Avoiding Some Pitfalls

Presented by David Franklin, April 11, 2007

Introduction

When beginning to write SAS code there are three steps that are carried out:

1. planning
2. write the program(s)
3. testing and validating

This presentation will present some useful tips in these three steps to avoid programming pitfalls.

Planning

There are three documents that you must have access to:

- Protocol
- SAP
- Table Shells
- CRF (would be nice if this was annotated)

Also helpful is to do a PROC CONTENTS of any datasets. There is a macro available called ContLstB that produces a document that combines a CONTENTS listing with a formats decode, an example shown below:

DM: Demographics				
Variable Name	**Label**	**Type**	**Format**	**Comment**
R_DRUG	Protocol Identifier (Study Drug)	C8		Database Variable
TAREA	Protocol Identifier (Therapeutic Area)	C40		Database Variable
SCTRY	Protocol Identifier (Study Country)	C4		Database Variable
PNO	Protocol Identifier (Protocol Number)	C15		
CNO	Center Identifier (Center Number)	C3		
PATNO	Patient Identifier (Patient Number)	C3		
CNOPATNO	Center-Patient Identifier	C7		cno||'-'||patno
PATINIT	Patient Initials	C3		
FORM_ID	eCRF Page Identifier	C40		
FORMINDX	RECORDID	N8		
EVENT_ID	Time Point Identifier	C8		
VDSEQ	Visit Sequence	N8		
SEX	Gender	N8	(Format:GENDER) 1=Male 2=Female	

One of the features of the macro shown above is that it is able to incorporate any comments to variables that the user may wish to add, an example of which is shown above for the CNOPATNO variable.

Another useful tip is to print out a few observations from each dataset using the following code:

```
proc sql noprint;
  select memname into :dslist separated by ' '
  from sashelp.vmember
  where libname in('RAWDATA');
  quit;
run;
%let i=1;
%do %while(%scan(&dslist,&i) ne );
  title1 "Dataset: %scan(&dslist,&i)";
  proc print data=rawdata.%scan(&dslist,&i) obs=5;
  run;
  %let i=%eval(&i+1);
%end;
```

Also consider using macros for "blocks" tables - if a group of tables is different only by a population and/or a variable cut then use a macro. A good example of this is the AE tables where they are the same layout and use the same code for counting but only differ in the variable used to select the Adverse Events for the count and/or a population. Another example is if the Demographics table is generated for Safety and ITT populations – the code is the same, the only difference is in the population selection. The advantages of using macros are:

- only one set of code needs to be written – only
- change is the difference in one or more parameter
- values
- only one set of code needs to be maintained
- only one set of code needs to be QC'ed

Write the Program(s)

NO HARDCODES

The title says it all. A hardcode maybe obvious, for example

```
if patid='001-001' then height=178;
```

or less obvious

```
if stop_y>year("&sysdate9"d) then
stop_y=year("&sysdate9"d);
```

If one of the clinical team say to do a hardcode try to talk them out of it, and if this is not possible then discuss with your manager. If hardcodes are necessary then this should be indicated in the log with a WARNING message. No hardcodes should exist in the final version of a table, listing or figure. A hardcode is different from a conversion as is done in vital signs and lab data, just to name a few.

Program to the CRF

The structure of the CRF is useful when writing the code. There are three types of variable – database, monitoring and data variables. Although eCRF has helped in trapping database errors don't assume that it works every time. Try to use data variables where ever possible. An example of a database variable is

```
Record ID
```

a monitoring variable is

```
Are there any Adverse Events? (Y/N)
```

and a data variable is

```
Adverse Event Description
```

Assume Invalid and Duplicate Data

Invalid or Duplicate data is something that should be considered when writing SAS code, particularly in AE, Vital Sign and Lab data. If there is multiple records possible then discuss with the team which to take for analysis and make a note of it in the program as a comment. The following code fragment shows the use for a check of duplicates and invalid data:

```
if n(hghtval) then do;
  select(hghtunt);
    when('CM') hghtsi=hghtval;
    when('IN') hghtsi=hghtval*2.54;
    otherwise
      put 'WAR' 'NING: Unexpected or unknown '
      'height unit: ' cnopatno= hghtval= hghtunt=;
  end;
  end;
  if sum(first.cnopatno,last.cnopatno)<2 then
    put 'WAR' 'NING: Unexpected multiple result: '
    cnopatno= hghtval= hghtunt=;
```

If the variable being analyzed is a numeric but the value is stored as a character it is best to do any comparisons using numeric values. This can be best shown in the following example:

```
if ^missing(labvalue) and labvalue^='0';
```

This line of code is okay when zero means '0' but during the life of a database a value of '0.00' may included that will not be caught.

Do not be afraid to use the log or PRINT procedure to output data considered dubious, the latter usually going to an LST file with the same program name that will not go to Medical Writing.

Dates

This is a special category of its own. If a full date is expected for a particular variable, e.g. Date of Birth, Treatment Date or Concomitant Medication date, then treat it as a full date and put out an exception to a log or LST file if a partial date is given. Only try to use a macro that deals with partial dates if partial dates are expected.

SAS Tip

Need to get a date from variable that is stored as a character, for example

```
03/06/2006 NUL:NUL:NUL
```

If the date should always be a complete date and partial dates are invalid, the SAS Date value can be got using the following statement:

```
INPUT(datestr,MMDDYY10.);
```

If the date can be incomplete there other methods will have to be employed.

Avoid overwriting DB variables

It is not good programming practice to write over data in database variables, even within a program where a temporary (work) dataset is being used. It makes it harder to track the flow of data when data from the stored dataset is being overwritten in the work area.

Initialize Variables

At the very least it is good programming practice to define the type and length of a new variable in a data step. This avoids the issue where SAS will assume a type and length of a newly created variable while the programmer may want another type and/or length.

Use Comments

This makes the program easier to understand when someone else other than yourself is looking at the program – that program may be looked at again in five years time!

Use the Program Header

It is good practice to use the program header. From an informational view the two most important fields are "WHY" and "NOTES" as the former say why the program was written the latter gives any information that shows any particular oddities that may be needed by anyone looking at the program in the future. Even if you are just changing a line of code for whatever reason, keep the NOTES field up to date.

Don't be afraid to use the Help button

If you are not sure of a SAS statement, function or procedure do click the help button or ask someone. To use an example

```
select trtcd, count(*) as count
```

is different to

```
select trtcd, count(age) as count
```

Testing and Validating

Check the Log

This cannot be emphasized enough. Use the QC macro to check if your program is running okay. Internal guidance suggests that there be nothing out of the QC macro for the program. However if after redoing the SAS code their still remains issues then these should be explained.

Review the Output

Just have a quick look at the output and check if the N's are as expected (n should always be less than or equal to N), that page breaks are okay and see if there are results for each parameter are around what is anticipated. Maybe consider doing a quick PROC TABULATE to check that nothing was lost in the code to make the data useful for the REPORT procedure call. It is good practice to spell check the titles and footnotes, and make sure that footnote references actually align with references in the output.

SAS Tip

SAS v9 provided a few new functions to concatenate text strings and variables:

CAT(*string-1 <,... string-n>*)
Concatenates without removing leading or trailing blanks
CATS(*string-1 <,... string-n>*)
Concatenates removing leading and trailing blanks
CATT(*string-1 <,... string-n>*)
Concatenates removing trailing blanks
CATX(*separator, string-1 <,...string-n>*)
Concatenates removing leading and trailing blanks, and inserts separators

This is a touch better than the old

```
trim(left(string-1))||' '||trim(left(string-2));
```

statements that were used prior to version 9.