PharmaSUG 2014 – Paper CC09

An Alternative Way to Detect Invalid Reference Records in Supplemental Domains

Steven Wang, inVentiv Health Clinical, Indianapolis, IN

ABSTRACT

Quite often in the process of development of SDTM datasets, we may just be interested in how to identify some specific errors in certain domains without the time and effort of running the OpenCDISC validator. The purpose of this paper is to demonstrate an alternative way to identify the invalid cross-referenced records with error code SD0077 in the supplemental domains by utilizing the SAS® macro language.

INTRODUCTION

In the process of developing SDTM datasets, it is not uncommon to derive XXSEQ or drop some records for some domains while unaware of the existence of their supplemental domains. OpenCDISC will cross-reference each record in the supplemental domains SUPPXX (SUPPAE, SUPPLB, etc.) with a related record in the parent domain. If a record in the supplemental domain cannot be tied to a reference record in the parent domain then it flags an error – "Invalid referenced record" (SD0077). That is, "Reference record defined by Related Domain Abbreviation (RDOMAIN), Unique Subject Identifier (USUBJID), Identifying Variable (IDVAR) and Identifying Variable Value (IDVARVAL) must exist in the target Domain." I will name this kind of records as "orphan records". Because it may take some extra effort and time to set up the running of OpenCDISC validator, it would be more efficient if a simpler SAS program can help to identify these orphan records in a short time.

REQUIREMENTS AND IMPLEMENTATIONS

REQUIREMENTS

Due to the difference in data structures between the supplemental domains and parent domains, there are three major differences that the program should be able to accommodate.

1. Different supplemental domains – Even the data structure of the supplemental domains is quite different from that of the parent domains, but there are similarities among supplemental domains. The program should be equally able to identify all the orphan records from any supplemental domain, regardless of the supplemental domain name.
2. Different variable names - The program should be able to dynamically identify the orphan records regardless of different values of IDVAR which may vary from domain to domain. For example, "AESEQ", "AEGRPID" are possible values of IDVAR for SUPPAE domain, while "LBSEQ", "LBGRPID" are possible values of IDVAR for SUPPLB domain,
3. Different data types - The program should accommodate different data types from different sources. For example, the type of variable AESEQ in SUPPAE is character but is numeric type in its parent domain AE.

For simplicity, this paper will keep the code to a minimum and only show how to meet the three main requirements mentioned above without worrying about error handling or input/output. Also in the code that will be presented below, only variables of the form ending with "SEQ" will be assumed to be numeric type and all other variables are assumed to be character type. These assumptions usually are sufficient in practice and worked well in real projects.

IMPLEMENTATIONS

The first impulse I might have is to use PROC TRANSPOSE to transform the data structure of the supplemental domain to match what is in the parent domain. However, this approach may have a hard time to dynamically deal with different variable names in different domains. So I took another approach which I will call as "macro array approach".

Briefly speaking, this approach will collect all the necessary information and store the collected information into "macro arrays" by PROC SQL.

Here is the beginning of the code. Parent domain will be denoted by `parentset` and therefore supplemental domain is denoted by `supp&parentset`. The parameter `inlib` denotes the input directory where both domains are stored.

```
%macro orphan(inlib=, parentset=);
data supp&parentset;
    set &inlib..supp&parentset;
run;

proc sql noprint; /*analyze the IDVAR column of the supplementary domain*/
    create table di_dvar_&parentset as
    select distinct IDVAR from supp&parentset
    order by IDVAR;

    select count(*) into :rcnt from di_dvar_&parentset;

    select IDVAR into
                :_ds_idvar1 - :_ds_idvar%sysfunc(strip(&rcnt))
    from di_dvar_&parentset;
quit;
```

