A Toolkit for CDISC Implementation

Robert T. Stemplinger, ICON Clinical Research, Redwood City, CA

ABSTRACT

Implementation of the CDISC standards presents a number of challenges and issues, and is a time and resource intensive process. But many of those challenges are related to the mapping of legacy or existing data structures to the SDTM domains. Once the necessary domains, variables, variable attributes, and associated controlled terminology have been properly specified, the actual programming of the structures is not radically different from the tasks most reasonably competent SAS® programmers are currently performing. Furthermore, since the SDTM domains represent a true standard structure, that is, one where variables can not be added or deleted (other than permissible variables), or their attributes changed, there is much opportunity for associated code standardization. This paper will detail areas where such standardization is possible, for the creation of SDTM compliant structures as well as their verification, and show examples of code where actual implementation has occurred.

INTRODUCTION

As the CDISC structures, particularly the SDTM domains, are defined as a true standard, there is much opportunity for code standardization and reuse. While there are advantages to acquiring or developing fully functional applications to assist in the creation of SDTM compliant structures, much can be done while evaluations or development of such products are ongoing or funds for their acquisition have not yet been appropriated. Without a vastly significant investment of time or resources, a reasonably robust toolkit for CDISC implementation can be developed.

SCOPE

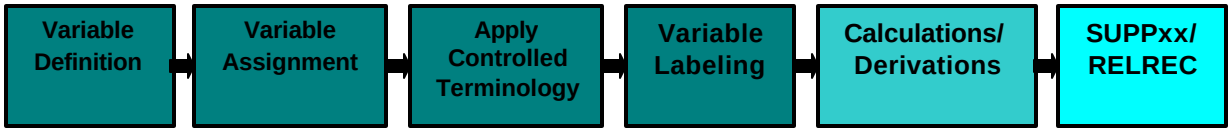
Of the models that comprise the CDISC standard, the most relevant for legacy data conversion are the Study Data Tabulation Model (SDTM), the Analysis Dataset Model (ADaM), and the Case Report Tabulation Data Definition Specification (CRT-DDS). Included with the SDTM domains would be the necessary trial design data sets, supplemental qualifier data sets, and the related records data set (if dictated via natural relationships in the data). For the purposes of this paper, only the SDTM structures are considered.

OVERVIEW OF CDISC IMPLEMENTATION

Implementation of the CDISC standards at ICON is a four step process. Step 1 entails annotating the CRF with the associated SDTM domain and variable names. This is the actual "mapping" of the legacy or existing data to the SDTM domains, and is often the most difficult step in terms of working with the CDISC structures. Step 2 is the creation of the SDTM specification. This specification details all of the domains required to properly model the legacy or existing data, and may include user defined domains as well as the standards, the associated variables and variable attributes, and the controlled terminology. In the absence of CDISC published controlled terminology, variable values are standardized upon close consultation with the sponsor. Step 3 is the actual programming of the SDTM structures and their verification. Step 4 is the creation of the CRT-DDS (define.xml). Step 3 is the area of focus for this paper.

TOOLS FOR STANDARDIZATION/AUTOMATION

Before discussing areas for standardization directly related to the structure of the SDTM data sets, it is worthwhile outlining a "standardized" or general program flow for the creation of each domain. A sample "standard" program flow is shown below.



In keeping with standard practice for code efficiency, the colored boxes represent single data steps. It is possible to define each domain, its variables and variable attributes, assign values and control terminology in a single data step. Calculated variables are defined and initialized to missing in that same data step for later processing. The benefits of such an approach are that variable metadata definition occurs only once, at the beginning of domain creation, which facilitates code readability, maintenance, and modification. Raw data variables needed for calculations and/or derivations, or for supplemental qualifier and/or RELREC data sets, are kept and propagated to subsequent data steps for later processing.

Of course, not every domain for every implementation lends it self to this flow, but it has served well as a general standard in over 100 study conversions comprised of over 2,000 data sets. From this general flow, much of the programming can be standardized and automated. For instance a few lines of code at the beginning of every data step can define the variables for each domain. For example, the following statements define the variables need for the CM and SUPPCM domains:

```
data cm(keep=studyid domain usubjid cmseq cmspid cmtrt cmdecod cmcat cmindc
            cmclas cmclascd cmdose cmdostxt cmdosu cmdosfrm cmdosfrq cmdosfrq cmroute
            cmstdtc cmendtc cmstdy cmendy cmstrf cmenrf
            cont pcaitem);
    length studyid $ 7 domain $ 2 usubjid $ 13 cmseq 8 cmspid $ 5 cmtrt $ 60 cmdecod $ 60
           cmcat $ 40 cmindc $ 100 cmclas $ 60 cmclascd $ 5 cmdose 8 cmdostxt $ 40
           cmdosu $ 5 cmdosfrm $ 25 cmdosfrq $12 cmroute $ 24 cmstdtc $ 19 cmendtc $ 19
           cmstdy cmendy 8 cmstrf $12 cmenrf $ 12;
```

