PharmaSUG 2023 - Paper MM-273

Better CDISC Standards with Metadata Programming

Sunil Gupta, Senior CDISC Consultant, Trainer and Author, Gupta Programming
Abhishek Dabral, Director, Clinical Programming, Alkermes Inc.

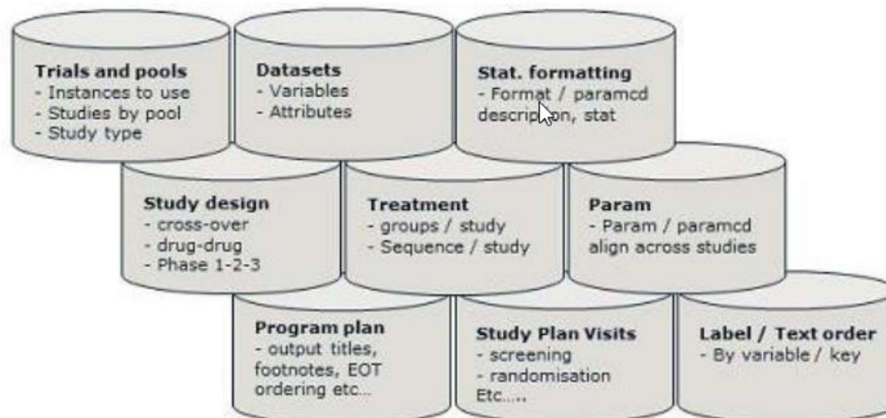
ABSTRACT

Is metadata programming ready for prime time? Are you ready to leverage CDISC standards seriously and minimize coding? Are you ready to take advantage of metadata programming methods to enforce automation, standardize workflow, increase efficiencies and ensure higher quality control in addition to reduced resources and time?

This paper shows several key applications as well as techniques for metadata programming. We will explore beyond the basic dictionary datasets and variable attributes (SAS® dictionary.columns and dictionary.tables) to more advanced programming methods of creating macro lists, looping through each item as well as code generation. These components create a robust global macro utility to be used across multiple studies as well as establish a foundation for corporate global standards. Tools that leverage metadata programming opens a world of applying regulatory rules to standardize raw data (SDTM transformation), data exploration using a single-source of truth view (eg tables from CDISC library) as well as checking for CDISC conformance (SDTM/ADaM rules).

INTRODUCTION

While SAS has built-in metadata by accessing datasets, programs and files, it is more useful to create customized metadata to better control and automate work flows. Below is an example of the metadata categories with example attributes. For each attribute, metadata enables defaults and a method to replace defaults with machine readable metadata files as inputs. This process is very similar to running SAS macros with metadata as macro parameters. Similar to define.xml metadata file, hierarchy rules with parent and child relationships can be enforced and utilized. This helps to define relationships and lookup tables between datasets.



When developing metadata programming, it is important to review macro goals of indirect reference, comprehensive scope, lists, substitutions, attributes and data value. When utilizing metadata and macro programming, the process generates SAS code that is data drive and specific for the input dataset, variables and values. Metadata macro programming can be designed for SDTM compliance. For example, information by domain can be extracted: # obs, # pts, required domains, required variable order, one or multiple key vars, dup all vars, study day < 0, and missing values. Selected examples will be reviewed to show SAS programming techniques.

Metadata and Macro Programming Examples

- Dataset Specifications from Excel file
- Proc SQL Dictionary Tables to get Dataset Attributes
- Cross-referencing Datasets
- Defensive Programming for Variable Type Specific Syntax
- Identifying Special Characters
- Standardizing Raw Data Values
- Loop through one macro variable with a list of values
- Standardize derivation of ISODATE (--DTC) variables
- Dynamically Executing SAS code
- System Environment Clean-up

