PharmaSUG2012 – Paper PO15

Automate Validation of CDISC ADaM Variable Label Compliance

Wayne Zhong, Octagon Research Solutions Inc.

ABSTRACT

A sometimes overlooked feature of the CDISC ADaM Implementation Guide is that it specifies label names for each ADaM variable. To improve quality of delivery and reduce the tedious task of checking variable labels, this paper presents techniques for producing a quick compliance report.

Details include how to check ADaM variables with placeholders like **xx** and *****, and how to detect at the dataset metadata level a few cases of ADaM non-compliance. The latest CDISC ADaM metadata can be stored in an EXCEL spreadsheet, and easily updated with future ADaM label rules.

INTRODUCTION

The Clinical Data Interchange Standards Consortium (CDISC) publishes the Analysis Data Model (ADaM) Implementation Guide (ADaMIG), a document that standardizes dataset structures and variables. This paper primarily covers checking for the correct application of labels to variables specified in the ADaMIG, however the following checks can be seamlessly included: correct variable type (ADaM variables only), variable name length (≤8), variable label length (≤40), and variable length (≤200).

The ADaMIG specifies three types of variables: the simplest are completely defined like **USUBJID** and **AGE**, some variables like **RACEGRy** and **TRTxxP** have placeholders (xx, y, or zz) for numeric values, and lastly variables like ***DT** and ***SDT** can have any prefix. The last two types of variables require care to identify from dataset metadata and check for label compliance.

METADATA

SAS® offers many ways to get a list of variable names and labels from a dataset. Suppose a quick check of all analysis datasets in the library DERLIB is desired, the following code accomplishes this and Display 1 shows the result. Some quick checks of variable name and label lengths using the LENGTH() function and variable length using variable LENGTH can be added here.

```
libname derlib ' your path here… ';

data metadata;
    set sashelp.vcolumn;
    where libname='DERLIB' and memtype='DATA';
    keep memname name type length label varnum;
run;
```

VIEWTABLE: Work.Metadata

	MEMNAME	NAME	TYPE	LENGTH	VARNUM	LABEL
1359	ADSL	STUDYID	char	10	1	Study Identifier
1360	ADSL	USUBJID	char	20	2	Unique Subject Identifier
1361	ADSL	SUBJID	char	10	3	Subject Identifier for the Study
1362	ADSL	SITEID	char	10	4	Study Site Identifier
1363	ADSL	INVNAM	char	200	5	Investigator Name
1364	ADSL	AGE	num	8	6	Age
1365	ADSL	AGEU	char	6	7	Age Units

Display 1

ADaM Label Data

A repertoire of ADaMIG label information is needed, EXCEL is the presented method here however any SAS readable file will do. Display 2 shows a sample of label information entered, please note that "…" was replaced with * to avoid a quirk with EXCEL.

	A	B	C
1	NAMEA	LABELA	TYPEA
2	STUDYID	Study Identifier	Char
3	USUBJID	Unique Subject Identifier	Char
4	TRTxxP	Planned Treatment for Period xx	Char
5	TRTxxPN	Planned Treatment for Period xx (N)	Num
6	*DT	Date of *	Num
7	*SDT	Start Date of *	Num
8	CHGCATy	Change from Baseline Category y	Char

Display 2

Calling the IMPORT procedure turns this spreadsheet into a SAS dataset. The addition of the suffix A to column names allows easier joining later on.

```
proc import datafile=' your excel file path here.xls '
    out=alabel replace dbms=xls;
run;
```

LIKE OPERATOR

In order to compare dataset metadata against ADaM label data, ADaM variables must first be identified. The LIKE operator found in the SQL procedure helps accomplish this. The LIKE operator recognizes two wildcard characters: '_' which matches any character and '%' which matches any sequence of 0 or more characters, all other characters must match that character. For example, "AL_" matches "ALP" but does not match "AL" or "ALPS". "AL%" matches "AL", "ALP", and "ALPS".

