Paper PH-01

The SDTM Programming Toolkit

David Scocca, Rho, Inc., Chapel Hill NC

ABSTRACT

Data standards such as the Study Data Tabulation Model (SDTM) make programmers' lives simpler but more repetitive. The similarity across studies of SDTM domain structures and relationships presents opportunities for code standardization and re-use. This paper discusses the development and use of tools to simplify the process of creating SDTM data sets, with examples of common tasks and the code to implement those tasks. It also discusses the usefulness of a metadata system and presents a general specification for an interface for accessing metadata.

Examples will include mapping study visits, parsing dates, and standardizing test codes.

INTRODUCTION

The Study Data Tabulation Model (SDTM) developed by the Common Data Standards Consortium (CDISC) is the approved way to provide clinical data for regulatory submissions.

http://www.cdisc.org/sdtm

We generally collect clinical data in formats other than SDTM. CDISC has a separate standard for data collection (Clinical Data Acquisition Standards Harmonization, or CDASH) and data management and EDC tools store clinical data in a variety of other data structures.

As SDTM programmers, we are assigned the task of converting clinical data into the standardized structures of SDTM following specifications provided by data managers, data standards specialists, or statisticians. Because the output structures are standardized, many of the programming tasks are similar both across the standardized data domains of a clinical study and across a wide range of clinical studies. With standard programming tools and utilities for these tasks we can make data conversion an easier and more robust process.

METADATA

The standard data structures of SDTM are easily described and stored as metadata; for a more extensive discussion of a metadata-driven process, see DiIorio and Abolafia (2007). Some metadata, such as the names, types, and labels of variables in domains, is effectively predetermined by the SDTM specification. A particular clinical study will require a customized meta database which will additionally specify:

- which subset of the domains in the SDTM specification will be used
- which custom domains for findings, events, or interventions will be used
- for each domain used, which of the permissible (optional) variables will be used and which will be omitted
- how the fields in the clinical data will map into the fields in the SDTM data

Storing and editing SDTM metadata involves tools that are outside the scope of this paper; see Abolafia and DiIorio (2008) for a discussion of the construction of tools for metadata management.

RETRIEVING METADATA: %setupSDTM

Because metadata storage and editing systems can vary so much, instead of providing sample code I'm going to provide a basic specification for a metadata access macro. Within this macro, your implementation will address your specific metadata storage system.

The **%setupSDTM** macro will perform two tasks: it will assign the necessary libraries for creating SDTM data sets and it will retrieve the metadata for a particular SDTM domain. This macro will need at least two parameters:

PATH= will specify a base directory beneath which libraries will be defined. Depending on your working environment, you may need to use more than one parameter to specify the necessary libraries or you may have more than one library for each purpose.

- Library from which to read metadata
- Library from which to read non-SDTM clinical data
- Library to which to write SDTM data

DOMAIN= will specify the domain for which you want to read the metadata.

The metadata for the selected domain will be returned in a series of global macro variables:

- **_domainAttrib**: a long ATTRIB statement setting the length, type, and label for all variables in the domain. This can be used in a DATA step to define the structure for an output data set.
- **_domainVarList**: a list of the variables in the domain, for use in a KEEP statement when creating the SDTM data set
- **_domainLabel**: the label for the domain data set
- **_keyFields**: a list of the key fields in sort order
- **_hasSuppQual**: a flag for whether there will be a supplemental qualifier data set for this domain
- **_suppQualAttrib**: an ATTRIB statement to define the structure of the supplemental qualifier data set
- **_suppQualVarList**: a list of the variables in the supplemental qualifier data set

(An ATTRIB statement can be easier to build and return than a series of LENGTH and LABEL statements since a single variable's attributes are adjacent rather than spread across multiple statements.)

For the examples below, I will also assume that the value of DOMAIN passed to the **%setupSDTM** macro is also stored in the global macro variable DOMAIN.

