hash table is a widely used data structure in object oriented programming languages. The addition of the hash object to the base SAS software creates a small space of overlap between object oriented programming concepts and the procedural programming concepts used in SAS. The purpose of this paper is to give a taste of how object oriented programming concepts can be applied to common clinical programming tasks.

INTRODUCTION

A hash object or associative array is a data structure that maps a key variable or variables to a bucket of associated data values. A SAS format or informat is an example of a simple hash table that maps a single variable to a single associated data value. Although conceptually similar to a format or array, the hash object is more powerful and flexible. The hash object extends the functionality of SAS formats and informats by allowing both the hash key and the associated data bucket to consist of multiple component parts. With SAS version 9 the hash object and the associated hash iterator object were added to the functionality of base SAS. This paper will show how to use the SAS hash object to merge together subject level data without using sort/merge steps or sql joins. I'll show how to define a hash object and how to load the hash table in a data step. I'll show how to use object.method syntax to invoke the find, check, add, replace, first, last, next, prev, and output methods against the hash object. I'll show how to use a hash iterator object to process the records in a hash object with multiple records for each value of the hash key. I'll bring these concepts together to show how a hash programming approach can be used to create a simple time to event analysis for hematologic normalization. Time to event analysis is an increasingly widely used analytical technique in the pharmaceutical industry. Typically the analytical datasets used for this type of analysis are created through data step programming or with proc sql. Hash programming provides an intriguing and seldom used alternative.

I'll conclude the paper with a discussion of the key differences between hash objects and the other types of data structures and lookup tables that perform similar programming functions. Hashing is a powerful and flexible addition to the programmer's armamentarium. The flexibility of hash programming techniques can simplify analytical programming by reducing the number of data step merges or sql joins needed to bring together the disparate data sources needed to conduct a time to event analysis.

HASH OBJECT DEFINITION

The hash object must be defined in a data step. The hash object exists for the duration of the data step in which it is created and is deleted from memory when the data step ends. The following code shows the syntax to define a hash object.

```
DATA _null_;

       LENGTH usubjid $11 age height weight 8;

       IF _n_=1 THEN DO;

              DECLARE HASH hadsl(DATASET:'adsl', ORDERED:'ascending');

              hadsl.DEFINEKEY('usubjid');

              hadsl.DEFINEDATA('usubjid','age','height','weight');

              hadsl.DEFINEDONE();
```

```
            CALL MISSING(usubjid,age,height,weight);

      END;

      [set datasetname];

RUN;
```

A length or attrib statement is needed to load the variables identified as hash key and data values into the program data vector (PDV). When the hash object is defined the SAS compiler does not automatically establish a link between the hash key and data values and the dataset variables that are loaded into them. That link has to be established explicitly using a length or attrib statement. The hash object is declared and loaded with the declare hash statement. In this case a hash object called hadsl is created and loaded with the contents of the dataset adsl. The ordered tag orders the data values in the hash object in ascending order of the hash key defined in the definekey statement. The definedata statement defines the data values of the hash object. The definedone statement ends the hash definition. The call missing statement initializes the hash data values to missing. Without the call missing statement SAS will generate a note in the log indicating that the hash data values are uninitialized. If a set statement is used the call missing statement must be placed before the set statement. Otherwise the values of the dataset read in with the set statement are overwritten with missing values.

HASH TABLE LOOKUPS AND NAVIGATION

The two methods for retrieving data values from a hash object are find and check. The find and check methods will both determine whether a key value exists in the hash object. If the hash object is populated with data values, the find method will write those values to the corresponding dataset variables. The code below shows how to merge two datasets using a hash object. The hash declaration defines a hash object hadsl and populates it with the data in adsl using the dataset tag. The find method is used to retrieve the data values from the hash object and merge them with the variables read from adlb in the set statement.

```
DATA adlb1;

      LENGTH usubjid $11 height age weight 8;

      IF _n_=1 THEN DO;

            DECLARE HASH hadsl(DATASET:'adsl');

            hadsl.DEFINEKEY('usubjid');

            hadsl.DEFINEDATA('usubjid','height','age','weight');

            hadsl.DEFINEDONE();

            CALL MISSING(usubjid,height,age,weight);

      END;

      SET adlb;

      rc=hadsl.FIND();

RUN;
```

This code is functionally equivalent to merging adlb with adsl using a sql join or a datastep merge. The hash approach has a slight advantage over the datastep approach in that it won't require any preliminary sorting. The hash approach is a direct lookup based on the hash key.