Dataset Specifications from Excel file

Best practices for creating SDTMs is to access SDTM IG excel file (often used as 'specifications') and convert it to a dataset to be used as metadata of SDTM attributes. Generally Proc Import or libname with excel engine is used to import excel files. The excel file process is more robust and allows for easy maintenance to updated SDTM IG versions. Inherent in this process is the requirement to validate excel file values before using them for code generation.

```
%macro attrib(specin= , adsin= );

* List of all alpha characters;
%let alpha=ABCDEFGHIJKLMNOPQRSTUVWXYZ;

**** Read in Excel file/Dataset of specs ****;
data specin ;
   set &specin. ;
   id = _n_;
run;

proc sort;
 by id;
run;

**** Check spec variable length content type and value  ****;
data _null_;
  set specin ;
   if indexc(upcase(length), "&alpha") > 0 then put "Var_" "LENGTH: is
CDISC definition in SPECS: " variable=  length= ;
   if upcase(type)='NUM' and (''< length < '3' or length > '8') then put
"Var_" "LENGTH: for Numberic Variable is out of rage: " variable= length= ;
run;

**** Defined ISO length to maximum 19 if any in specs ****;
data specin ;
   set specin ;
   if indexc(upcase(length), "ISO") > 0 then length='19';
run;

**** Generate domain macro variables with suffix for each record in spec
file - varnam#, vartyp#, varlen#, varlbl# ****;

data _null_;
  set specin  end=eof nobs=numb;
    call symput ('varnam'||trim(left(put(_n_,3.))), trim(variable));
```

```
     call symput ('vartyp'||trim(left(put(_n_,3.))), trim(type));
     call symput ('varlen'||trim(left(put(_n_,3.))), trim(left(length)));
     call symput ('varlbl'||trim(left(put(_n_,3.))),trim(label));

     if eof then call symput ('attrnum', trim(left(put(numb, 3.))));

run;

* Create temp shell data set with attributes and one all missing record;
data &adsin.;
   format _all_;
   informat _all_;

Loop through all records and macro variables to create attribute
statements;
   %do i=1 %to &attrnum. ;
      %if %upcase(&&vartyp&i..) = CHAR %then %do;
         %if &&varlen&i.. ne %then %do;
             attrib &&varnam&i.. length=$&&varlen&i.. label="&&varlbl&i..";
                &&varnam&i = '';
         %end;
         %else %do;
             attrib &&varnam&i.. length=$125 label="&&varlbl&i..";
                &&varnam&i = '';
         %end;
      %end;
      %else %if %upcase(&&vartyp&i..) = NUM %then %do;
         %if &&varlen&i.. ne %then %do;
             attrib &&varnam&i.. length=&&varlen&i.. label="&&varlbl&i..";
                &&varnam&i = .;
         %end;
         %else %do;
             attrib &&varnam&i.. length=8 label="&&varlbl&i..";
                &&varnam&i = .;
         %end;
      %end;
   %end;

 output;
run;
%mend attrib;
```

* Notice the use of && in the macro variable names for correct macro resolution.
** Note that the Protocol Model (PRM) is useful to create trial summary domains.

In addition, this method can be extended to create raw to SDTM mapping specifications to include raw variable name, SDTM variable name, subset conditions and mapping method. This then becomes more data-driven method to build the mapping syntax needed for each SDTM variable. Below is an example of a raw to SDTM mapping specification file.

Obs	Domain	Variable	Source_ domain	Source_var	mapping	group
1	AE	USUBJID	AE	subjid_raw_char	USUBJID = subjid_raw_char;	1
2	AE	AEHLTCD	AE	hltc_num	AEHLTCD = hltc_num;	1
3	AE	USUBJID	AECD	subjid_raw_char	USUBJID = subjid_raw_char;	2
4	AE	USUBJID	SFAE	subjid_raw_char	USUBJID = subjid_raw_char;	3
5	AE	AEHLTCD	SFAE	hltc_char	AEHLTCD = input(hltc_char,best32.);	3
6	AE	USUBJID	SFAECD	subjid_raw_char	USUBJID = subjid_raw_char;	4
7	AE	USUBJID	SFAESS	subjid_raw_char	USUBJID = subjid_raw_char;	5
8	AE	AEHLTCD	SFAESS	hltc_char	AEHLTCD = input(hltc_char,hltc_c_i.);	5

Table 4.2: SAS data set of Excel file converted to data set mappings.

