PharmaSUG SDE, Boston, 2014

There is Data Missing, What's Wrong With Your Program?

David Franklin, Manager Statistical Programming, Quintiles, Cambridge, MA

ABSTRACT

We have all heard it, or a variation of "There is data missing, what's wrong with your program?" You or your team spent a significant amount of time programming the SDTM to the specifications given to you, or maybe you did the specifications yourself and worked from them. Maybe it is another transfer of the data and running the same set of programs to create the SDTM datasets. In all cases the SAS LOGs appeared to go okay with no ERROR messages found.

Interpretation of specifications is certainly one of the reasons why programs to create SDTM datasets do not work as expected, particularly on different transfers of the raw data, but the more common reason is the data and how the SAS program works with it. This paper looks at a number of WARNING and NOTE messages in detail that may indicate that there are serious issues with your SAS programs, each one of them contributing to your SDTM datasets having problems. Also presented is a small macro that will search though all the logs in a directory and search for these issues.

INTRODUCTION

Consider programmers create and run the programs to create the SDTM, but:

- Despite new data, the data still looks like the previous data
- There are variables with values of '*******' instead of data
- Somehow the subset '022-001' has a DOB of 25SEP1963 on the eCRF but with updated data the DOB reads 13OCT1948 and the DOB on the eCRF has not changed
- On some calculations, results that should be present are not there
- Variables have no values
- Variables that have been recalculated in the program appear as unaltered values in the datasets

Avoid this by DEFENCIVE PROGRAMMING and always assume that programs fail with new data, no matter how minor the data change.

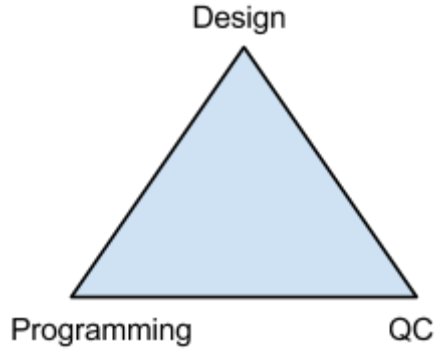
SDTM CONSIDERATIONS

Basic Stages to SDTM:

- Design/Specification (set out the mapping document, including annotating the CRF)
- Programming (programmer takes the specification and writes the program)
- QC (checking that the data is transferred correctly into SDTM)

What anyone should have before considering an SDTM transfer:

- Study Protocol
- Study SAP
- General programming and SDTM mapping guidance and/or SOPs
- CDISC Study Data Tabulation Model Standards (version being used)
- CDISC Study Data Tabulation Model Implementation Guide (version being used)
- CRF



The Programming Triangle

KNOW THY DATA

It is very important that before any review or analysis of data, consider doing a PROC CONTENTS of all datasets and output the first 20 observations of all datasets. Below is a macro that may be useful:

```
%macro (libdir=  /*Libname where data is*/);
  title1 "Dataset Directory";
  proc print data=sashelp.vmember noobs;
    var memname memlabel;
    where libname="%upcase(&libdir)";
  proc sql noprint;
    select memname into :dslist separated by ' '
    from sashelp.vmember
    where libname in("%upcase(&libdir)");
  quit;
  run;
  %let i=1;
  %do %while(%scan(&dslist,&i) ne );
    title1 "Dataset: %scan(&dslist,&i)";
    proc contents data=rawdata.%scan(&dslist,&i) varnum;
    proc print data=rawdata.%scan(&dslist,&i) obs=20;
    run;
    %let i=%eval(&i+1);
  %end;
%mend;

*Example call;
libname rawdata 'c:\study\rawdata' access=readonly;
%prtcont(libdir=rawdata);
```

PLAN FOR MACROS

Advantages for programming:

- only one set of code needs to be written
- 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

An example is processing ISO dates -- one macro can do all processing rather than create individual code for each occurrence of a ISO date.

ALWAYS QUESTION THE DESIGN/SPECIFICATION MAPPING DOCUMENT

People write the specifications are human – check that it makes sense and if you have questions, make them known. Often the Primary Programmer is the first person to review and use the document so is effectively doing a QC, therefore communication between the author of the document and the Programmer Programmer is expected. The QC Programmer will take the document, and in the case of parallel programming, will do the same job and talk with the author of the document with any issues and then compare his/her version with that of the Primary Programmer.