Note that the italicized, bolded variables are not part of the CM SDTM domain, but are needed later for the creation of SUPPCM, so they are kept. Examples of variable assignments range from constant assignment to value formatting. For example:

```
studyid="xxxx-yy";
domain="CM";
usubjid="xxxxxx-yy-"||put(ptid,z3.);
cmseq=.;
cmspid=left(mdnum);
if (xcm) then cmcat="CONCOMITANT MEDICATIONS";

sex=upcase(put(sexn,sex.));
race=upcase(put(racen,race.));
```

Here note that the italicized, bolded variable is defined and initialized, but not used until a subsequent data step. Of increasing importance in the creation of SDTM domains is controlled terminology. Applying the terminology can occur in the initial data step as well. For example, the following code excerpt can be used to control terminology for CMDOSU in the CM domain:

```
if (compress(mdunit) in ("MG","MGS")) then cmdosu="mg";
else if (compress(mdunit) in ("G","GM")) then cmdosu="g";
else if (compress(mdunit) in ("I.U.","IU")) then cmdosu="IU";
else if (compress(mdunit) in ("ML","MLS","CC")) then cmdosu="ml";
else if (compress(mdunit) in ("UNIT","UNITS")) then cmdosu="unit";
```

The final step to complete variable attribute definition is to label the variables. Using this approach, most of the domain is programmed. A subsequent step is used to merge in the reference date range and to derive –DY variables, --RF flags, and to assign the –SEQ number for all records in the domain.

So having standardized the approach, then, it is worth considering possibilities for automation. CDISC has published domain metadata in MS Excel format. After careful review and modification of these spreadsheets, they can be imported and used to automate the definition of metadata for each domain in a zero observation data set. The importation of such information is trivial, and could be as simple as executing the code shown below:

```
%macro import(dsn=);
    proc import datafile="&dsn..csv" out=&dsn dbms="dlm" replace;
        delimiter=',';
        getnames=no;
    run;

    data sdtmin.sdtm&dsn(drop=var1 var2 var3 var4 var9 var10 var11 var13 var14 var15
                         rename=(var5=name var6=sdtmlbl var7=sdtmtyp var8=sdtmct var12=sdtmcre));
        length seq 8.;
        set &dsn end=lastobs;
        if (_n_ > 1);
        seq=var1;
        var5=upcase(var5);
        var8=left(tranwrd(var8,"*",""));
        if (var5 ne " ");
    run;
%mend;
%import(dsn=ae);
```

Once the metadata exists as a zero observation data set, it can be used as a shell for initial domain definition.

While there are real benefits to creating machine readable specifications that define transforms, derivations, and assignments, that is beyond the scope of this paper, and leads to development of more complex CDISC tools. Further possibilities for automation do present for variable assignment and application of controlled terminology, however, which facilitate the

development of conversion programming. For instance, consider that all date/time values in SDTM domains must comply with the ISO 8601 standard. It is possible to automate the creation of date/time values in this format with relative ease, using an approach of concatenating the individual date/time pieces required by the ISO 8601 format. Such an approach allows for the handling of partial dates. For example, the code below will parse and format a date value in ISO 8601 format. Note that two assumptions are made by the code: (1) missing month or day values are identified by '00', and (2) year values of '00' are assign to the year 2000. Consider the code below:

```
%if (&datevar ne  ) %then %do;

    /** INITIALIZE DATE COMPONENTS. **/
    length mm dd yy $2 yyyy $4 &d8601var $ 19;
    mm=" ";
    dd=" ";
    yy=" ";
    yyyy=" ";

    /** ASSIGN DATE COMPONENTS. **/
    mm=substr(&datevar,1,2);
    dd=substr(&datevar,4,2);
    yy=substr(&datevar,7,2);

    /** SET UNKNOWN MONTH AND DAY COMPONENTS TO MISSING. **/
    if (mm="00") then mm=" ";
    if (dd="00") then dd=" ";

    /** IF THE YEAR IS LESS THAN OR EQUAL TO THE CURRENT YEAR ASSUME 21ST CENTURY AND **/
    /** PREFIX YEAR WITH A '20'. IF THE YEAR IS GREATER THAN THE CURRENT YEAR ASSUME  **/
    /** THE 20TH AND PREFIX WITH A 19.                                                **/
    if (yy ge 0 and yy le substr("&sysdate",6,2)) then yyyy="20"||yy;
    if (yy gt substr("&sysdate",6,2)) then yyyy="19"||yy;

    /** CONCATENATE DATE COMPONENTS TOGETHER WITH DASHES THEN DELETE THE DASHES AT THE END. **/
    if (mm ne "" or dd ne "" or yyyy ne "") then do;
        &d8601var=yyyy||"-"||mm||"-"||dd;
        &d8601var=TRANWRD(&d8601var,"-  ","");
        &d8601var=TRANWRD(&d8601var,"-*","");
    end;
    drop mm dd yy yyyy;
%end;
```