Examples

These returned global values can then be used in your SAS® code. For example, assume you have called **%setupSDTM** for the SDTM domain DM:

```
%global DOMAIN ;
%let DOMAIN=DM;
%setupSDTM(PATH=O:\XPharma\DrugY\StudyZ, DOMAIN=&DOMAIN.) ;
```

You can then read a clinical data set RAW.DEMOG into a temporary data set with all variables properly defined for the SDTM domain using the **_domainAttrib** macro variable:

```
data work.&DOMAIN._1 ;
   %unquote(&_domainAttrib.) ;
   set raw.DEMOG ;
   [...]
  run ;
```

You can create the final SDTM data set from a draft data set using the **_keyFields** macro variable to sort the data set and the **_domainVarList** macro variable to remove unwanted columns:

```
proc sort data=work.&DOMAIN._1
          out=work.&DOMAIN._Final(keep=&_domainVarList.) ;
   by &_keyFields. ;
  run ;
```

You can use similar code to structure and create a supplemental qualifier data set by using the **_suppQualAttrib** and **_suppQualVarList** macro variables.

BUILDING PROGRAMMING TOOLS

After writing a few SDTM programs, you will start to recognize the repeated tasks that can be addressed by common code. At this point, you will want to start developing standardized code, in the form of macros or user-written SAS functions, to perform these tasks. Having standard code will both speed up your programming and eliminate opportunities for errors.

What follows are some strategies for dealing with common tasks along with sample code to implement a basic solution. The macros can be either included directly in SDTM programs or stored in project-specific, client-specific, or company-wide AUTOCALL libraries for all programs to use.

PARSING AND CONVERTING DATES

One common set of tasks involves generating the ISO 8601 date and date/time values required by SDTM. Clinical data systems store dates in a variety of other forms: numeric SAS date values, separate day, month, and year fields, and a variety of different character representations. Fortunately, a given clinical study will generally store dates consistently—although lab results or other outside data may still differ from the majority of CRF data.

A simple example is from a study where the clinical data had character date variables containing date values in the form YYYYMMDD. Some of these values were truncated to year-only or year-and-month-only, and in some cases a value of “C” was used to indicate that a treatment or condition was continuing at a particular visit. The following macro was used to convert one of these character dates (usually stored in a variable ending in “DF”) into an ISO 8601 date stored in an SDTM variable:

```
* Create an ISO date from a variable-width YYYYMMDD field ;
* varDTC is the name of the variable in which to store the ISO date ;
* varDF is the name of the variable containing the variable-width date ;
%macro isoFromDF(varDTC, varDF) ;
   do ;
      * Initialize the result to missing ;
      &varDTC = '' ;
      * If the date is complete, do the conversion using SAS informat/format ;
      if length(&varDF) eq 8 then
         &varDTC = put(input(&varDF, YYMMDD8.), IS8601DA10.) ;
      * If the date is YYYYMM, insert a hyphen ;
      else if length(&varDF) eq 6 then
         &varDTC = catX('-', subStr(&varDF, 1, 4), subStr(&varDF, 5, 2)) ;
      * If the date is just a year, it is already in the correct format ;
      else if length(&varDF) eq 4 then
         &varDTC = &varDF ;
      * Warn about other unexpected values ;
      else if &varDF not in ('' 'C') then
         put 'WAR' 'NING: Unrecognized date value ' &varDF.= ;
     end;
  %mend isoFromDF ;
```

Once this macro has been defined, you can use it to process your dates. If there are other specific values that need to be handled as missing for a particular domain, you can handle those before calling the macro:

```
if AESTDF not in ('UNKNOWN' 'N/A') then %isoFromDF(AESTDTC, AESTDF)
else AESTDTC = '' ;
```