START YOUR PROGRAM OFF WITH A PROC DELETE

If program had ERROR and stopped, dataset is not replaced – strongly recommend use of PROC DELETE or PROC DATASETS with a DELETE statement so that if program fails there is not a reliance on the date/time of a dataset to make sure the program worked.

PROC DELETE is an old procedure recently resurrected with SAS 9.4 but has been present and undocumented for many years. Basic syntax is:

PROC DELETE DATA=<libname.>memname;

or

PROC DELETE DATA=<libname.>_ALL_;

Read more in the SAS documentation for more on PROC DELETE.

NO HARDCODES

In general no hardcodes should be allowed during programming as it means that there is a disjoint between the data from source and the dataset being created. Below is some examples of hardcodes:

Example 1

```
if patid='001001' then height=178; *Data will always be 178, even if new data;
```

Example 2

```
if stop_y>year("&sysdate9"d) then stop_y=year("&sysdate9"d);
  *Difficult to keep track of and data created from one year to the next
   has possibility of changing;
```

Example 3

```
if pt='001001' then do;
if birthdt='25MAY2014'd then do;
  birthdt='25MAY1970'd;
  put 'WAR' 'NING: Hardcode for patient 001001 to '
      'correct birth date issue.';
end;
else put 'WAR' 'NING: Hardcode for patient 001001 '
         'should be reviewed as DOB in DM changed.';
end; *Will track if data changes and put a WARNING message out to the LOG;
```

Example 4

```
if pt='001001' and date()<="31OCT2014"d then
   birthdt='25MAY1970'd; *Time limited change;
```

TYPES OF VARIABLES IN THE CRF

Consider there are three types of variables:

- Database variables (created by the database)
- Monitoring variables (questions like “Did the subject have any AEs?”)
- Data variables (an actual record of an AE)

ASSUME DUPLICATE AND INVALID DATA

When looking at data one should always program assuming that there is duplicate and invalid data as data can be very unclean. An example below is of some code that does conversion, but also checks for invalid units and duplicates:

```
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=;
end;
```

OTHER CONSIDERATIONS

When writing your program and working with the data, also keep in mind:

- Dates (consider partial dates in particular for validity, e.g. 31FEB2014 is invalid but possible as a partial date)
- Avoid overwriting DB variables (good practice, for variables that are created for your program, precede them with an ‘_’; similar for work datasets, precede the name of the dataset with a ‘_’)
- Initialize variables (some modern versions of SAS set a default length of 200 characters for a character variable, older versions generally took the length of the first non-missing character value -- best defined)
- Use comments (the program will long exist well after the one you wrote it has gone, so best comment difficult or unusual code)
- Use a program header (if a client or regulatory agency ask for the code, it is best to have something like a program header stating at the very least the purpose of the program)
- Don’t be afraid to ask for help or ask questions

LOOK AT THE OUTPUT SDTM DATASET AND THE SAS LOG

ALWAYS LOOK AT THE OUTPUT AND LOG.

- At least answer question, does output dataset appear reasonable
- Check the SAS LOG for at least 'ERROR:‘ and 'WARNING:‘ messages, but others worth considering include 'uninitialized‘, 'not exist‘, 'extraneous‘, 'repeats‘, 'not resolved‘, 'converted to numeric‘, 'converted to char‘, 'is already on the library‘, 'sas set option obs=0‘, 'stopped due to looping‘, 'sas went‘, 'unknown‘, 'replaced‘, 'Missing values‘, 'invalid‘, 'At least one W.D format was too small‘, 'truncate‘, 'Unreferenced label‘, 'Truncation may result‘, 'overwritten by data set‘.

Below is a simple SAS LOG checker that can check all the SAS LOG files in a directory for ERROR and WARNING messages:

```
%let logchkdr=%str(C:\MyStudy\SDTM\programs);
data _alllogs0;
  attrib _txt length=$256 informat=$char256.
            format=$char256.
         _fn length=$256
         _myinfile length=$256
         _ln length=8;
  infile "&logchkdr.\*.log"
         lrecl=256 filename=_myinfile;
  input _txt $varying256. len;
  _fn=scan(strip(_myinfile),-1,'\');
  _ln=_n_;
proc sort data=_alllogs0;
  by _fn _ln;
run;
title1 "QC of SAS LOG(s)";
data _null_;
  retain _k 0; *Internal counter;
  file PRINT;
  set _alllogs0 end=eof;
  by _fn _ln;
  if first._fn then do;
     put // @1 '***** LOG FILE: ' _fn /;
     _k=0;
  end;
  if index(_txt,'ERROR:') or
     index(_txt,'WARNING:') then do; *Adapt list for own use;
     _k+1; put _txt;
  end;
  if last._fn and _k=0 then put @1 '** NO ISSUES FOUND';
  if eof then put / '/*EOF*/';
run;
```