The hash iterator object provides a way to traverse a hash object. The hash iterator object is defined using a declare hiter statement that associates the hash iterator object with the name of the hash object you want to traverse.

```
DECLARE HITER hiplat('hplat');
```

This code declares a hash iterator object called hiplat and associates it with the hash object hplat. The newly created hash iterator object can then be used to traverse the hplat hash object using the first, last, next, and prev methods.

The following code loads the platelet count data from adlb into a hash object called hplat. A hash iterator object called hiplat is used to process the records in hplat to identify platelet counts greater than or equal to 150 and output them to a dataset called crit1.

```
DATA crit1;

      LENGTH usubjid $11 ady aval 8;

      IF _n_=1 THEN DO;

            DECLARE HASH hplat(DATASET:"adlb(where=(paramcd='PLAT')
",ORDERED:'ascending');

            DECLARE HITER hiplat('hplat');

            hplat.DEFINEKEY('usubjid','ady');

            hplat.DEFINEDATA('usubjid','ady','aval');

            hplat.DEFINEDONE();

            CALL MISSING(usubjid,ady,aval);

      END;

      rc=hiplat.FIRST();

      DO WHILE(rc=0);

            rc1=hplat.FIND();

            if aval>=150 then OUTPUT;

            rc=hiplat.NEXT();

      END;

RUN;
```

The first method on the hash iterator hiplat moves to the first value in hash object hplat. The do loop iterates through the values in hplat and outputs the values with aval>=150 to dataset crit1.

HASH ANALYSIS

If we want to identify the list of distinct subjects who meet the criterion for platelet count normalization that can be done by creating a second hash object with usubjid defined as the hash key. The following code creates a second hash object called platnorm with usubjid as the hash key. The add method is used to populate the platnorm hash object with the first value from the hplat hash object that satisfies the criterion for platelet count normalization. The output method is used to create an output dataset called crit1 from the hash object platnorm.

```
DATA _null_;

       LENGTH usubjid $11 ady aval 8;

       IF _n_=1 THEN DO;

              DECLARE HASH
hplat(DATASET:"adlb(where=(paramcd='PLAT'),ORDERED:'ascending');

              DECLARE HITER hiplat('hplat');

              hplat.DEFINEKEY('usubjid','ady');

              hplat.DEFINEDATA('usubjid','ady','aval');

              hplat.DEFINEDONE();

              DECLARE HASH platnorm(ORDERED:'ascending');

              platnorm.DEFINEKEY('usubjid');

              platnorm.DEFINEDATA('usubjid','ady','aval');

              platnorm.DEFINEDONE();

              CALL MISSING(usubjid,ady,aval);

       END;

       rc=hiplat.FIRST();

       DO WHILE (rc=0);

              if aval>=150 then rc1=platnorm.ADD();

              rc=hiplat.NEXT();

       END;

       rc1=platnorm.OUTPUT(DATASET:'crit1');

RUN;
```

This code takes advantage of the fact that by default the hash object will store one set of data items per key value. If you want to keep the last value instead of the first you would use the replace method instead of the add method. If you want to store multiple values in the hash object for each distinct value of the hash key that can be done through the multidata tag on the hash declaration. The default setting for the multidata tag is no.

The default behavior of the multidata tag can be used to write hash code that will identify the distinct values of a variable. This is functionally equivalent to using a proc freq or the distinct keyword in proc sql.

The preceding example showed how to use a hash object to identify subjects who meet the criterion for normalization based on a single lab parameter. Hematologic normalization is a compound time to event parameter that requires the assessment of platelet counts and lactate dehydrogenase (ldh). Hematologic normalization is considered to occur when the subject achieves platelet count normalization and ldh normalization simultaneously. Platelet count normalization is defined as a platelet count greater than or equal to 150. LDH normalization is defined as an LDH value less than or equal to the upper limit of normal (anrhi). The following example will show how to assess each individual component of a compound efficacy parameter using a separate hash object for each component