A slightly more complex case is where a character date resembles the SAS DATE9. format, but in which fields with missing day or month components have leading spaces. In this case there is both a macro and a format: the format is designed to convert three-character month abbreviations to two-digit months and the macro uses this format to handle partial dates. Again, dates of “C” are expected and should be mapped to missing values.

```
* Format to convert three-letter abbrev to two-digit character month number ;
proc format ;
   value $MONtoMM
      'JAN' = '01'
      'FEB' = '02'
      'MAR' = '03'
      'APR' = '04'
      'MAY' = '05'
      'JUN' = '06'
      'JUL' = '07'
      'AUG' = '08'
      'SEP' = '09'
      'OCT' = '10'
      'NOV' = '11'
```

```
      'DEC' = '12'
      ;
  run ;

* Create an ISO date from a variable-width DDMONYYYY field ;
* varDTC is the name of the variable in which to store the ISO date ;
* varDate is the name of the variable containing the variable-width date ;
%macro isoFromDate(varDTC, varDate) ;
   do ;
      * Initialize the result to missing ;
      &varDTC = '' ;
      * Eliminate leading spaces on the date ;
      &varDate = left(&varDate) ;
      if length(&varDate) eq 9 then
         &varDTC = put(input(&varDate, Date9.), IS8601DA10.) ;
      * Handle MONYYYY case ;
      else if length(&varDate) eq 7 then
         &varDTC = catX('-',
                        subStr(&varDate, 4, 4),
                        put(subStr(&varDate, 1, 3), $MONtoMM.)) ;
      * If the date is just a year, it is already in the correct format ;
      else if length(&varDate) eq 4 then
         &varDTC = &varDate ;
      * Warn about other unexpected values ;
      else if &varDate not in ('' 'C') then
         put 'WAR' 'NING: Unrecognized date value ' &varDate.= ;
     end;
  %mend isoFromDate ;
```

Clinical data often has dates and associated times stored in separate fields, while the SDTM specification requires combined date and time values. Generally, it is easier to handle the date first and to then append a time value if one is provided. The following macro appends time values of the form HHMM (24-hour times without colons but with leading zeroes) to already-populated date values:

```
* Add an ISO time to an ISO date/time field containing only a date part ;
* varDTC is the name of the variable containing a complete ISO date ;
* varTF is the name of the variable containing a time in the form HHMM ;
%macro isoAddTime(varDTC, varTF) ;
   do ;
      * Verify that the field already contains a complete date ;
      if length(&varDTC) ne 10 then
         put 'ERR' 'OR: Attempt to add time to incomplete date value' ;
      * If the time value is populated, insert a colon and append it ;
      * to the date, separated by a "T" ;
      else if length(&varTF) eq 4 then
          &varDTC = catX('T', &varDTC,
                         catX(':', subStr(&varTF, 1, 2), subStr(&varTF, 3, 2))) ;
      * Warn about other unexpected values ;
      else if &varTF not in ('C') then
         put 'WAR' 'NING: Unrecognized time value ' &varTF.= ;
     end;
  %mend isoAddTime ;
```

Appending a time can then be done as a second step after populating the initial date value ;

```
if AESTDF not in ('UNKNOWN' 'N/A') then do ;
   * Populate the date portion of AESTDTC ;
   %isoFromDF(AESTDTC, AESTDF)
   * If time is missing, avoid an error for a partial date ;
   if AESTTF not in ('' 'UNK') then %isoAddTime(AESTDTC, AESTTF)
  end ;
else AESTDTC = '' ;
```

USING FORMATS TO AVOID MERGING

The SDTM specification requires data to be consistent across domains. For example combinations of visit names stored in VISIT and visit numbers stored in VISITNUM should match the values stored in the Trial Visits domain TV. By defining SAS formats you can do this standardization without needing to sort and merge data sets.

Example: Visit Names

For example, the following code defines a numeric format TV_VISIT. that maps visit numbers to visit names based on the values stored in SDTM.TV. Any visit numbers not found in the TV domain will be mapped to the visit name "UNSCHEDULED".

```
proc sql ;
   * Load visit number/name combinations from TV domain ;
   create table work.TV_Visits as
   select left(put(VISITNUM, Best12.)) as Start,
          VISIT as Label,
         'TV_Visit' as fmtName,
         ' ' as HLo length=1
   from SDTM.TV
   order by Start
   ;
   * Add a record to make other values map to unscheduled ;
   insert into work.TV_Visits (Start, Label, FmtName, HLo)
      values (' ', 'UNSCHEDULED', 'TV_Visit', 'O')
   ;
  quit ;
* Create a format based on the data set ;
proc format CntlIn=work.TV_Visits ;
  run ;
```

With the TV_VISIT. format thus defined, you can use it to derive visit names that will be consistent with visit numbers. For example, if the clinical data has a visit number stored in VISIT_NO which should map directly to VISITNUM, you can populate both the visit number and the visit name as follows:

```
VISITNUM = VISIT_NO ;
VISIT = put(VISITNUM, TV_VISIT.) ;
```

Example: Reference Dates

Many SDTM domains contain a study day variable. This value is rarely collected on the CRF and must usually be derived based on a reference start date. In a particular domain, computing a study day requires the value of DM.RFSTDTC for the subject in addition to the finding or event date stored in the domain.