PROC SQL DICTIONARY TABLES TO GET DATASET ATTRIBUTES

Once the libname is define, proc sql can be used to create a lookup table of all raw datasets, variable names, types, length, label and formats. This table is common and can then be cross-referenced as needed by dataset or variable name for macro processing. Once SDTMs are created, then another similar lookup table can be created for cross-referencing SDTM attributes. As a general rule, convert working SAS model program to macro variables and logic to minimize macro programming debugging time.

```
PROC SQL;
 CREATE TABLE RAW_VARS AS
 SELECT UNIQUE libname, memname, name, type, length, label, format
 FROM SASHELP.VCOLUMN
 WHERE upcase(libname) = 'RAW';
QUIT;
```

The table below shows the SASHELP views that can be accessed for all types and levels of control in the SAS session.

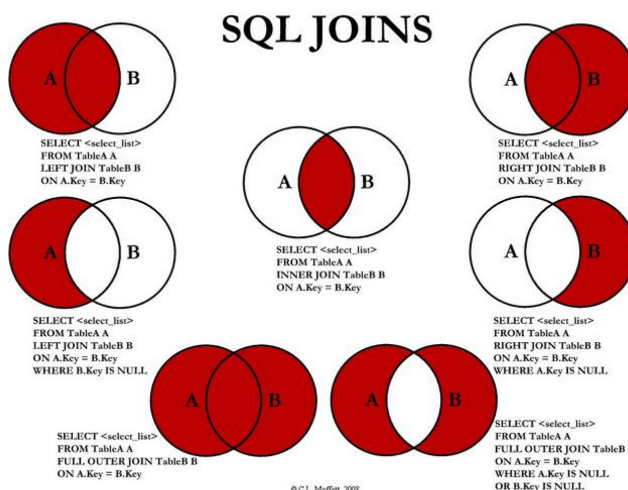
Table 1. Organized List of DICTIONARY Tables and SASHELP Views

Category	DESCRIPTION	DICTIONARY Table	SASHELP VIEWS*
SAS Session	SAS Options	OPTIONS**	VOPTION
	SAS/Graph Options	GOPTIONS**	VGOPT
	TITLE Statements	TITLES	VTITLE
	Defined macro variables	MACROS	VMACRO
	LIBNAME Information	LIBNAMES	VLIBNAM
	Available Engines	ENGINES	VENGINE
	Files defined in FILENAME statements, or implicitly	EXTFILES	VEXTFL
Members	Tables, Catalogs, and Views	MEMBERS	VMEMBER
Data Set and Related Metadata	Table and Table-Specific Information	TABLES	VTABLE
	View and View-Specific Information	VIEWS	VVIEW
	Columns from every table	COLUMNS	VCOLUMN
	DICTIONARY tables and their columns	DICTIONARIES	VDCTNRY
	Indexes	INDEXES	VINDEX
Catalogs and Related Metadata	Catalog and Catalog-Specific Information	CATALOGS	VCATALG
	Available Formats	FORMATS***	VFORMAT
	Styles	STYLES	VSTYLE
Constraints	Check Constraints	CHECK_CONSTRAINTS	VCHKCON
	Referential Constraints	REFERENTIAL_CONSTRAINTS	VREFCON
	Table Constraints	TABLE_CONSTRAINTS	VTABCON
	Constraint table usage	CONSTRAINT_TABLE_USAGE	VCNTABU
	Constraint column usage	CONSTRAINT_COLUMN_USAGE	VCNCOLU
Other	Remember information	REMEMBER****	VREMEMB

CROSS-REFERENCING DATASETS

In general, the purpose of cross-referencing datasets is to identify and process matches and non-matches. Inner joins are examples of exact matches. Full outer joins and except joins are examples of non-matches. Except joins identify all records from the left dataset that do not match records in the right dataset. This is different from the full outer join which identify records that do not match between left and right datasets. Cross-referencing datasets enables SAS developers to confirm assumptions as well as loop through variables and records for dynamically creating SAS syntax.

Below diagram shows the many common Proc SQL joins.



