Automated Excel-lent Validation

Janet Stuelpner, Left Hand Computing, Inc.

ABSTRACT

During the clinical trials process, the raw data is entered into a database, cleansed, manipulated and then analyzed. Prior to tables and listings, analysis datasets are created that include both raw data and derived data. A great deal of documentation is required to make sure that all of the algorithms are specified and implemented correctly. Most often, the documentation for these analysis datasets is in the form of an excel spreadsheet that states the variable names, labels, variable type, length of value, algorithm for derivation, dataset of origination, whether the variable is raw data or derived, etc. Once the data specifications are complete, the analysis dataset is created. At this point, before further manipulation takes place, there must be a validation process to insure that the data is correct and reliable. This paper shows an automated method to compare the excel spreadsheet that contains the data specifications and compares it to the actual analysis dataset.

INTRODUCTION

The data has been entered into a database. It has been reviewed, queries sent out to the investigators and answers have been received, edit checks have been completed. The raw data is ready for processing and analysis. What is the next step? Specifications must be written to create analysis datasets for safety and efficacy. Common variables need to be identified and created. Dataset specific variables must be identified and an algorithm defined to create each one of them. The person creating the specifications may not be (and probably is not) the person that creates the analysis dataset. To be sure that everything is completed according to specifications, a validation process must be put into place. The specifications for each dataset can be put into an excel spreadsheet and validation can be done automatically by reading the data from the spreadsheet and creating SAS© datasets to use as a comparison to the dataset that holds the data. Let's see what types of data the spreadsheet might contain, the types of datasets that hold the data and how we can automate the process.

THE SPREADSHEET

Each organization must stipulate what information is common to all datasets within a study. I am going to refer to this information as the common variables. Common variables will include a combination of raw and derived data. The protocol number, investigator, subject number, gender, race, country are some of the raw data components. This is data that is taken directly from the case report form (CRF). Some of the derived components of common variables might include: age, various flags (e.g., evaluabilty population, safety population) and an additional variable that contains the character data that has been converted from the numeric fields. An example of the common variables can be found in table 1 below. These are variables that might populate a dataset of its own, but will probably be included in each of the analysis dataset. The variables that are particular to one type of dataset are the next ones to be discussed. For example, you want to have a spreadsheet devoted to adverse event, laboratory or dosing data. The rest of the spreadsheet would contain those variables, both raw and derived that pertain to a specific type of data. For example, in the adverse events spreadsheet the data of interest might include two types of data, just like the common variables. The adverse event dataset will include raw data that is taken directly from the CRF and derived data where a specific algorithm is captured in the analysis dataset specification. The raw fields might include: start date of the adverse event, stop date of the adverse event, severity of the event, relevant adverse event term, seriousness of the event (mild, moderate, severe), action taken (stop drug, reduce dose), outcome of event, relationship to the study medication. Some of the derived fields could be: duration, dose of study drug at time of event. The spreadsheet will contain all of the relevant information for the adverse event file. See Table 2 for an example of the data in the adverse event analysis dataset.

THE PROGRAM

The way that the data is processed (i.e., the type of program that is written) will depend on the SAS products at your site. The focus of this paper will be the lowest common denominator. What I mean is that the code presented here will assume the core product base that is consistent with all SAS users. All sites have the base SAS product. In the base product, we can use the DATA step to read the data into a SAS dataset. In order to read the data in, the data must be in a format that the DATA step can read. Therefore, the spreadsheet needs to be saved in a comma separated variable (.CSV) file. The .CSV format is available for all versions of SAS and all SAS products. So, the next thing that must be done after the spreadsheet is created is that it must be stored as a .CSV file. This is easy to do in Excel. (Note: to view the .CSV file so that you can see the way the data is actually formatted, open the file with WordPad). Once the data is saved, the program retrieves the data using a DATA step to read it into a SAS dataset. Generally, the file contains labels or headers for each column. The program is written so that the header does not have to be removed. The program will start to read the file from the second line rather than the first. One thing that needs to be remembered is that Excel keeps a carriage control character at the end of the line. This affects the last column of the spreadsheet. Because of this, make your input file one column longer than it needs to be. If you create the file with an extra column that is not read into SAS, the result will be easier to manipulate the other columns once the SAS dataset is created. Another thing to remember is that SAS will only look at the first 20 records to determine the type of data that is contained in that field. If you have a column that does not have any data in the first 20 records, SAS assumes that the field is a character variable. When reviewing the code that is generated by the SAS Import Wizard, check the variables length and data type. Make any corrections that are necessary. Remember that you are the one who knows your data the best.