We can avoid the need to merge by defining a numeric informat to convert subject identifier values from USUBJID into the SAS numeric value for the subject's reference start date:

```
* Create a data set mapping USUBJID to reference date ;
proc sql ;
   create table work.Ref_Dates as
   select USUBJID as Start,
          left(put(input(scan(RFSTDTC, 1, 'T'), YYMMDD10.), 8.)) as Label,
          'Ref_Date' as FmtName,
          'I' as Type
   from SDTM.DM
   order by start ;
  quit ;
* Create an informat based on the data set ;
proc format CntlIn=work.Ref_Dates ;
  run ;
```

Now, we can compute a study day from the value of USUBJID and the numeric value of an event date. For example, in a data step where USUBJID is already populated and the numeric adverse event start date is stored in AESTDT, we can compute the adverse event start study day as follows:

```
_RefDate = input(USUBJID, Ref_Date.) ;
if nMiss(AESTDT, _RefDate) eq 0 then
   AESTDY = AESTDT - _RefDate + (AESTDT gt _RefDate) ;
```

Example: Lab Mapping

Lab data can be complicated; one common situation is that the test codes provided in the raw lab data do not match codes from the SDTM controlled term list. Some codes will need to be converted to match existing terminology, and tests for which there is no reasonable match will need to be added to the list of terminology for the study. When faced with a set of studies with a number of non-standard codes, we gathered all lab codes that appeared in a study and stored them in a column of an Excel spreadsheet. We added additional columns for LBTESTCD and LBTEST, and had someone with lab data experience map each of the codes used in the study to standard test codes and names.

In a handful of cases, the non-standard codes were used to convey additional information about the test method or the type of specimen used. While mapping the raw test codes, we also populated columns for LBSPEC and LBMETHOD values when those values were conveyed by the original test code. We could then load this Excel spreadsheet into a SAS data set WORK.ALL_TESTS with the columns Raw_Code, LBTestCD, LBTest, LBSpec, and LBMethod.

We then used this data set as the basis of a set of formats:

```
* Data set to define formats ;
proc sql ;
   create table work.Lab_Mapping as
      select Raw_Code as Start,
             LBTestCD as Label,
             '$LB_Code' as FmtName
      from work.All_Tests
   union
      select Raw_Code as Start,
             LBTest as Label,
             '$LB_Name' as FmtName
      from work.All_Tests
   union
      select Raw_Code as Start,
             LBSpec as Label,
             '$LB_Spec' as FmtName
      from work.All_Tests
   union
      select Raw_Code as Start,
             LBMethod as Label,
             '$LB_Meth' as FmtName
      from work.All_Tests
   order by FmtName, Start ;
  quit ;
* Create formats from data set ;
proc format CntlIn=work.Lab_Mapping ;
   run ;
```

Within a data step, we could then use these formats to populate values in the LB domain. In this example, the data in RAW.LAB_DATA has the non-standard code stored in the value LABCODE:

```
data work.&DOMAIN._1 ;
   %unquote(&_domainAttrib.) ;
   set raw.Lab_Data ;
   [...]
   LBTEST = put(LABCODE, $LB_Name.) ;
   if LBTEST eq LABCODE then
      put 'WAR' 'NING: Unmatched raw lab code ' LABCODE= ;
   else do ;
```

```
      LBTESTCD = put(LABCODE, $LB_Code.) ;
      LBSPEC = put(LABCODE, $LB_Spec.) ;
      LBMETHOD = put(LABCODE, $LB_Meth.) ;
    end ;
  [...]
run ;
```

This example shows that we can use the formats to populate variables in the LB domain without having to merge the raw lab data set with the lookup table. We also want to detect any lab codes that are not in the lookup table; while there are codes in the raw data that are identical to the desired terms from the controlled term list, the lab test name in LBTEST should never match the raw test code. If applying the $LB_Name format to a code returns the original code, it means that there was no record in the lookup spreadsheet for that code.