Below is an example of Proc SQL except join to identify ae_new records that do not match ae_old records.

```
proc sql;
create table differ as
select columna, columnb
```

```
 from PRODLIB.ae_new
except
select columna, columnb
 from PRODLIB.ae_old;
```

DEFENSIVE PROGRAMMING FOR VARIABLE TYPE SPECIFIC SYNTAX

Smarter SAS developers leverage Datastep functions or metadata variable functions to better monitor variable types and length for example. Being proactive to assure variable types are character or numeric help to prevent macro errors. By inputting the variable name into these metadata variable functions, SAS programmers have access to great data for controlling process flow and tasks.

Dictionary/Array Reference Functions	Result See SAS Paper, SAS Paper 2
CALL LABEL ROUTINE	Assign label to variable
VLABEL()	Returns the variable's label Example, Y=VLABEL(X); or Y=VARLABEL(X);
VLENGTH()	Returns the length of the string or numeric value from a character or numeric variable
VNAME()	Returns the variable's name, argument is often arrary index reference See also INDEX("&VAR", 'END') > 0 to compare variable names
VTYPE()	Returns the variable's type (N, C)
VVALUE()	Returns the formatted value

Below is a list of useful SAS SCL functions that extends the power of programming options for better macro flow designs.

Function name	Function action, Example
EXIST	Verifies the existence of a SAS member (dataset, catalog), returning a 1 or 0: `%if %sysfunc(exist(work.sae)) %then…`
OPEN	Opens a SAS dataset and returns a value (id). Many of the subsequent functions in this table use the id as an argument (and not the dataset name): `%let dsid = %sysfunc(open(crt.d_ae));`
CLOSE	Closes the dataset given by the id, and returns a value (0 if successful). Any dataset opened with the OPEN function should be closed with the CLOSE function: `%let rc = %sysfunc(close(&dsid));`
DSNAME	Returns the dataset name associated with a dataset id: `%let dsname = %sysfunc(dsname(&dsid));`
ATTRC	Returns the value of a character attribute for the dataset associated with the id: `%let att = %sysfunc(attrc(&dsid,attrib));` where *attrib* is (amongst others) sortedby - sort order if dataset is sorted (otherwise blank) label - dataset label mem - dataset name
ATTRN	Returns the value of a numeric attribute for the dataset associated with the id: `%let att = %sysfunc(attrn(&dsid,attrib));` where *attrib* is (amongst others) nobs - number of observations nvars - number of variables crdte - date created (SAS date time format)

IDENTIFYING SPECIAL CHARACTERS

Special character functions can cause havoc in SAS macros. It is best practices to identify and remove special characters if they are not expected. In addition, unmatching quotes can also cause SAS errors, warnings and notes. Generally, because manual values are entered in excel files, there is potential for accidental entry of special characters or extra quotes.

```
** Macro quoting function nrstr() keeps from resolving &thisds (masks the
&) **;
proc sql noprint;
select '%nrstr(&thisds)=' || cats(memname) || ' or ' into :&var separated
by ' ' from dictionary.columns where libname="&lib" and name="&var"
&where;
;
quit;
```

Below are examples of SAS code to remove special characters.

Remove non-printable characters such as carriage returns. In the last parameter of the below function, ‘k’ means to keep and ‘w’ means writeable and ‘i’ means to ignore case. Notice that there are three parameters in the compress() function.

```
comment = compress(strip(Comment), , 'kw');
```

Remove carriage return (‘OD’x) and line feed (‘OA’x) hidden characters.

```
comment = compress(comment, ’0D0A’x);
```

Below is a useful table to identify consistency in values within variables. They help to ensure ‘clean’ values.

