PharmaSUG 2018 - Paper BB-21

Creating a Data Shell or an Initial Data Set with the DS2 Procedure

Matthew Wiedel, Syneos™ Health

ABSTRACT

When one creates a standardized database, each dataset has a specific title, structure, and collection of variables. They are most likely created from one or more raw datasets. Variables within each dataset will most likely have specific lengths, formats, and labels. As studies become more complex, the requirements placed upon the information become more nuanced and complicated. So far, the data step and the SQL procedure are quite sufficient. However, there may come a time when a different approach to data creation will be needed.

The DS2 procedure is a completely different way to work with a SAS dataset than that of the good ol' data step. Object oriented programming is a different way of thinking, with its threads, methods, packages, declaration statements… It also provides an extended array of formatting options. Proc DS2's declare statement is akin to the data step's attribute statement as it assigns formats and labels to variables. This is an exercise to demonstrate a way to grab information from a spreadsheet, place it into a dataset using two different avenues and compare the two as they relate to the platform used and the resulting data shell.

INTRODUCTION

Eventually the entire world will become a giant data lake. SAS programmers will need to learn to swim and no better way to dip one's toe into the murky water than with the DS2 procedure. It's a data step that looks like a procedure which uses threads, packages and methods (see OOP). Along with other procedures such as FEDSQL, the relatively new SAS data step, Proc DS2, is here to address the need to voyage into the cloud and handle future big data projects.

Up ahead is a suggested model using OOP to create a dataset. Technically, it would be considered a thread rather than a program. It has 4 parts: importing specifications and macro variable creation, thread setup (initial data input), package authoring (variable definition), and data creation (thread and package declaration).

Databases typically have specifications which tell the programmers and data users how the data sets were created. Variable names, formats, lengths, labels, instructions, and general information can usually be found in these documents. If placed in an excel file, a Proc Import can grab variable specific information per dataset and thus used to create a data shell or an initial dataset with all the correct formatting. One could use a data step that implements an attribute function to create a data shell/ initial dataset or accomplish the same thing with the DS2 procedure using its interesting techniques.

BRIEF DS2 PROCEDURE DESCRIPTIONS

Declare statements are to Proc DS2 as attribute statements are to data steps. Both assign formats, lengths, and labels to a variable. However, the data step has only two data types, character and numeric, while the DS2 data procedure allows for a myriad of various data types such as binary, varchar, smallint…. The capability to work across relational databases is a major reason for DS2's greater versatility.

Methods are like macros, a small, reusable collection of code that can receive parameter values. The hermeneutics of methods are a bit different than those of a macro, however. It's considered an object which connects and works in concert with other methods to build the data. In this example, the methods will return a values.

A package is a collection of methods and can be used universally among all other programs if output into a package library. Just as with the use of a program that contains a catelog of macro programs, the package can be used to have various methods called upon to perform particular jobs.

One either writes a program or a thread. Threads are used as parallel programs and create efficiencies when querying data. It can process multiple rows of data concurrently and then pass the resulting data to the main data program. When the data sets are large and the need to process it at a reasonable clip, these threads come in handy.

DATA SPECIFICATION AND MACRO VARIABLES

This exercise is really a suggested framework of dataset creation using only DS2, SQL, and embedded FEDSQL code. As a move from data step use, the hope is to implement the advantages afforded the data scientist by the DS2 procedure, in particular, the reusability of packages which can create permanent objects for all standardized databases. Threaded programs would also create better allocations of memory and make programs run faster.

Initially, the specification should have formatting and labelling information for each variable in a particular dataset. This information can then be imported and put into a macro variables using SQL, and in turn, employed for the creation of either a data shell or an initial dataset. The intention is that all the preliminary metadata can rest in one place, inside the dataset requirement, and only be updated there for the entire database.