parameter. The values of the component hash objects are read into the program data vector and merged with the adsl data. A censor variable is created based on the values from the component hash objects. The combined data is stored in a third hash object along with the derived analysis variables. The contents of the combined hash object are written to an output dataset called hemanorm that can be used for time to event analysis.

```
DATA _null_;

       LENGTH usubjid $11 height weight platdy plat ldhdy ldh censordy 8 ;

       IF _n_=1 THEN DO;

              DECLARE HASH hplat(DATASET:"adlb(where=(paramcd='PLAT')
rename=(ady=platdy aval=plat))",ORDERED:'ascending');

              DECLARE HITER hiplat('hplat');

              hplat.DEFINEKEY('usubjid','platdy');

              hplat.DEFINEDATA('platdy','plat');

              hplat.DEFINEDONE();


              DECLARE HASH hldh(DATASET:"adlb(where=(paramcd='LDH') rename=(ady=ldhdy

                                  aval=ldh anrhin=ldhhin))",ORDERED:'ascending');

              DECLARE HITER hildh('hldh');

              hldh.DEFINEKEY('usubjid','ldhdy');

              hldh.DEFINEDATA('ldhdy','ldh','ldhhin');

              hldh.DEFINEDONE();

              DECLARE HASH platnorm(ORDERED:'ascending');

              platnorm.DEFINEKEY('usubjid');

              platnorm.DEFINEDATA('usubjid','platdy','plat');

              platnorm.DEFINEDONE();


              DECLARE HASH ldhnorm(ORDERED:'ascending');

              ldhnorm.DEFINEKEY('usubjid');

              ldhnorm.DEFINEDATA('usubjid','ldhdy','ldh','ldhhin');

              ldhnorm.DEFINEDONE();


              DECLARE HASH hemanorm(ORDERED:'ascending');

              hemanorm.DEFINEKEY('usubjid');
```

```
        hemanorm.DEFINEDATA('usubjid','height','weight','platdy','plat',
        'ldhdy','ldh','censor','aval');
        hemanorm.DEFINEDONE();
        CALL MISSING(usubjid,height,weight,sex,platdy,plat,ldhdy,ldh,ldhhin);
END;
rc=hiplat.FIRST();
DO WHILE (rc=0);
        IF plat>=150 THEN rc1=platnorm.ADD();
        rc=hiplat.NEXT();
END;
rc=hildh.FIRST();
DO WHILE (rc=0);
        IF ldh<=ldhhin THEN rc1=ldhnorm.ADD();
        rc=hildh.NEXT();
END;
SET adsl END=eof;
rc1=platnorm.FIND();
rc2=ldhnorm.FIND();
rchema=(rc1=0) + (rc2=0);
IF rchema=2 THEN DO;
        censor=0;
        aval=MAX(platdy,ldhdy);
END;
ELSE DO;
        censor=1;
        aval=censordy;
END;
rc3=hemanorm.ADD();
IF eof THEN DO;
```

```
            rc=platnorm.OUTPUT(DATASET:'platnorm');

            rc=ldhnorm.OUTPUT(DATASET:'ldhnorm');

            rc=hemanorm.OUTPUT(DATASET:'hemanorm');

      END;

RUN;
```

This code defines five hash objects called hplat, hldh, platnorm, ldhnorm, and hemanorm. Hplat and hldh have associated hash iterator objects called hiplat and hildh. The hplat hash object is loaded with hash values from the dataset adlb. The where clause is used to select records from adlb with paramcd=PLAT. The rename clause is used to rename ady and aval as platdy and plat. The hldh hash object is loaded with hash values from adlb where paramcd=LDH. The rename clause is used to rename ady, aval, and anrhin as ldhdy, ldh, and ldhhin. Anrhin is needed for the LDH parameter because LDH normalization is assessed relative to the normal range. The key values for the hplat and hldh hash objects are the subject id and the lab assessment day. These hash objects are structured like the lab data with one record per subject per assessment day. Three additional subject level hash objects are defined with usubjid as the hash key. The platnorm hash object stores the day of the first occurrence of platelet count normalization for each subject. The ldhnorm hash object stores the day of the first occurrence of ldh normalization. The hemanorm hash object stores the contents of both component hash objects along with the derived analysis variables aval and censor. The three subject level hash objects do not have associated hash iterator objects defined on them.

The hiplat hash iterator object is used to traverse the hplat hash object to find the first hash value with plat>=150. The first value of plat/platdy that satisfies the platelet normalization criterion is loaded into the platnorm hash object using the add method. The hildh hash iterator traverses the hldh hash object. The first value of ldh/ldhdy that satisfies the ldh normalization criterion is loaded into the ldhnorm hash object. The hplat and hldh hash objects will contain as many hash values as there are subjects who meet the respective normalization criteria. In this respect a hash object is a dynamic data structure. Unlike a static array a hash object does not have a predefined number of elements.

The set statement reads in the adsl dataset and the find method is used to merge the hash values from platnorm and ldhnorm with the adsl data. If the subject exists in both hash objects the subject is considered to meet the criteria for hematologic normalization and the censor variable is set to 0. Otherwise the censor variable is set to 1. If the subject has the event (censor=0) aval is set to the maximum of platdy and ldhdy. This value is the earliest day at which both component normalization criteria are simultaneously satisfied. If the subject is censored (censor=1) aval is set to the value of censordy from the adsl dataset. Typically this value would be the study withdrawal date or the date of last contact. The values of usubjid, height, weight, plat, platdy, ldh, ldhdy, aval, and censor are written out to the hemanorm hash object. The output method is used to create platnorm and ldhnorm datasets that contain the set of subjects who satisfy each of the component normalization criteria. A hemanorm dataset is created that contains the contents of both component hash objects along with the derived analysis variables.