ANYALNUM	Search for alphanumeric characters
ANYALPHA	Search for alphabetical characters
ANYCNTRL	Search for control characters
ANYDIGIT	Search for digit characters
ANYFIRST	Search for characters that are valid for being the first character of a SAS variable name
ANYGRAPH	Search for graphic characters
ANYLOWER	Search for lowercase letters
ANYNAME	Search for characters that are valid in a SAS variable name
ANYPRINT	Search for printable characters
ANYPUNCT	Search for punctuation characters
ANYSPACE	Search for white-space character, which include blank, horizontal and vertical tab, carriage return, line feed, and form feed
ANYUPPER	Search for uppercase letters
ANYXDIGIT	Search for hexadecimal characters

STANDARDIZING RAW DATA VALUES

Proc format is ideal for standardizing raw data values. Either from proc format or a format catalog, which can be created from an excel file or dataset, raw values can be directly mapped to standard values. This method is a best practices since the PUT() makes the conversion easy without having to insert a series of if then logic. In this simple example, F is converted to ‘Female’ and ‘M’ is converted to ‘Male’. This model can be expanded for most all character or numeric raw to standard values. The format catalog and PUT() functions are easy to maintain for one or multiple studies.

```
proc format;
```

```
 value $sex 'F' = 'Female'
                   'M' = 'Male';
run

data dm;
 set demog;
 sex = put(raw_sex, $sex.);
run;
```

Loop through one macro variable with a list of values

For the previous raw to SDTM mapping specifications, the loop is created to process and apply the mapping specifications. Each row contains all of the SAS syntax needed to dynamically map raw to SDTM variables without any hard coding of SAS statements. Separate macro variables exist for source data, rename variables, copy, transformations and dropping variables. Each SAS statement needs to valid syntax. Where needed, non-missing values should be checked. Towards the end of the data step, only the SDTM variables should be kept. For ADaMs, a basic data structure needs to be created.

```
%*** Loop through each extract to bring in the data. ***;
%do &__z_i= 1 %to &number_extracts;

 %*** Derive source data ***;
 %*** Derive rename variables ***;
 %*** Derive variable to the identify the input source data ***;
 %*** Derive copy variables ***;
 %*** Derive variables format transformations ***;
 %*** Derive drop variables ***;

%*** Extract ***;
 data __z_i_source_der_&__z_i;
  attrib &__z_i_source_dom_var_name length=$50 label='source domain';
 %*** Derive source data and rename variables ***;
 set &inlib..&__z_i_source_name_in.&__z_i_source_var_rename;

%*** Identify the input source data name ***;
 &__z_i_source_dom_var_name = "&__z_i_source_name_in";

 %*** copy variables ***;
 &__z_i_source_var_copy_syntax;

 %*** variables format transformations ***;
 &__z_i_derive_var_syntax;

 %*** drop variables ***;
 &__z_i_drop_var_syntax;
 run;
%end;
```

Below is an example of looping through a list of values in one macro variable. Proc sql creates the macro variable list and the do loop cycles through each centre value in Proc print.

```
proc sql noprint;
 select distinct centre  into :cens separated by ' '
 from derived.demo  where highrec = 1;
 quit;

 %do i = 1 %to &sqlobs;
```

```
%let cen = %scan(&cens, &i);
proc print data = rand;
 where centre = &cen;
 title1 "Randomization listing for centre &cen";
run;
%end;
```