In the supplemental domain, only distinct values of IDVAR will be read and stored into “macro array” `&&_ds_idvar&i`. Each IDVAR value stored in `&&_ds_idvar&i` (for each i) will be supposed to correspond to a variable in the parent domain. The actual values of such variable will be obtained from IDVARVAL, which will be compared with those in the parent domain.

```
%do i=1 %to &rcnt;
%let sortkey=USUBJID &&_ds_idvar&i;
… … …
proc sort data=_tmp_supp&parentset.&i; by &sortkey; run;
proc sort data=&inlib..&parentset out=&parentset; by &sortkey; run;

data _orphan_&parentset._&&_ds_idvar&i(keep=&keepkey);
    merge _tmp_supp&parentset.&i(in=a) &parentset(in=b);
    by &sortkey;
    if a and not b;
run;
%end;
```

As it runs through all the counters, all values of IDVAR are compared with the parent domain and any orphan records will be kept in the dataset `_orphan_&parentset._&&_ds_idvar&i`.

Since some variables in different domains may have different data types, some type changing mechanism will be implemented as follows:

```
proc sql noprint;
    create table tmpvar as
    select distinct * from dictionary.columns
    where libname=upcase("&inlib") and memname=upcase("&parentset") and
    name=upcase("&&_ds_idvar&i") ;

    select type into :_vartp from tmpvar;
quit;

data _tmp_supp&parentset.&i;
    set supp&parentset;
    if idvar="&&_ds_idvar&i" or idvar=upcase("&&_ds_idvar&i");

    %if &_vartp=num %then %do;
                &&_ds_idvar&i=input(IDVARVAL,best.);
    %end;
    %else %do;
           length &&_ds_idvar&i $ 200;
           &&_ds_idvar&i=IDVARVAL;
    %end;
run;
```

The idea of the above code is to get the data type `&_vartp` of the variable `&&_ds_idvar&i` before each run and if it is numeric type, then change it from the supplemental domain. Otherwise, set the length to 200 and keep the value

as is. Then create a variable `&&_ds_idvar&I` with the value `IDVARVAL` in the supplemental domain. When merging the supplemental domain with the main domain, setting the sort keys to be `USUBJID &&_ds_idvar&I`.

That is about it. Integrate all the pieces together into a short macro and it can do the work. A full version of the code will be attached in the appendix.

SOME SAMPLES

To show that this macro really works. I made up a couple of examples to illustrate this.

I first created a pair of datasets AE and SUPPAE as follows.

```
data ae;
input USUBJID AESEQ AEDECOD$ AEGRPID;
datalines;
1 1 a 1
1 2 b 1
1 3 c 1
1 4 d 1
2 1 x 1
2 2 y 1
2 3 z 1
;
data suppae;
input USUBJID RDOMAIN$ IDVAR$ IDVARVAL QNAM$;
datalines;
1 AE AESEQ 1 x1
1 AE AESEQ 2 x2
1 AE AESEQ 5 x4
1 AE AESEQ 6 x5
1 AE AEGRPID 1 x4
1 AE AEGRPID 2 x5
1 AE AEGRPID 4 x6
2 AE AESEQ 1 y1
2 AE AESEQ 4 y3
;
```

After I called the macro,

```
%orphan(inlib=work, parentset=ae);
```

I got the output as the following:

(work._orphan_ae_aegrpid)

	USUBJID	IDVAR	IDVARVAL
1	1	AEGRPID	2
2	1	AEGRPID	4

(work._orphan_ae_aeseq)

	USUBJID	IDVAR	IDVARVAL
1	1	AESEQ	5
2	1	AESEQ	6
3	2	AESEQ	4

In this example, one can see the macro can pick out the orphan records from SUPPAE for numeric `AESEQ` or `AEGRPID`.