Below would be an example of the meat of an ADSL specification to be used for our purposes of dataset creation:

DOMAIN	VARIABLE	LABEL	TYPE	DEFINITION	FORMAT
ADSL	STUDYID	Study Identifier	CHAR(12)	DM.STUDYID	$12.
ADSL	USUBJID	Unique Subject Identifier	CHAR(22)	DM.USUBJID	$22.
ADSL	SUBJID	Subject Identifier for the Study	CHAR(6)	DM.SUBJID	$6.
ADSL	SITEID	Study Site Identifier	CHAR(4)	DM.SITEID	$4.
ADSL	AGE	Age	DOUBLE	DM.AGE	8.
ADSL	AGEU	Age Units	CHAR(8)	DM.AGEU	$8.
ADSL	SEX	Sex	CHAR(6)	DM.SEX	$6.
ADSL	RACE	Race	CHAR(100)	DM.RACE	$100.
ADSL	ARM	Description of Planned Arm	CHAR(200)	DM.RACE	$200.
ADSL	TRT01P	Planned Treatment for Period 1	CHAR(4)	Dependent on DM.ARM	$4.

These are the required variables in an ADSL dataset. A SAS dataset is created using the import procedure:

```
%macro begin(dset);
PROC IMPORT OUT= work.&dset
DATAFILE= "C:ADAM_REQUIREMENT.xlsx" DBMS=xlsx REPLACE;
  SHEET="&dset";
  GETNAMES=YES;
RUN;
```

For this program, the TYPE column is a DS2 specific value. It really is a type (length) and is placed as the first item in a declare statement. Although this example implements two types: Character

and Double, there are a many others including integer, binary, float, and so on. This is one of the things that sets it apart from the data step which only considers two data types: numeric and character.

At this point, a small addition is made to the imported data set and macro variables created. A DS2 procedure can be used to create a format variable (formt):

```
proc ds2;
data &dset._dclr (overwrite=YES);
/*overwrite=YES is necessary to replace the older version*/
declare char(15) formt;
method run();
set {select *,
       case
        when format = ' ' then ' '
        /*format needs to be in single quotes to compile*/                    else 'format
        '||trim(left(format))
       end as formt
    from &dset where variable ^= ' '};** functions ne and ~= do not work here;
end;
enddata;
run;
quit;/* Resulting dataset will not show up in SAS Enterprise Guide Output Data Tab*/
```

There are 4 system methods: init() - Initialization, run() - Running, term()-Termination and setparms() – Set Parameters. Here, the run method is used to grab the imported data using an embedded FEDSQL query. Everything is then wrapped up with an enddata, run, and quit statement. The re-running of a regular data step will replace a previous work data set automatically. However, for the DS2 procedure, it is necessary to include the 'overwrite=YES' option in the data statement in order to replace previous output dataset. The other annotation describes different 'unexpected' results or conditions one finds when running the DS2 procedure.

From this new dataset, an SQL procedure can be implemented to count the number of variables read from the specification; hence, create macro variables for the variable types, names, labels, and formats :

```
proc sql;
select count(*) into : cnt from &dset._dclr;
quit;

%let cnt=&cnt;
proc sql;
select variable into :var1-:var&cnt from &dset._dclr;
select label into :lab1-:lab&cnt from &dset._dclr;
select formt into :format1-:format&cnt from &dset._dclr;
select type into :type1-:type&cnt from &dset._dclr;
quit;

%do j=1 %to &cnt;
%let var&j=&&var&j;
%let lab&j=&&lab&j;
%let format&j=&&format&j;
%let type&j=&&type&j;
%end;
```

DATA PROCEDURE OUTPUT MODEL

DATA SHELL CREATION

If a data shell is the goal, then all one would need is the variable declaration portion of the DS2 data procedure:

```
proc ds2;
data &dset._start2(overwrite=YES);
```

```
%do i=1 %to &cnt;
  dcl &&type&i &&var&i having label %tslit(&&lab&i) &&format&i;
%end;

enddata;
run;
quit;
```

A same result can be achieved using a data step with an attribute function:

```
data adsl start3;
%do i=1 %to &cnt;
  attrib &&var&i label = "&&lab&i" length= &&fmt&i;
%end;
run;
```

There are a few more things a programmer will need to keep in mind when creating a data set using the DS2 procedure. When declaring a variable, it is only necessary to give the type (char(10), integer, float…) and variable name (USUBJID, COUNTRY,…). Thus, label and formatting assignments are extras. One may notice in the declare %do loop, the label uses a system macro function %tslit. This is needed since the double quotations " " are handled differently within this procedure. Whatever is within " " is considered as another variable (i.e. trt02p="TRT2" would be interpreted as assigning trt02p the value of the TRT2 variable). %tslit is like double quotations in a data step setup (also note the label for usubjid_dm is in single quotes, otherwise it would error.)