As mentioned above, an area of growing importance, and complexity, is controlled terminology. As the SDTM structures have stabilized in terms of their development, attention is now starting to shift toward standardizing content. To date, CDISC has published three terminology packages. Application of this terminology can be painstaking and complex, particularly for projects consisting of many studies with largely varying data sets. But just as it was possible to import the domain metadata, it is also possible to import the controlled terminology. Once imported and usable as a data set, at the very least the identification of controlled terminology violations is readily automated.

It is also possible to automate the creation of entire data sets. The supplemental qualifiers data set has the same structure regardless of the content it is intended to hold. Instead of importing metadata and augmenting contents with subsequent programming, it makes more sense with supplemental qualifiers to create the data set in one step. For example, the following code can be used to create a supplemental qualifiers data set for the CM domain:

```
data sasdata.suppcm(label="Supplemental Qualifiers - CM"
                    keep=studyid rdomain usubjid idvar idvarval qnam qlabel qval qorig);
    length studyid $ 7 rdomain $ 2 usubjid $ 13 idvar $ 5 idvarval $ 8 qnam $ 8 qlabel $ 100
           qval $ 200 qorig $ 40 qeval $ 40;
    set cm;
    by usubjid;

    rdomain="CM";
    idvar="CMSEQ";
    idvarval=put(cmseq,8.);
    qeval=" ";
    qorig="CRF";

    qnam="CONT";
    qlabel="CONTINUING";
    qval=upcase(left(cont));
    if (qval not in (" ", ".")) then output;

    qnam="PRIOR";
    qlabel="PRIOR";
```

```
    qval=upcase(left(prior));
    if (qval not in (" ", ".")) then output;
run;
```

Note the italicized sections are simply repeats of the same code for a different variable. This type of code repetition lends itself to macro development. The code above can be replaced by a simple macro:

```
%macro supp(domain=,dsn=,idvar=,varstr=,vrlblstr=,qorigstr=,qevalstr=);
    data sasdata.supp&dsn(label="Supplemental Qualifiers - &domain"
                        keep=studyid rdomain usubjid idvar idvarval qnam qlabel qval qorig);
    length studyid $ 7 rdomain $ 2 usubjid $ 13 idvar $ 5 idvarval $ 8 qnam $ 8 qlabel $ 100
           qval $ 200 qorig $ 40 qeval $ 40;
    set cm;
    by usubjid;

    %let varidx=1;
    %do %while(%scan(&varstr,&varidx,' ') ne ' ');
        %let var=%scan(&varstr,&varidx,' ');
        %let varlbl=%scan(&vrlblstr,&varidx,' ');
        %let qorig=%scan(&qorigstr,&varidx,' ');
        %let qeval=%scan(&qevalstr,&varidx,' ');

        rdomain=upcase("&domain");
        idvar=upcase("&idvar";
        idvarval=put(&idvar,8.);

        qnam=upcase("&var");
        qlabel=upcase("&varlbl";
        qval=upcase(left(&var));
        %if (&qorigstr ne  ) %then %do;
            qorig="&qorig";
        %end;
        %if (&qevalstr ne  ) %then %do;
            qeval="&qeval";
        %end;
        if (qval not in (" ", ".")) then output;

        %let rowidx=%eval(&rowidx+1);
    %end;
run;
```

And the SUPPCM domain can be created by a simple macro call:

```
%supp(domain=cm,dsn=cm,idvar=cmseq,varstr=cont prior,vrlblstr=CONTINUNING PRIOR,
      qorigstr=CRF CRF,qevalstr=);
```

The macro would define the variable attributes, assign the values, etc.

TOOLS FOR VALIDATION/VERIFICATION

Perhaps the most beneficial area for toolkit development is in the area of SDTM data set structure and content verification. Using relatively simple techniques, the published SDTM domain structures can be imported and used to verify those that have been programmed. Similar techniques as those discussed below can be used to verify the consistency of data sets for projects made up of multiple studies. This harmonization effort is greatly enhanced by electronic integrity checks that verify that each domain has been programmed consistently across all studies within the project.