Once the data is in the SAS dataset, we need to get the data definition from the analysis dataset. We can do this in many ways, but I have chosen to use a PROC SQL and read data from DICTIONARY.COLUMNS. You can create a table (or a dataset) that contains the information that you want to compare with the specifications. Remember, that the dictionary tables can only be used in a PROC SQL because DICTIONARY is a libref that is reserved for use by PROC SQL. To see how each dictionary table is defined, you can use a DESCRIBE TABLE statement. After you know how a table is defined, you can use its column names to subset the data in a WHERE clause to get more specific information. For example:

```
PROC SQL;
  DESCRIBE TABLE DICTIONARY.COLUMNS;
```

```
create table DICTIONARY.COLUMNS
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   name char(32) label='Column Name',
   type char(4) label='Column Type',
   length num label='Column Length',
   npos num label='Column Position',
   varnum num label='Column Number in Table',
   label char(256) label='Column Label',
   format char(16) label='Column Format',
   informat char(16) label='Column Informat',
   idxusage char(9) label='Column Index Type'
  );
```

After running the program, the information can be found in the SAS log. You now have the variable names and can use a WHERE statement in the procedure to create a SAS dataset with the subset of information that is contained in the specifications.

Before the data is merged for the comparison, a couple of variables need to be added to each dataset. A sequence of the variables can be very useful to track variables that are not included in both datasets. The sequence will easily show if there have been any variables that were added or removed between the specifications and the analysis dataset. We can get a sequence by putting the value of _n_ into a variable. The other variable that is added is NAME that is the variable name in the specifications and the analysis dataset. This will be the variable that is used to merge the two files together.

We are now ready to merge the data together and compare the values from each dataset. Flags are created with a value of one whenever the values do not match and a missing value when the datasets match. Then the information is written to an output dataset. If all of the data from the specifications is an exact match with the analysis dataset, a message is printed on the log. However, if there are discrepancies between the two files, a PROC PRINT or PROC REPORT can be produced and the output can be reviewed. The type of output that you produce is dependent on the version of SAS at your site and the type of output that you prefer. With version 8 and higher, the output can be created as a listing, HTML, RTF, PDF, etc. The complete program is included at the end of the paper.

OTHER OPTIONS

Data retrieval

If your company has PROC ACCESS for Excel, the spreadsheet retrieval portion of the program can be different. The DATA step that is used for retrieving the data from the .CSV file is not needed. You can use the IMPORT WIZARD to create the code to capture the data from the Excel spreadsheet and read the data with the PROC ACCESS. Instead of reading the data from a .CSV file with a DATA step, PROC IMPORT is used to read the Excel spreadsheet. This allows the user to read data directly from the spreadsheet (.XLS file) without saving the data in a different format. Your Excel spreadsheet can contain multiple worksheets. Each worksheet can be read from one file individually and converted into a SAS dataset. Capturing the code from the IMPORT WIZARD will allow you to edit the code or create a macro so that it can be used repeatedly.

Data comparison

Another way to compare the data in the specification and the analysis SAS datasets is to use PROC COMPARE. This can be used instead of the DATA step and PROC PRINT. This is a good alternate method to review the data. One of the good features of this procedure is that any variables that are added to either dataset will be flagged at the beginning of the output. There is a section in the PROC COMPARE that lists all of the variables that are unique to a dataset. There is a complication, however, when using the procedure. If a record is added to either dataset at any point in the process and the rows in the two datasets are not equal, the PROC COMPARE will flag every record as containing an error. If the record is added at the end, it will not make a significant difference to the output. However, if the record is added toward the beginning of the dataset, this will make your output long and cumbersome to review. It will compare the values of all of the records for all of the variables. At the beginning of the validation process, this can lead to a great deal more review and add quite a bit of time to the process. This method is a great way to document the final specification listing along with the final analysis dataset. It is certainly a good choice when used in the correct manner.

Comparison of Different Versions of the Specifications

The program can be easily adapted to compare the various iterations of the specification. This is a great feature because you can see the changes that were made and review them for error in the way that the change was made. The second part of the program that reads the contents of the analysis dataset can be changed to read a second set of data specifications from a .CSV file. Then compare the two SAS datasets and see where the differences occur.

CONCLUSION

The SAS system has many wizards and windows where the code can be captured, tailored and used. The automation of the process to compare the data specifications with the information in the analysis data set makes the review process easier and more time efficient. The program can point out discrepancies very quickly and for a large number of files in one program. The program can be easily adapted to compare specifications against each other. When several versions of the specifications are prepared, the changes that are made can be reviewed easily with this method of automating the review process. This will allow for consistent specifications and analysis datasets. The program can be created in such a way as to have all of the protocol specific information at the beginning of the program. Once the changes have been made, the program can be run.

