The Use of Metadata in Creating, Transforming and Transporting Clinical Data

Gregory Steffens, ICON Development Solutions, Ellicott City, MD, USA

ABSTRACT

Creating databases is a significant part of the work required to analyze clinical trials. Data flows from applications that collect and clean data as well as from many other data sources, such as laboratories which supply electronic lab data. This source data is used to create standardized raw data (e.g. SDTM), study analysis databases and on to integrated databases. This represents a great deal of database design, creation, transformation and validation. This paper presents innovative technical solutions that vastly reduce the effort and cost to implement the data flow, while delivering a consistently higher level of quality. The technology is built with base SAS, standardized metadata set structures and the SAS macro language. So, no new software products are required – just effective design and process. It will be seen that standardizing metadata is at least as important as standardizing clinical trial data and that automation can be data standard neutral, not assuming any data standard. Clarification of the objectives of data standards and metadata is necessary in order to measure the success of these standards. Standards are not an end themselves; they are a means to the end of creating analysis with less effort and cost and to attain a high and consistent quality. Data standards, alone, will not meet these efficiency and quality objectives, standard metadata, data automation and good process definition is also required.

INTRODUCTION

Pharmaceutical companies must address the need to define database structures quickly and easily in order to support such tasks as the following.

- support clinical trial analysis databases
- import data from central laboratories and CROs
- export data to Data Safety Monitoring boards and other data review groups
- share data between corporate sites
- create integrated databases of multiple study databases
- submit data and metadata to the FDA and other regulatory agencies and reviewers

Industry-level standards are being defined by CDISC to help address this need and most pharmaceutical companies have had corporate standards governing database structures for many years. However, often these industry and corporate standards are stored in .pdf or MSword documents or in excel worksheets that do not rigorously follow a metadata standard. As such they are accessible only to people reading the documents. Storing this same information in rigorously standardized metadata sets adds much value to the effort put into defining these standards and study data specifications because this information is made available to computer programs that can automate database creation and data transformations. This paper describes the value of standard metadata set structures that are capable of storing specifications for any of the above database requirements and others too. These metadata sets, along with SAS macros and a SAS/AF application that access these metadata sets, supports the specification, creation, importation, exportation, comparison and validation of databases. Further, these metadata sets can be used to export data and metadata to XML format following the CDISC define schema. The metadata set structures can be implemented in any relational database or in SAS itself. The macros support such functions as:

- publishing the specification in several formats, including the define.xml and define.html formats
- implementing all the data set and variable attributes to the database with a simple macro call (variable name, type, length, label, format, etc.)
- listing discrepancies between the data specification and the database, including structure and valid values
- sorting the data by primary keys
- reordering variables in the PDV according to FDA preferences

- creation of user defined format catalogs
- creation of character decode variables from code variables
- comparison of database structures to standards or to other studies, to assist in enforcing data standards and consistency
- Creation of study specifications from data standards or other study specifications
- transforming a database from one database structure to another, using map metadata which defines a map between metadatabases

As data standards are developed there is great value in documenting these in metadata sets, that are accessible to computer programs, rather than in word or .pdf. The metadata set structures must be standardized and capable of containing descriptions of all the database attributes required to build that database.

In summary, when importing data from outside sources, the data specification is implemented in metadata set so that computer programs can automatically compare the data submitted by the lab or CRO to the specification and list discrepancies where the database does not conform to the specification. When specifying study analysis databases, macros assist in the building of the database by automatically adding all the specified variable and data set attributes, such as labels and formats, adding decode variables, sorting the data, building format catalogs, etc. When submitting data to the FDA a thorough specification can be printed from the metadata or exported to XML format, with bookmarks and hyperlinks.

STANDARDIZED METADATA

The first step is to define a standard metadatabase design and use that standard design in all instances of defining database standards and study database specifications. A metadata design includes two main components – a standard list of attributes of databases and a standard metadata set design used to store those attributes.

THE TWO COMPONENTS OF A STANDARD METADATA DESIGN – STANDARD ATTRIBUTES IN A STANDARD METADATABASE

STANDARD ATTRIBUTES

The first component of a standard metadatabase design is a standard list of attributes that describe the characteristics of databases that are required to be known when creating, validating and reviewing the database. These attributes define database characteristics at different levels of detail, such as attributes of a data set, of each variable, of valid values of variables such as codelists, of descriptions of the variables purpose, of descriptions of the variable's derivation logic and of a special set of attributes that I call "row-level" metadata.

The data set level attributes include a short and a long name of each data set, the data set label, the order the data set should be displayed in a define file, the location of the data set and CDISC attributes required to create a define.xml file such as structure, repeating, is_reference_data, purpose and class.

The variable level attributes include a short and long name of each variable, the data set that the variable is included in, the label, type, length, format to assign to the variable, the name of a list of valid values or codelist to associate with the variable, the relationship between code and decode variables, the order to present the variables in a define file and to order the variables in the data set program data vector, the name of a description to attach to the variable, whether the variable is a header variable that should be copied to other data sets, the ability to flag a variable as a supplementary qualifier when creating CDISC data sets and other attributes required to create the define.xml file. It is also useful to include various flags that categorize variables. For example, flagging a variable as a derived variable, or related to primary efficacy endpoints, or safety endpoints support the ability to publish the metadata content for different audiences, by filtering different subsets of variables to publish. A programmer may need all the variables but others may want to focus only on safety or only on efficacy.