While lookup mismatches will not happen when we build the spreadsheet as described above, the code that tests for a match allows us to run our program against the same spreadsheet on a similar study, or on a new version of the lab data from the same study, and to detect any new test codes that will need to be incorporated into the mapping spreadsheet.

COMMON TASKS

For tasks that are common to many SDTM domains, you can encapsulate standardized code in a macro. Calling the macro is quicker and more reliable than re-creating the same code every time.

Checking Key Field Uniqueness

Most SDTM domain data sets have multiple records per subject. A numeric sequence variable is used to uniquely identify each record for a subject; this variable is named *xx*SEQ, where *xx* is the name of the domain. The sequence number is generally based on a particular sort order for a subject's records. The domain data set is sorted by identified key fields and *xx*SEQ values are then assigned sequentially, assuming that the list of key fields is sufficient to uniquely order the records.

While creating an SDTM data set, it is easy to check that records are uniquely identified by the key fields. The following macro will check for uniqueness and produce a data set containing any records which are not uniquely identified:

```
%macro checkKeyFieldUniqueness(dsName=) ;
   * Identify the last key field ;
   %local _lastKeyField ;
   %let _lastKeyField=%scan(&_keyFields., -1);

   * Sort the data set by the key fields ;
   proc sort data=&dsName. out=&dsName. ;
      by &_keyFields. ;
     run ;
   * Identify records that are not uniquely identified ;
   data work.&DOMAIN._Dupes ;
      set &dsName. ;
      by &_keyFields. ;
      * If the record is not uniquely identified ;
      if not (first.&_lastKeyField. and last.&_lastKeyField.) then do ;
         * Generate a log message ;
         put 'WAR' 'NING: Duplicate observation' ;
         output ;
        end ;
     run;
  %mend checkKeyFieldUniqueness ;
```

Calling this macro for a draft data set for the LB domain:

```
%checkKeyFieldUniqueness(dsName=work.&DOMAIN._Draft) ;
```

will perform a check of key field uniqueness. If the currently defined key fields are not unique, it will produce log warnings and a data set WORK.LB_Dupes which you can examine to find columns which could be added to the list of keys to uniquely identify data set rows.

Assigning Sequence Numbers

Once key values have been identified, the process of assigning values to the sequence variable is the same across domains. The following macro will populate the sequence number variable in a draft domain data set:

```
%macro assignSequence(dsName=) ;
   * Sort the data set by the key fields ;
   proc sort data=&dsName. out=&dsName. ;
      by &_keyFields. ;
     run ;
   * Assign sequence numbers ;
   data &DOMAIN._Seq ;
      set &dsName. ;
      by STUDYID USUBJID ;
      retain _count ;
      if first.USUBJID then _count=0 ;
      _count = _count + 1 ;
      &DOMAIN.SEQ = _count ;
      drop _count ;
     run;
  %mend assignSequence ;
```

Calling this macro on a draft data set for the LB domain:

```
%assignSequence(dsName=work.&DOMAIN._Draft) ;
```

will create a data set WORK.LB_Seq in which the values of LBSEQ are populated to uniquely identify records for a subject.

Creating Supplemental Qualifier Values

A supplemental qualifier data set has a vertical structure in which each value is stored in a separate row. The fields QNAM, QLABEL, and QORIG store metadata—variable name, variable label, and origin—while the actual data value is stored in the field QVAL. QVAL is never missing; if there is no data value, no row is created in the supplemental qualifier data set.