AUTHOR CONTACT

Janet Stuelpner
Left Hand Computing, Inc
326 Old Norwalk Road
New Canaan, CT 06840

(203) 966-7520 voice
(203) 966-8027 fax
jstuelpner@snet.net

Common Variables

Variable name	Variable Label	Type	Length	Decodes /Format	Raw/ Derived	Derivations	Comments	CDISC Variable
PROTOCOL	Protocol Number	Char	12		Raw	=DEMOG.PROTOCOL		Yes
INVID	Investigator Number	Num	8		Raw	= DEMOG.INVID		No
SITEID	Site ID	Num	8		Derived	= DEMOG.SITE		No
SUBJID	Subject ID	Char	5		Derived	=DEMOG.PATNO (convert to character)		Yes
AGE	Age (Years)	Num	8		Derived	= DEMOG.AGE		Yes
AGEUNITN	Age Unit {N}	Num	8		Derived	= DEMOG.AGE		Yes
SEXN	Sex {N}	Num	8	1=Male 2=Female	Raw	= DEMOG.SEX		No
SEX	Sex {C}	Char	6		Derived	= 'Male' if DEMOG.SEXN=1 Else = 'Female' if A_AE.SEXN=2 Else = ' '		Yes
RACEN	Race {N}	Num	8		Raw	= DEMOG.RACE		No
RACE	Race {C}	Char	8		Derived	= 'White' if DEMOG.RACE = 1 Else = 'Black' if DEMOG.RACE = 2 Else = 'Hispanic' if DEMOG.RACE = 3 Else = 'Asian' if DEMOG.RACE = 4 Else = 'Other' if DEMOG.RACE = 5 Else = ' '		Yes
COUNTRY	Country	Char	3		Raw	= DEMOG.COUNTRY		Yes
WEIGHT	Baseline Weight (kg)	Num	8		Raw	= DEMOG.WEIGHT		No
HEIGHT	Baseline Height (cm)	Num	8		Raw	= DEMOG.HEIGHT		No
SAFETY	Safety	Num	8	0 = No 1 = Yes	Derived			No
EVAL	Evaluable	Num	8	0 = No 1 = Yes	Derived			No
TRTCD	Treatment	Num	8	1 = 1,2,3 2 = 2,3,1 3 = 3,1,2 4 = 1,3,2 5 = 2,1,3 6 = 3,2,1	Derived			Yes
DMREFDT	First Dose of ANY Drug	Num	8	Date9.	Derived	= DRUG.PILL from the first record in DRUG for each subject		Yes

Table 1

Dataset Specific Variables
Example: Adverse Event Data

Variable name	Variable Label	Type	Length	Decodes /Format	Raw/ Derived/Linked	Derivations	Comments	CDISC Variable
AETERM	Reported Term	Char	120		Raw			Yes
AEDECOD	Dictionary Term	Char	120		Raw			Yes
AEBODSYS	Body System	Char	120		Linked			Yes
AESTDT	Start Date of Event	num	8	ddmmmyyyy	Raw			Yes
AEENDT	End Date of Event	num	8	ddmmmyyyy	Raw			Yes
AESTTM	Start Time of Event 24 hr clock	Num	8		Raw			Yes
AEENTM	End Time of Event 24 hr clock	Num	8		Raw			Yes
AEDUR	Duration of event in days	Num	8		Derived	End Date - Start date + 1		Yes
AESEV	Severity/ Intensity of Event	Char	8	1 = Mild 2 = Moderate 3 = Severe	Raw			Yes
AESER	Serious Criteria	Char	3	0 = No 1 = Yes	Raw			Yes
AEACTTRT	Study Drug Action Taken	Char	30	1=None taken 2=Reduced 3=Interupted 4=Stopped Permanently	Raw			Yes
AEOUT	Outcome of Event	Char	30	1 = Ongoing 2 = Recovered 3 = Recovered with Sequelae 4 = Death 5 = Lost to follow-up	Raw			Yes
AEREL	Causality (Related To Treatment)	Char	18	1 = Not Related 2 = Possibly Related 3 = Probably Related 4 = Definitely Related				Yes
AEONGO	Ongoing Adverse Event	Char	3	0 = No 1 = Yes	Derived	Yes or No		Yes