This is a simplified example of a time to event analysis based on a hash programming approach. Real world time to event analyses would involve more complicated normalization algorithms but the logic shown in this example is sufficient to illustrate the technique. In the remainder of this paper I'll discuss the differences between hash objects and the other types of data objects and lookup tables that are available in base SAS. I'll try to provide some motivation for when and why a hash based programming approach might be advantageous relative to the available alternatives.

HASHING VS THE ALTERNATIVES

In cases when a hash object is used as a simple lookup table it is functionally similar to a SAS format or informat. But the hash object is more flexible and has several advantages over formats and informats. Formats and informats are type restricted. A character format converts a character type variable to another character type variable. A numeric format converts a numeric variable to a character variable. An informat does the reverse. But a hash object

is highly flexible in terms of variable type. The hash key and hash data values can consist of any combination of character and numeric variables. In addition the hash key and data values can consist of any number of elements. By contrast a SAS format or informat is always a one-to-one lookup table. In this respect a format or informat is analogous to a highly simplified hash table where both the hash key and hash data values are restricted to a single element.

An array has some of the functionality of a hash object but with several important differences. One difference is that arrays in SAS are constrained to be of a single type. An array can consist of either character or numeric elements but not both. Hash objects are not type constrained. Another difference is the manner in which hash objects are traversed through the hash iterator object. Array elements can be accessed either explicitly through their numeric index values or implicitly through a do over statement. The first, last, next, and prev methods allow greater flexibility of movement through the hash data values than is possible with an explicitly indexed array. Hash key values can consist of several variables, not all of which need be of the same type. Hash key values can be assigned programmatically during data step programming by combining elements of other data step variables or hash values. Functionally this is equivalent to deriving a merge variable in a data step and then using the newly derived variable in a subsequent data step merge or sql join.

Hash objects are memory resident. Hash table lookups occur more quickly than equivalent code based on formats because the hash table lookup doesn't require hard disk access. Since they are memory resident hash objects must be small enough to fit into the available memory. This is a potential disadvantage of loading large datasets into a hash object.

CONCLUSION

In the context of time to event analysis the flexibility of hash objects is useful because it allows different lab parameters to be stored in different hash objects and accessed programmatically within a single data step. The same analysis written with data step merges or sql joins would require a greater number of programming steps. The hash object effectively substitutes complexity in the data structures for analytical programming complexity. The result is a more concise and elegant program that is less prone to programming errors.

In the pharmaceutical industry analytical accuracy and code transparency are more important considerations than program efficiency. The reason is that the pharmaceutical industry is a regulated industry in which analytical programming can be audited by regulatory authorities. Analytical errors have the potential to seriously misrepresent the risk/benefit profile of the study drug. That misinformation would lead to poor decision-making on the part of both producers and consumers of pharmaceutical products. Code transparency is almost as important a consideration as accuracy because it affects the sponsor's ability to explain and defend analytical results to the regulatory authorities. Code transparency affects the maintainability of the code, which can become important for programs that are copied to many studies or which persist for many years. Efficiency is a relatively unimportant consideration in the pharmaceutical industry because clinical trials datasets are generally small, typically not more than a few thousand records. Given the speed of modern computer processors, datasets consisting of a few thousand records can be processed virtually instantaneously. Even highly inefficient programming logic would not significantly increase the processing time. By contrast in the financial and telecommunications industries datasets consisting of hundreds of millions of records are common. In those industries program efficiency is a more serious consideration.

Hash programming can improve analytical accuracy and code transparency by reducing the number of processing steps required to carry out an efficacy analysis. The overall complexity of the analytical programming is reduced through the use of more flexible data structures. Fewer processing steps means there are fewer opportunities for analytical errors. Keeping up with the hashwork can help to keep the analytical house clean!

REFERENCES

Burlew, Michele M. 2012. SAS Hash Object Programming Made Easy. Cary, NC: SAS Institute Inc.

Secosky, Jason and Bloom, Janice 2007. "Getting Started with the DATA Step Hash Object". Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

Miles Dunn

Principal Statistical Programmer

Alexion Pharmaceuticals, Inc.

352 Knotter Drive

Cheshire, CT 06410

USA

203-250-6385

dunnm@alxn.com