The empty shell can be used to carry the prescribed labels and formats when the final dataset is output at the end of a data creation program. It can also be the starting place and populated along the way. To create this initial data set, further work can be done using various aspects of the DS2 procedure.

THREAD USE

With creative FEDSQL merging and queries, the data analyst is able to create an initial dataset that gets close to the prescribed final product. In this given setup, the dataset created will be specific to the ADSL. Other datasets will use a different merge. Although not necessary with this simple project, a thread will be used to merge the DM and DS datasets as an another example of implementing a complex imbedded FEDSQL query:

```
proc ds2;
thread dmds / overwrite=yes;
  method run();
    set {select a.dsdecod,b.maxvis, c.*
         from sdtm.ds a
         join (select max(visitnum) as maxvis, usubjid from sdtm.ds group by usubjid) b
         on a.usubjid=b.usubjid and a.visitnum=b.maxvis and   a.dscat='DISPOSITION EVENT'
         right join libdev.dm c on a.usubjid=c.usubjid};
  end;
endthread;
run;
quit;
```

The set statement picks the last disposition event in the DS dataset and merges it with the DM dataset. Unlike the SQL procedure, the FEDSQL code uses the ANSI standard. Code that works with an SQL procedure may not work using FEDSQL, thus the SAS programmer who wants to fully implement the DS2 procedure should also get familiarized with FEDSQL.

PACKAGES TIME

Macro caches are very handy. The institutional knowledge within can be used over and over again by the SAS programmer with a simple macro call. Often, a diversity of programmers contribute to these sets of macros, thus exhibiting many interesting and useful coding techniques. Sometimes, these macros sit permanently and are universally used. Other caches go project to project and are modified based on the specific needs of the work at hand. DS2 packages can be used in a similar matter.

Packages are a collection of methods. These methods can be invoked similarly as a macro to perform a task and to often return a value. In this example, to show the usefulness of instituting a project specific package, three variables will have their own method assigned.

```
proc ds2;
package plib.bedrock / overwrite = yes;
/*signature 1*/
method usubjid(char(22) usubjid) returns char(22);
return upcase(usubjid);
end;

/*signature 2*/
method usubjid(char(12) stud, char(4) site, char(6) subjid) returns char(22);
dcl char(22) usubj; /*variable local to this method*/
usubj=stud||'-'||site||'-'||subjid;
return upcase(usubj);
end;

method trt01p(char(200) arm) returns char(4);
dcl char(4) trt; /*variable local to this method*/
if upcase(arm)='Treatment 1' then trt='TRT1';
else trt='TRT2';
return(trt);
end;

endpackage;
run;
quit;
```

This simple example exhibits a few services and items offered by this technology. Initially, each method has a return statement, and it is necessary for these methods to indicate in their return statement the variable type. Secondly, one notices that there are two usubjid methods. They have the same name, but different signatures since the parameter calls are different. During the data portion of the DS2 thread, one will see the usefulness of this method overloading.

Another thing to note is that the package is output to a permanent package library,plib, thus enabling other programs to use this package and its methods. Also, one notes that there is a method that declares a variable. This variable is considered local and will cease to exist once the method is done. Any variable within the package will only be local to that package and will also disappear once the package is being used. No package level variables were declared. Lastly, it is important to include the ‘overwrite = yes’ statement so that the package is rewritten every time it runs.