Table 2

THE PROGRAM

```
/****************************************************************************
* program:  compare_data_and_specs.sas
* author:   Janet Stuelpner
* purpose:  check analysis data sets against data specifications
*---------------------------------------------------------------------------
* Program design
* 1. one macro created that can be run for each analysis data set
* 2. import csv file with analysis data set specs into SAS data set
* 3. get a contents of analysis data set variables into a SAS data set
* 4. compare 2 data sets and print out any discrepancies
****************************************************************************/
/*clean session before running program*/
options source source2;
dm 'cle log';
dm 'cle out';
title;
footnote;
/** user define global macro variables **/
%let protocol=nda0505;
%let drug=thecure;
%let drugname=Thecure;
%let lib=MYDATA;
%let test=N;

/** libname statement for each protocol **/
libname spec "/&drug/&protocol/testdata";

/** user defined macros **/
%macro setup(env=);
 %global pop;
 options nocenter ls=178 ps=65 pageno=1 nodate;
 /*used for testing and debugging program*/
 %if &env=T %then %do;
   options mprint symbolgen mlogic;
   %let pop=%str(where pno in(3,4,6));
 %end;
 %else %do;
   options nomprint nosymbolgen nomlogic;
   %let pop=;
 %end;
 %mend setup;

%setup(env=P);

/*************************/
/** program starts here **/
/*************************/
%macro compare(spec,dsn,prot);
    /** spec=data set name for analysis dsn **/
    /** dsn =csv file name                  **/
    /** prot=protocol sub directory         **/
    /** read data from .csv file and create SAS data set using data step*/
 data work.&spec (keep=name slabel stype slength);

    /** clear ERROR detection macro variable **/
 %let _EFIERR_ = 0;
    infile "/&drug/&prot/testdata/&dsn..csv" delimiter = ',' missover dsd ;
    format NAME $10. ;
    format SLABEL $40. ;
    format STYPE $4. ;
    format SLENGTH 3. ;
    informat NAME $10. ;
    informat SLABEL $40. ;
    informat STYPE $4. ;
    informat SLENGTH 3. ;
    input  NAME $
           SLABEL $
           STYPE $
           SLENGTH
           ;
   /** starting record/number of records **/
   /** first row is column headings      **/
    if  _n_ ge 2 then do;
        output;
        if _ERROR_ then call symput('_EFIERR_',1);
      end;
 run;

proc sort data=work.&spec out=specs;
  by name;
run;

data spec;
  length name $32;
  set specs;
  srow=_n_;
  sname=name;
run;

/*Use dictionary.columns to get info about dataset*/
/*Sort data by name of variable                   */
proc sql;
 create table adsn as
 select       name,
              label as alabel,
              format as aformat,
              type as atype,
              length as alength
 from dictionary.columns
 where libname="&lib"
       and memname="&dsn"
 order by name
 ;
quit;

data adsn2;
  set adsn;
  arow=_n_;
  aname=name;
  keep name alabel aformat alength atype arow aname;
run;

/*merge data together and check to see if there are any discrepancies*/
data comb;
   merge spec
         adsn2
         end=eof;
   by name;
   retain check 0;
   if arow=. or srow=. then      failr=1;
   if alabel ne slabel then      faill=1;
   if aname ne sname then  failn=1;
   if alength ne slength then    faild=1;
   if atype ne stype then  failt=1;
   if failr=1 or faill=1 or failn=1 or faild=1 or failt=1 then check+1;
   if eof then call symput('check',put(check,best8.));
 run;

%if &check>0 %then %do;
  /** if GE version 8 then create PDF file **/
  %if &sysver=8.2 %then %do;
   options orientation=landscape;
   ods pdf file="/home/userabc/compare_&prot._&spec..pdf";
  %end;

proc print data=comb split='*';
  where failr=1 or faill=1 or failn=1 or faild=1 or failt=1;
  var  name
       srow   arrow               /* check number of var   */
       failn  sname  aname        /* check variable name   */
       faill  slabel alabel       /* check variable label  */
       faild  slength  alength    /* check variable length */
       failt  stype    atype      /* check variable length */
  ;
  label srow  ='Specs*Sequence'
        arow  ='Analysis DS*Sequence'
        failn ='Name'
        sname ='Specs*Name'
        aname ='Analysis DS*Name'
        faill ='Label'
        slabel='Specs*Label'
        alabel='Analysis DS*Label'
        faild ='Length'
        slength='Specs*Length'
        alength='Analysis DS*length'
        ;
  title1 "Drug:  &drugname  Protocol: &protocol   Analysis Dataset:  &dsn";
  title2 "Discrepancies between specifications and dataset";
  footnote "Program:  /&drug/&protocol/compare_data_specs.sas     Rundate:  &sysdate";
run;
  %if &sysver=8.2 %then %do;
   ods pdf close;
  %end;
%end;
%else %do;
  %put "Drug:  &drugname  Protocol: &protocol   Analysis Dataset:  &dsn";
  %put "No Discrepancies between specifications and dataset";
  %put "Program:  /&drug/&protocol/compare_data_specs.sas     Rundate:  &sysdate";
%end;

%mend compare;

/*run macro for each analysis data set*/
%compare(ae, A_AE, &protocol);
%compare(ds, A_DISPOS, &protocol);
%compare(lb, A_LABV, &protocol);
```