Two macros can help simplify this process. In each macro, the parameters specify a field in the data set to be stored in the supplemental qualifier data set along with the metadata for that field. Since QVAL is a character field, when the source field is numeric the macro allows an additional parameter of a format to use for conversion to character.

```
* Create a record in SUPPQUAL for a character value ;
%createSuppValC(dsField=, Name=, Label=, Origin=CRF) ;
   * Only create if value is non-missing ;
   if &dsField. ne '' then do ;
      QNam = "&Name." ;
      QLabel = "&Label." ;
      QVal = &dsField. ;
      QOrig = "&Origin." ;
      output ;
     end ;
  %mend createSuppValC ;
* Create a record in SUPPQUAL for a numeric value ;
%createSuppValN(dsField=, Name=, Label=, Format=Best12., Origin=CRF) ;
   * Only create if value is non-missing ;
   if &dsField. gt .Z then do ;
      QNam = "&Name." ;
      QLabel = "&Label." ;
      * Use the specified format and left justify ;
      QVal = left(put(&dsField., &Format.)) ;
      QOrig = "&Origin." ;
      output ;
     end ;
```

```
    %mend createSuppValN ;
```

For example, assume we have a draft data set for the exposure domain EX and we need to store two values from CRF fields in the supplemental qualifier data set: the number of capsules taken (NUM_CAPS in the draft data set) and the color of the capsules (CAP_COL). The following code creates a draft data set WORK.SUPPEX_1 using the macros above:

```
data work.SUPP&DOMAIN._1 ;
   %unquote(&_suppQualAttrib.);
   set work.EX_Draft ;

   * Assign SuppQual values for all fields ;
   RDomain = "&DOMAIN." ;
   IDVar = "&DOMAIN.SEQ" ;
   IDVarVal = left(put(&DOMAIN.SEQ, 4.) ;
   QEval = '' ;

   %createSuppValN(dsField=NUM_CAPS,
                   Name=NUM_CAPS,
                   Label=Number of Capsules,
                   Format=2.) ;

   %createSuppValN(dsField=CAP_COL,
                   Name=CCOLOR,
                   Label=Capsule Color) ;
  run;
```

Storing Final Data Sets

Another common task is storing the final domain and supplemental qualifier data sets. The following macro takes draft domain and supplemental qualifier data set names as parameters and writes permanent data sets to the library SDTM:

```
%macro saveDataSets(domainDraft=, suppDraft=) ;
   * Save domain data set ;
   data SDTM.&DOMAIN.(label="&_domainLabel.") ;
      set &domainDraft.(keep=&_domainVarList.) ;
     run ;
   %* Check whether a SuppQual data set is defined ;
   %if %eval(&_hasSuppQual.) %then %do ;
      * Save supplemental qualifier data set ;
      * SORTSEQ option makes IDVARVAL sort like a numeric ;
      proc sort data=&suppDraft.(keep=&_SuppQualVarList.)
                out=SDTM.SUPP&DOMAIN.(label='Supplemental Qualifiers')
                SortSeq=Linguistic(Numeric_Collation=On) ;
         by STUDYID RDOMAIN USUBJID IDVAR IDVARVAL QNAM ;
        run ;
     %end ;
  %mend saveDataSets ;
```

A more fully developed version of this macro might do more to check the parameters and might refuse to overwrite existing data sets if the draft data sets are empty.

CONCLUSION

Data standards increase your opportunities to develop and re-use standardized SAS code. Having your own toolkit of SDTM programming solutions will let you create data sets more quickly and reliably.

REFERENCES

Abolafia, Jeff and Frank DiIorio. 2008. “Managing The Change And Growth Of A Metadata-Based System,” SAS Institute Inc. Proceedings of SAS Global Forum 2008. Cary, NC: SAS Institute, Inc.

Abolafia, Jeff and Frank DiIorio. 2011. “Brave New World: How to Adapt to the CDISC Statistical Computing Environment,” Proceedings of the 2011 Pharmaceutical SAS Users Group Conference. Cary, NC: SAS Institute, Inc.

DiIorio, Frank and Jeff Abolafia. 2007. “The Electronic Project: Effectively Using Metadata Throughout the Project Life Cycle,” Proceedings of SAS Global Forum 2007. Cary, NC: SAS Institute, Inc.

CONTACT INFORMATION

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

David Scocca
Rho, Inc.
6330 Quadrangle Drive
Chapel Hill, NC 27517
919-595-6274
Dave_Scocca@RhoWorld.com