Two source datasets, an SQL INNER JOIN using the LIKE operator, and the resulting dataset are presented below. Some variables are not shown.

```
Dataset:  METADATA        Dataset:  ALABEL
Variable: (NAME)          Variable: (NAMEA)
Values:   USUBJID         Values:   USUBJID
          TRT01P                    TRTxxP
          TRTACP                    *DT
          TRTSDT                    *SDT


proc sql;
    create table check1 as
        select a.*, b.*, length(b.namea) as len from
        metadata a inner join alabel b on
        a.name like translate(b.namea,'___%','xyz*')
        order by memname, varnum, len;
quit;


Dataset:  CHECK1
Variable: (NAME)        (NAMEA)       (LEN)
Values:   TRT01P         TRTxxP        6
          TRTACP         TRTxxP        6
          TRTSDT        *DT            3
          TRTSDT        *SDT           4
          USUBJID       USUBJID        7
```

In the SQL procedure, the TRANSLATE function is used to convert ADaMIG wildcard characters " xyz* " to either "_" or "%" LIKE operator wildcards. Another variable LEN is created, its purpose is evident from dataset CHECK1: joining using the LIKE operator with wildcards will cause a many-to-many Cartesian join so the length LEN of the ADaM name variable helps determine which match is best: the longest.

While no requirements are made for the prefix for *DT and the like, the same is not true for variables such as TRxxP where xx is required to be two integers so we see that TRACP needs to be flagged as an non-ADaM variable. This is visited in the next section.

CHECKING INTEGERS

A reduced CHECK1 dataset is used for the example in this section. As a reminder, the variable LABELA holds the ADaMIG specified labels. In the DATA step, first the best match from the previous step is kept, then if any of (“xx” “y” “zz”) is found in the ADaM name, the correponding characters in the variable NAME receives the COMPRESS function to remove digits. If the result is not missing, then the variable is determined to be not ADaM. If the result is missing, then those corresponding characters (now determined to be numbers) are added to the ADaMIG label using the TRANWRD function.

```
Dataset:  CHECK1
Variable: (NAME)  (NAMEA) (LABELA)
Values:   TRT01P   TRTxxP  Planned Treatment for Period xx
          TRTACP   TRTxxP  Planned Treatment for Period xx
          TR01PQ2  TRxxPQy Planned Pooled Treatment y for Period xx
```

```
data check2;
    set check1;
    by memname name len;
    if last.name;

    if index(namea,'xx')>0 then do;
        if compress(substr(name,index(namea,'xx'),2),,'D') ne '' then delete;
        else labela=tranwrd(labela,' xx',' '||substr(name,index(namea,'xx'),2));
    end;
    if index(namea,'y')>0 then do;
        if compress(substr(name,index(namea,'y'),1),,'D') ne '' then delete;
        else labela=tranwrd(labela,' y',' '||substr(name,index(namea,'y'),1));
    end;
    if index(namea,'zz')>0 then do;
        if compress(substr(name,index(namea,'zz'),2),,'D') ne '' then delete;
        else labela=tranwrd(labela,' zz',' '||substr(name,index(namea,'zz'),2));
    end;
run;
```

```
Dataset:  CHECK2
Variable: (NAME)  (NAMEA) (LABELA)
Values:   TRT01P   TRTxxP  Planned Treatment for Period 01
          TR01PQ2  TRxxPQy Planned Pooled Treatment 2 for Period 01
```

Rather than deleting non-ADaM variables, it may be preferable to output them separately in a report. The new ADaM labels are ready for a direct comparison with metadata labels except for…

WILDCARD LABELS

An opportunity to check labels such as Date of * and Start Date of * existed in the SQL step earlier in the paper but was omitted for clarity. The LIKE operator is used again and this time on LABEL and LABELA to check all metadata labels for ADaM compliance.

```
proc sql;
    create table check3 as
        select *, case
                    when label like translate(labela,'%','*') then 'Y'
                    else 'N'
                  end as compliant from check2;
quit;
```

A check for variable type compliance (Char or Num) can be added either here or in the previous step. As a challenge, this step can be eliminated altogether and folded into the two previous steps. For non-compliant labels as well as any other error check findings, an output XLS file or a simple print can be produced.

CONCLUSION

With the techniques presented in this paper, ADaM label checking can be done with a few clicks. Label checking is just one of many generic checks that can be implemented to improve the quality of ADaM datasets, the code shown can be run independantly but can also become a macro as a part of a larger suite of ADaM compliance and quality assurance tools.

CONTACT INFORMATION:

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

Author Name:	Wayne Zhong
Company:	Octagon Research Solutions Inc.
Address:	585 East Swedesford Road, Wayne, PA
Work Phone:	(610) 535-6500 x5535
Email:	wzhong@octagonresearch.com
Web:	www.octagonresearch.com