For each study, there are two basic types of checks that need to be performed, data set level and variable level. The first step is to assemble a list of all domains that need to be verified. This can be done using PROC CONTENTS as follows:

```
proc contents data=&datlib._all_ out=alldsn(keep=memname memlabel) noprint;
run;

proc sort data=alldsn nodupkey;
    by memname;
run;
```

Since all supplemental qualifier data sets must have the same structure, and there is only one SUPPQUAL structure imported from the CDISC published domain metadata, a simple IF-THEN-ELSE construct can reassign the zero observation data set to be used for checking:

```
if ("&dsn" in ("SUPPAE","SUPPCM","SUPPCO","SUPPDM","SUPPDS","SUPPDV","SUPPEG","SUPPEX",
               "SUPPIE","SUPPLB","SUPPMH","SUPPPE","SUPPQS","SUPPSC","SUPPSG","SUPPSU",
```

```
                "SUPPVS","SUPPDI","SUPPML","SUPPOA","SUPPOC","SUPPPC","SUPPPP")) then do;
        call symput('indsn','suppqual');
    end;
    else do;
        call symput('indsn',"&dsn");
    end;
```

Once this has been done, the checks can be executed. Examples of the types of checks run are: verification of data set labels, verification that no illegal variables are present on the data set, verification that all required variables are present and that there are no null values, verification that expected variables are present, verification of variable types and labels, verification of variable order, verification of ISO 8601 date/time format, verification that –TESTCD variables are not greater than 8 characters in length, verification that –TEST variables are not greater than 40 characters in length (except for IETEST which is limited to 200 characters), verification that COVAL, COLVAL1 … COVALn are not greater than 200 characters in length, flagging of permissible variables with all values null, flagging of controlled terminology violations, and an assortment of suspicious value checking. The primary construct around which all check programming is based is a looping mechanism, of which there are two varieties. The first cycles through a list of data sets, passing the data set name and label to the check macro:

```
%macro loopdsn;
    %let maxdsn=0;
    data _null_;
        set stdydsn end=lastobs;
        if (lastobs) then call symput('maxdsn',put(_n_,best.));
    run;

    %do dsnidx=1 %to &maxdsn;
        data _null_;
            set stdydsn;
            if (_n_=&dsnidx) then do;
                call symput('dsnnm',left(trim(memname)));
                call symput('dslb',left(trim(memlabel)));
            end;
        run;
        %chk(dsn=&dsnnm,dsl=&dslb);
    %end;
%mend;
%loopdsn;
```

As each data set is passed through the macro, the variables on the data set are also listed and cycled through, and the desired check is executed. For instance, this code flags –TESTCD values greater than 8 characters:

```
%macro chkcd(var=);
    data chk5i;
        set &datlib..&dsn;
        length dslbl $ 6 name $ 9 value flag $ 200;
        dslbl="&updsn";
        name="&nm";
        value=&var;

        if (length(&nm)>8) then
          flag="--TESTCD variable value greater than 8 characters in length.";
    run;

    proc sort data=chk5i (keep=dslbl name value flag) nodupkey;
        by name value;
        where (flag ne " ");
    run;
%mend;
```

The second loops through individual variable names. The code below, for instance, loops through all the variable names in a data set. Variable names containing the string ‘TESTCD’ are stored in a macro variable, and then a call to the actual check macro shown above is made.

```
%do idx=1 %to &maxobs;
    data _null_;
        set sdtm&indsn;
        if (_n_=&idx) then do;
            if (substr(name,3,6)="TESTCD") then call symput('trig','1');
            else call symput('trig','0');
            call symput('nm',left(trim(name)));
        end;
    run;
```

```
    %if ("&trig"="1") %then %do;
        %chkcd(var=&nm);
         data chk5;
            set chk5 chk5i;
        run;
    %end;
%end;
```

This code is a slight, more simple variation of that shown above, looping through the variables in a data set and passing them through the CDISC published terminology to flag violations.

```
%do idx=1 %to &maxobs;
    data _null_;
        set sdtm&indsn;
        if (_n_=&idx) then do;
            call symput('nm',left(trim(name)));
        end;
    run;
    %chkterm(var=&nm);

    data chk10;
        set chk10 chk10i;
    run;
%end;
```

Finally, once all checks have executed for all domains, output is collected and PROC REPORT is used to generate a report in .rtf format for review.

CONCLUSION

With a minimal investment of time and resources, a fairly robust set of tools can be programmed in SAS that will greatly facilitate the programming and verification of SDTM compliant structures. As programming experience and expertise grows within the organization, so too will the toolkit. This investment proves to be well worth whatever time is spent, as time and resource is actually conserved programming subsequent CDISC implementations.

CONTACT INFORMATION

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

Robert T. Stemplinger
ICON Clinical Research
555 Twin Dolphin Drive, Suite 400
Redwood City, CA 94065
Phone: (650) 620-2165
Fax: (650) 591-0611
Email: stemplingerr@iconus.com