The program will put out any issues it finds in the SAS LOG out to the listing, and if no issues are found then a message will still put out the message ‘** NO ISSUES FOUND’.

PROC COMPARE, BUT NOT EVERYTHING MAY NOT SEEM CORRECT

People rely on the note “NOTE: No unequal values were found. All values compared are exactly equal.” in the PROC COMPARE listing to satisfy that two datasets are the same.

- Message will appear if variable exists in one dataset and not the other (message comes up because the common variables are the same)
- Message will appear if records are missing in one dataset but not the other (message come up because the common records are the same)
- Message will appear if variables are different types (message come up because the common variables are the same)

Refer to paper from Joshua Horstman and Roger Muller, PharmaSUG 2013, paper CC36. “Don’t Get Blindsided by PROC COMPARE”.

The following is a macro useful for putting a

```
%macro compds(baseds= /*Base Dataset*/
             ,compareds= /*Compare Dataset*/
             ,byord= /*Order of unique records*/
             );

  *Do compare;
  proc sort data=&baseds;
    by &byord;
  proc sort data=&compareds;
    by &byord;
  proc compare base=&baseds compare=&compareds;
  run;

  *Capture result;
  %let retcode=&sysinfo;

  *Put messages out to SAS LOG;
  data _null_;

    *Initialize;
    retain _sysinfoval &retcode;
    put "******************************************";
    put " PROC COMPARE SUMMARY RESULT";
    put "******************************************";
    put;
    put "Result of Comparison between datasets "
        "%upcase(&baseds) and %upcase(&compareds) are: ";

    *Set up messages;
    array compmsg {16} $ 40 _temporary_ (
      "Data set labels differ", "Data set types differ",
      "Variable has different informat",
      "Variable has different format",
      "Variable has different length",
      "Variable has different label",
      "OLD data set has observation not in NEW",
      "NEW data set has observation not in OLD",
      "OLD data set has BY group not in NEW",
      "NEW data set has BY group not in OLD",
      "OLD data set has variable not in NEW",
      "NEW data set has variable not in OLD",
      "A value comparison was unequal",
      "Conflicting variable types",
      "BY variables do not match",
      "Fatal error: comparison not done"
      );

    *Put return code into bit code, reversing so first is on left;
    _bitcode=reverse(put(_sysinfoval,binary16.));

    *If return code is 0 then compare is identical;
    if index(_bitcode,'1')=0 then put @3 "Datasets are identical";

    *Compare not identical, put out messages;
    else do;
      put @3 "The following issue(s) are found:";
      do i=1 to 16;
        if substr(_bitcode,i,1)='1' then put @5 '-' compmsg(i);
      end;
    end;

    put;
    put @1 '***** END OF COMPARE *****';
  run;
%mend compds;
```

Below are two examples of the macro call:

Example 1

```
%compds(baseds=ADEF, compareds=ADS.ADEF,
        byord=STUDYID USUBJID ADT PARAM AVISITN);

*OUTPUT IN LOG*;
Result of Comparison between datasets ADEF and ADS.ADEF are:
 Datasets are identical
***** END OF COMPARE *****
```

Example 2

```
%compds(baseds=ADSL,compareds=ADS.ADSL,
        byord=STUDYID USUBJID);

*OUTPUT IN LOG*;
Result of Comparison between datasets ADSL and ADS.ADSL are:
 Data set labels differ
 A value comparison was unequal
***** END OF COMPARE *****
```

CONCLUSION

This paper has looked at a number of items to think about, from a programming perspective, when bringing CRF data to SDTM. Also provided was a number of code samples and macros that should be part of any programmers toolkit.

DISCLAIMER

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations,or practices of Quintiles Inc.

CONTACT INFORMATION

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

Name: David Franklin
Enterprise: Quintiles, Inc.
Email: David.Franklin@Quintiles.com