DERIVATION OF ISO DATES (--DTC) VARIABLES

Depending on the type of source data, date/time values may contain complete, partial, or invalid dates. Many issues arise when dealing with such date types and it usually requires manipulation of these date/time values before being used in any calculation. Macro programming utilizing fcmp function can be used to derive DTC variable from date and time variables

```
/*-----------------------------------------------------------------------
Description
1) dtc_from_dttm: FCMP function to derive DTC variable from date and time
variables
2) dtc_from_dt: FCMP function to derive DTC variable from date variable
3) day_from_dtc: FCMP function to derive Study Day from Date part of DTC
variable
 ----------------------------------------------------------------------*/

proc fcmp outlib=work.funcs.dtc_from_dttm;
 function dtc_from_dttm( datvar$, timvar$ ) $;
 attrib _return_value1 length=$50;
 _return_value1 = '';
 if not missing(datvar) then do;
   if compress(scan(upcase(datvar),1,'/')) in ( '' 'UNK')  then
   _return_value1 = '-';
   else _return_value1 = compress(scan(datvar,1,'/'));

    if compress(scan(upcase(datvar),2,'/')) in ( '' 'UNK')  then
   _return_value1 = compress(_return_value1) || '--';
    else _return_value1 = compress(_return_value1) || '-'
   ||compress(scan(datvar,2,'/'));
    if compress(scan(upcase(datvar),3,'/')) in ( '' 'UNK') then
   _return_value1 = compress(_return_value1) || '--';
   else _return_value1 = compress(_return_value1)|| '-' ||
   compress(scan(datvar,3,'/'));
    end;

    if not missing(timvar) then do;
       if missing(_return_value1) then _return_value1 = '-----';
          if compress(scan(upcase(timvar),1,':')) in ( '' 'UNK') then
          _return_value1 = compress(_return_value1) || 'T-';
       else _return_value1 = compress(_return_value1) || 'T' ||
       compress(scan(timvar,1,':'));
    if compress(scan(upcase(timvar),2,':')) in ( '' 'UNK') then
   _return_value1 = compress(_return_value1) || ':-';
   else _return_value1 = compress(_return_value1) || ':' ||
   compress(scan(timvar,2,':'));
    if compress(scan(upcase(timvar),3,':')) not in ( '' 'UNK') then
   _return_value1 = compress(_return_value1) || ':' ||
   compress(scan(timvar,3,':'));
    end;
```

```
            *** Code to remove trailing '-' or ':', in case of mising
values at the end of datetime value;
        if not missing(_return_value1) then
        _return_value1=substr(_return_value1, 1 , length(_return_value1)-
        indexc(compress(reverse(_return_value1)),'1234567890')+1);
        if anyalpha(tranwrd(tranwrd(upcase(datvar),'UNK',''),'/','')) GT 0
        then
           put "CP WARNING: Note from dtc_from_dttm standard FCMP function,
        Date variable has an unexpected value: " datvar ;
     /*
      Return value
    */
       return(_return_value1);
      endsub;

run;
 proc fcmp outlib=work.funcs.dtc_from_dt;
 function dtc_from_dt( datvar$) $;
 attrib _return_value2 length=$50;
         _return_value2 = '';
         if not missing(datvar) then do;
                if compress(scan(upcase(datvar),1,'/')) in ( '' 'UNK') then
         _return_value2 = '-';
         else _return_value2 = compress(scan(datvar,1,'/'));
         if compress(scan(upcase(datvar),2,'/')) in ( '' 'UNK') then
         _return_value2 = compress(_return_value2) || '--';
         else _return_value2 = compress(_return_value2) || '-'
         ||compress(scan(datvar,2,'/'));
         if compress(scan(upcase(datvar),3,'/')) in ( '' 'UNK') then
         _return_value2 = compress(_return_value2) || '--';
         else _return_value2 = compress(_return_value2)|| '-' ||
         compress(scan(datvar,3,'/'));
        end;

        if not missing(_return_value2) then
        _return_value2=substr(_return_value2, 1 , length(_return_value2)-
        indexc(compress(reverse(_return_value2)),'1234567890')+1);
        if anyalpha(tranwrd(tranwrd(upcase(datvar),'UNK',''),'/','')) GT 0
        then
       put "CP WARNING: Note from dtc_from_dt standard FCMP function, Date
variable has an unexpected value: " datvar ;

    /*
      Return value
    */
      return(_return_value2);
      endsub;
run;

proc fcmp outlib=work.funcs.day_from_dtc;
 function day_from_dtc( datevar1$, datevar2$);
 attrib _return_value3 length=8;
 _tmp_dtvar01 = substr(datevar1,1,10);
 if length(compress(_tmp_dtvar01)) eq 10 then
      _tmp_dtvar1 = input(_tmp_dtvar01,yymmdd10.);
 else
```

```
        _tmp_dtvar1 = .;
            _tmp_dtvar02 = substr(datevar2,1,10);
  if length(compress(_tmp_dtvar02)) eq 10 then
        _tmp_dtvar2 = input(_tmp_dtvar02,yymmdd10.);
  else
        _tmp_dtvar2 = .;

  if n(_tmp_dtvar2,_tmp_dtvar1)=2 then do;
     if _tmp_dtvar1<_tmp_dtvar2 then  _return_value3 =(_tmp_dtvar1-
     _tmp_dtvar2);
      if _tmp_dtvar1>=_tmp_dtvar2 then _return_value3 =(_tmp_dtvar1-
     _tmp_dtvar2+1);
      end;
  return(_return_value3);
endsub;
run;
```