INITIAL DATA FORMATION

Things are now set to create a dataset. The last portion of this example is the data step which reads the data through a thread. The previously created macro variable values will be used to declare the data vector with labels and formatting. The packaged methods are used to create a few variables:

```
proc ds2;
data adsl_start(overwrite=YES keep=(%do j=1 %to &cnt; &&var&j %end;));

%do i=1 %to &cnt;
  dcl &&type&i &&var&i having label %tslit(&&lab&i) &&format&i; /*Global variables*/
%end;
```

```
dcl char(22) usubjid_dm having label 'DM usubjid' format $22.; /*Global variable*/
dcl package plib.bedrock bed();/*Packages need to be declared and given a name: bed()*/
dcl thread dmds dmds; /*Threads also need to be declared and given a name: dmds*/

method run();
  set from dmds threads=2;  /*Thread processes 2 rows at a time*/
    usubjid_dm=bed.usubjid(usubjid); /*signature 1 usubjid method in bedrock package*/
    usubjid=bed.usubjid(studyid,siteid,subjid); /*signature 2 usubjid method in bedrock pkg*/
    trt01p=bed.trt01p(arm);  /*Another example calling a method within a package*/

end;
enddata;
run;
quit;
```

Packages and threads need to be declared similar to variables. Packages are also instantiated which means that memory is allocated for the contents within the package. This instantiation occurs when the parentheses are added at the end of the name it's given. Without the parentheses, the package variable has a name but can't be used. This is another example of the DS2 procedure's sensitivity to both memory allocation and computing efficiencies. A programmer is forced to understand and manually control these things which were previously done automatically by the regular data step and macro language.

Similar to macro calls, methods are called upon to form 3 variables from the bedrock package. The other variables are read directly from the data created by the FEDSQL merge. Then, viola', an initial ADSL dataset with all the required variables including the usubjid_dm variable is created for the sake of this example. A portion of the resulting dataset:

STUDYID	USUBJID	SUBJID	SITEID	AGE	AGEU	SEX	RACE	ARM	TRT01P	usubjid_dm
STUDYID-DS2	STUDYID-DS2-10-3000	3000	10	45	YEARS	M	White	Treatment 1	TRT1	STUDYID-DS2-10-3000
STUDYID-DS2	STUDYID-DS2-10-3001	3001	10	77	YEARS	M	Asian	Treatment 2	TRT2	STUDYID-DS2-10-3001
STUDYID-DS2	STUDYID-DS2-10-3002	3002	10	2	YEARS	M	Asian	Treatment 1	TRT1	STUDYID-DS2-10-3002
STUDYID-DS2	STUDYID-DS2-10-3003	3003	10	56	YEARS	M	Black or African American	Treatment 1	TRT1	STUDYID-DS2-10-3003
STUDYID-DS2	STUDYID-DS2-10-3004	3004	10	28	YEARS	M	Asian	Treatment 2	TRT2	STUDYID-DS2-10-3004
STUDYID-DS2	STUDYID-DS2-10-3005	3005	10	11	YEARS	F	Asian	Treatment 2	TRT2	STUDYID-DS2-10-3005

CONCLUSION

For any dataset then, the process is as such:

1. Import specification details (Proc Import)
2. Adjust details to be used in DS2 procedures (Proc DS2 (imbedded FEDSQL))
3. Create macro variables (Proc SQL, count, data types, data names, labels, formats, informats)
4. Create Thread to query and merge various feeder datasets (Proc DS2)
5. Create Package of variable methods (Proc DS2)
6. Create Data using the macronized specification information (Proc DS2)

Object oriented programming, memory allocation, modular code, threads vs. programs, ANSI FEDSQL queries can all seem strange to the SAS programmer who is used to writing with more of a stream of conciousness. The DS2 procedure seems to write like a collection of character studies that when combined create an overall narrative. It may be important in the future to know how to write in both styles. This example is more of an outline for future OOP novels.

REFERENCES

Eberhardt, Peter. March 2016. *The DS2 Procedure: SAS® Programming Methods at Work.* Cary, NC: SAS Institute Inc.

Kumbhakarna, Viraj R, 2017 “PROC DS2: What’s in it for you?” *Session 0916-2017*: Orlando, FL: SAS Global Forum 2017, ,

ACKNOWLEDGMENTS

I would like to thank my manager Upendra Thapaliya for the support and help in the writing this paper. I would also like to thank Nancy Brucken for answering all my questions and giving me guidance during the submission process.

CONTACT INFORMATION

Matthew Wiedel
Syneos™ Health
402-480-5792
Matthew.wiedel@syneoshealth.com