Then I tried another pair of datasets LB and SUPPLB as follows with numeric `AESEQ` and character `AEGRPID`.

```
data lb;
input USUBJID LBSEQ LBDECOD$ LBGRPID$;
datalines;
1 1 a t
1 2 b t
1 3 c t
1 4 d t
2 1 x t
2 2 y t
2 3 z t
;
```

```
data supplb;
input USUBJID RDOMAIN$ IDVAR$ IDVARVAL$ QNAM$;
datalines;
1 LB LBSEQ 1 x1
1 LB LBSEQ 2 x2
1 LB LBSEQ 5 x4
1 LB LBSEQ 7 x5
1 LB LBGRPID t x4
1 LB LBGRPID s x5
1 LB LBGRPID r x6
2 LB LBSEQ 1 y1
2 LB LBSEQ 4 y3
;
```

Now I called the macro again

```
%orphan(inlib=work, parentset=lb);
```

And the output is

(work._orphan_lb_lbgrpid)

	USUBJID	IDVAR	IDVARVAL
1	1	LBGRPID	r
2	1	LBGRPID	s

(work._orphan_lb_lbseq)

	USUBJID	IDVAR	IDVARVAL
1	1	LBSEQ	5
2	1	LBSEQ	7
3	2	LBSEQ	4

Again, the macro correctly identified the orphan records for SUPPLB.

CONCLUSION

A short but efficient and independent macro is worth the time and effort as a supplement of the OpenCDISC validator to pinpoint some specific issues in-stream. Some assumptions such as coexistence of the main and the supplemental domains as well as the necessary variables in both domains make the code simpler without dealing with any error-handling or input/output issues. Once the orphan records are identified, either remove them from the supplemental domain or derive the records in the parent domain as necessary. Also this approach can be extended to check out other cross-reference issues specified by OpenCDISC report.

REFERENCES

http://www.opencdisc.org/

ACKNOWLEDGMENTS

I would like to thank Keith Hibbetts and Daniela Popa.for their support and help to refine this paper through their valuable feedback.

CONTACT INFORMATION

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

Steven Wang
inVentiv Health Clinical
Indianapolis, IN
(317) 439-9917
steven.wang@inventivhealth.com

APPENDIX – Full Source code

```
%macro orphan(inlib=, parentset=);

    data supp&parentset;
        set &inlib..supp&parentset;
    run;

    proc sql noprint;   /*analyze the IDVAR column of the supplementary domain*/
           create table di_dvar_&parentset as
           select distinct IDVAR from supp&parentset
           order by IDVAR;

           select count(*) into :rcnt from di_dvar_&parentset;

           select IDVAR into
                       :_ds_idvar1 - :_ds_idvar%sysfunc(strip(&rcnt))
           from di_dvar_&parentset;
    quit;

    %let keepkey=USUBJID IDVAR IDVARVAL;

    %do i=1 %to &rcnt;
           %let sortkey=USUBJID &&_ds_idvar&i;

           proc sql noprint; /*find the TYPE of the variable in the parent domain*/
                  create table tmpvar as
                  select distinct * from dictionary.columns
                  where libname=upcase("&inlib") and memname=upcase("&parentset")
                  and name=upcase("&&_ds_idvar&i") ;

                  select type into :_vartp from tmpvar;
           quit;

           data _tmp_supp&parentset.&i;
                  set supp&parentset;
                  if idvar="&&_ds_idvar&i" or idvar=upcase("&&_ds_idvar&i");
                  /*filter the supplementary domain by IDVAR and create a new
                  variable by the name of IDVAR*/
                  %if &_vartp=num %then %do;
                              &&_ds_idvar&i=input(IDVARVAL,best.);
                  %end;
                  %else %do;
                  length &&_ds_idvar&i $ 200;
                              &&_ds_idvar&i=IDVARVAL;
                  %end;
           run;

           proc sort data=_tmp_supp&parentset.&i; by &sortkey; run;
           proc sort data=&inlib..&parentset out=&parentset._tmp(keep=&sortkey);
           by &sortkey; run;

           /*Merge with the parent domain to find out the orphans*/
           data _orphan_&parentset._&&_ds_idvar&i(keep=&keepkey);
                  merge _tmp_supp&parentset.&i(in=a) &parentset(in=b);
                  by &sortkey;
                  if a and not b;
           run;

    %end;*end i;
%mend;
```