DYNAMICALLY EXECUTING SAS CODE

The power of call execute() within a Data Null step allows dynamically executing SAS statements or macro programs. The example below creates a dataset that contains the names and type from the sashelp.class dataset. This temp dataset will be used in the Data Null step with call execute to call a macro for each record. The report macro will take as parameters name and type to select Proc freq or Proc means depending on the variable type.

```
proc sql;
 create table work.Vars as
 select name, type
 from dictionary.columns
 where memname="CLASS" and libname="SASHELP";
quit;

%macro report(var= , type= );
 %if &type=char %then %do;
 proc freq data=sashelp.class;
  table &var;
 run;
 %end;
 %else %do;
 proc means data=sashelp.class;
  var &var;
 run;
 %end;
%mend report;

data _null_;
 set work.Vars;
 call execute('%report(var='||strip(name)||' , type='||strip(type)||');');
run;
```

Below is the SAS code generated from the call execute in the Data Null step. This method provides great flexibility to process a list of variable names within a dataset.

```
%report(var=Age , type=num);
%report(var=Sex , type=char);
%report(var=Age , type=num);
```

```
%report(var=Height , type=num);
%report(var=Weight , type=num);
```

SYSTEM ENVIRONMENT CLEAN-UP

For each SAS macro development, before existing the macro, there should be a system environment clean-up process. Below is a list of the file categories to be addressed.

1. OPTIONS
2. GOPTIONS
3. System macro variables
4. Temporary data sets and formats
5. Libraries
6. Filenames
7. Titles and Footnotes
8. Macros
9. Macro variables

Reset statements are provided for selected files.

```
title; footnote;
goptions reset=all;
%let syscc = 0; ** Operating environment condition code **;
%let sysrc = 0; ** Operating system condition code **;
%let syslibrc = 0; ** Libname statement condition code **;
%let sysfilrc = 0; ** Filename statement condition code **;
proc datasets library = work kill; quit;

proc sql noprint;
 select unique libname into :mylibs separated by ' clear; libname '
 from dictionary.libnames where libname not in
('MAPS','SASHELP','SASUSER','WORK');
quit;
libname &mylibs clear;

proc sql noprint;
 select name into :mymacrovars separated by ' '
 from dictionary.macros where scope = 'GLOBAL'; quit;
%symdel &mymacrovars mymacrovars;
```

SUMMARY

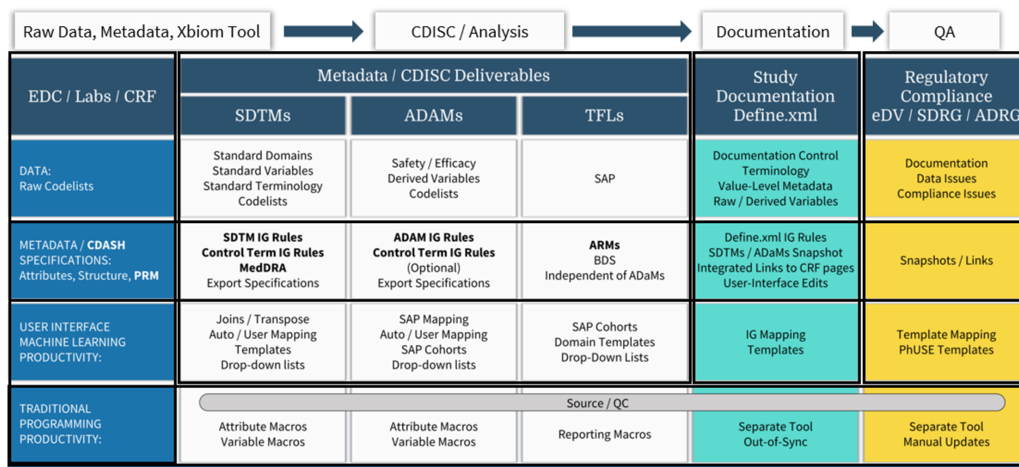
The techniques presented in this paper are a sample of the advanced metadata macro programming. By applying these methods, SAS programmers will have the power to streamline SDTM automation for better control, reduced cycle times and less errors.

For SDTM automation, PointCross's Xbiom tool has a user friendly interface to metadata panels which helps to capture and manage the work process flow. Within any MDR system, it is important to have good version control of CDISC/Control Terminology Implementation Guide (IG) changes and compliance checks. In addition, the MDR tool should also effectively manage legacy studies with an audit trail that shows the history of study and standards implementation guides and control terminology versions and create a robust impact assessment report with end-to-end data flow and changes. This is important for traceability and tracking the evolution of study amendments with CDISC standards and is a key piece of information for communication with regulatory agencies like FDA. See PHUSE's template here: https://advance.phuse.global/display/WEL/Clinical+Study+Data+Reviewer%27s+Guide+%28cSDRG%29+Package.

Below is a summary of key MDR dashboards and key objectives.

- Phases: Design and Plan, Setup, Execute, Analysis and Reporting
- Workflow Process: Control and Manage Operations from Draft, Review and Approval
- Metadata Objects: Raw Data, Global/Study Specifications, Custom Domains, CDISC/Control Terminology IG
- Impact Analysis: Differences in Study/CDISC IGs with drill-down option to variable level
- Implementation of rules in the study data elements across E2E standards (CDASH, SDTM, ADaM etc.)
- Ability to inherit data elements in a hierarchical fashion (Global, Therapeutic Area/TA, Program/ study) level.

Below is the work process flow from end-to-end. The best SDTM automation tools leverage all CDISC metadata throughout the process.



REFERENCES

Accessing SAS® metadata and using it to help us develop data-driven SAS programs, Iain Humphreys
https://www.lexjansen.com/phuse/2009/tu/TU06.pdf

Call Execute: Let Your Program Run Your Macro, Artur Usov
https://www.lexjansen.com/phuse/2014/cc/CC06.pdf

Efficiency Comes From Reusability and Repeatability
Hanming Tu, Dave Evans
https://www.lexjansen.com/phuse/2016/dh/DH08.pdf

Metadata integrated programming, Jesper Zeth, Jan Skowronski
https://www.lexjansen.com/pharmasug/2017/AD/PharmaSUG-2017-AD17.pdf

Monitoring SDTM Compliance in Data Transfers from CROs, Sunil Gupta
https://www.lexjansen.com/pharmasug/2022/QT/PharmaSUG-2022-QT-199.pdf

Proper Housekeeping – Developing the Perfect “Maid” to Clean your SAS® Environment, Chuck Bininger
https://www.lexjansen.com/nesug/nesug09/cc/CC11.pdf

Programmatically mapping source variables to output SDTM (Study Data Tabulation Model) variables based upon entries in a standard specifications Excel® file workbook, Frederick Cieri, Rama Arja, Ramesh Karuppusamy
https://www.lexjansen.com/pharmasug/2019/BP/PharmaSUG-2019-BP-340.pdf

SAS® DICTIONARY: Step by Step, Patrick Thornton
https://www.lexjansen.com/wuss/2008/dmw/dmw07.pdf

Simple Ways to Use PROC SQL and SAS DICTIONARY TABLES to Verify Data Structure of the Electronic Submission Data Sets, Christine Teng, Wenjie Wang
https://www.lexjansen.com/pharmasug/2006/Tutorials/TU03.pdf

Unleash the Power of Less Well Known but Useful SAS® DATA Step Functions, Timothy Harrington
https://www.pharmasug.org/proceedings/2019/AP/PharmaSUG-2019-AP-154.pdf

CONTACTS

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

Sunil Gupta, Submission SME
Gupta Programming
213 Goldenwood Circle
Simi Valley, CA 93065
GuptaProgramming@gmail.com
SASSavvy.com, R-Guru.com

Abhishek Dabral, Director, Clinical Programming, Alkermes Inc
40 Musket Dr
Basking Ridge, NJ 07920
abudabral@gmail.com
https://www.linkedin.com/in/abhishekdabral

*Any brand and product names are trademarks of their respective companies.