The valid value attributes include a name of the value list that can be assigned to a variable, the valid value of the code variable and the valid value of a decode variable, the ability to define either a list of discrete values or a list of ranges of values and the order in which to display the values in a define file and in TFLs. It is important to represent the relationship between a valid code value and its decode value as well as the relationship between a value list and the variable it is assigned to. For example, metadata should represent a values list for sex that includes both the information that 1=Male and 2=Female and also the fact that this list is associated with a variable named "sex" as a code variable and "sex_dcd" as a decode variable.

The description attributes include the derivation logic to assign to a derived variable, a comment to attach to a variable to explain its source and use and the name of the description.

The final category of database attributes to include in a metadata design is row-level metadata. The objective of this level of attributes is to define tall-thin data structures in a robust manner to more fully support automation of data transformations. Some variables in tall-thin data sets, like VSRN, do not have the same attributes in all rows or observations. Different attributes pertain depending on the row and the row types are defined by a parameter variable like VSTESTCD. I call these variables, whose attributes differ in different rows depending on the parameter variable value, parameter related variables (or paramrel variables). The full set of variable level attributes must be assigned to parameter related variables for each value of the parameter variable. Without this, data transformation automation is not possible and a full description of the database is also not possible. I identified the need for row level metadata in the first CDISC pilot project and we are attempting to demonstrate this in the second pilot, but it will require an enhancement to the define.xml schema. These paramrel variables can also be referred to as virtual variables because they require all the definition that a variable requires but they do not exist as separate physical variables in the database. If the data set was transformed from a tall-thin data structure to a short-wide one, then the virtual paramrel variables would become separate physical variables. The definition of paramrel variables includes the SAS format and so can be useful when generating TFLs. For example, the VSSTRSN variable will have different SAS formats depending on the value of the VSTESTCD parameter variable. The TFL program can read the row-level metadata to find out what format to use to present the variable value with the correct precision.

To illustrate why row level metadata is required, consider the following simple data set examples. In the first data set a tall-thin structure is presented and in the second data set the same information is included in a short-wide data structure. On first glance it would appear easier to define the tall-thin data set as compared to the short-wide one because the tall-thin data set has only 5 variables compared to the 10 variables in the short-wide data set. But really the amount of definition of the tall-thin data set is at least as much as the short-wide one. The tall-thin data set has 14 unique cells to define whereas the short-wide data set has only 10 cells to define. The parameter variable is VSTESTCD and the paramrel variables are VSLOC, VSORRES and VSORRESU. The attributes of the paramrel variables are different for different values of VSTESCD. For example, the format of VSORRES for systolic BP is different than that of BMI and the valid values of VSORRESU are also different for different VSTESTCD values. Defining valid values for VSORRESU would define 4 values, but would allow CM to be a unit for BMI, which is incorrect. But metadata that allows specification of the value of VSTESTCD and the name of the paramrel variable allows specific and robust definition of all its row-level attributes. Transformation of the tall-thin data set into the short-wide data structure requires row level metadata, in order for the transformation program to know what short-wide variables to create and from what tall-thin values.

USUBJID	**VSTESTCD**	**VSLOC**	**VSORRES**	**VSORRESU**
1	SYSBP	STANDING	120	Mm Mg
1	HEIGHT		185	CM
1	WEIGHT		90	KG
1	BMI		26.3	Kg/M**2

USUBJID	SYSBP	BPSYSLOC	BPSYSU	HEIGHT	HEIGHTU	WEIGHT	WEIGHTU	BMI	BMIU
1	120	STANDING	mm Mg	185	CM	90	KG	26.3	Kg/m**2

STANDARD METADATA SET DESIGN

The second component of a metadatabase design includes a standard data design used to store the standard database attributes. This can follow a relational database design, as I have done, or a hierarchical design, as CDISC has done with the define.xml schema.

But whatever design type you follow, it is important to realize that the metadata design is a separate thing from the publication of the metadata content. One advantage of metadata is that its content can be published in different formats; e.g. html, xml, excel, SAS, etc. Metadata design and publication format need to be separated from each other and it is very rare that you would look directly at the metadata itself when you want to learn about the database it describes. Instead you should look at one of a set or publication formats that applies to your need at the time. This publication format can take the form of an xml style sheet or a set of SAS macros that publish the metadata content in the format and level of detail required for the purpose. Separating metadata design from publication format, allows the metadata to fully address two somewhat opposing requirements – to create a highly structured representation of a database structure that is programmatically accessible and to create a database requirements document that is easy for people to interpret and navigate. The metadata design should also be considered as separate from the format

used to enter information into a metadatabase. A GUI can be developed to assist users in entering metadata content and the GUI can present more user-friendly formats and not require the user to learn the metadata structure and relationships.

The following table describes the 5 metadata sets.

Metadata Set Name	**Purpose**	**Primary Key Metavariables**
TABLES	Describes data set level attributes	TABLE
COLUMNS	Describes variable level attributes	TABLE COLUMN
COLUMNS_PARAM	Describes row-level parameter related variable attributes in tall-thin data sets	TABLE COLUMN PARAM variable value PARAMREL variable name
VALUES	Creates controlled terminology lists, valid values lists and code lists that can be associated with a COLUMN or a PARAMREL, by name of the value list.	FORMAT (the name of the value list) START END
DESCRIPTIONS	A SAS catalog of source entries that contain derivation descriptions, methods and comments that can be associated with the specification, a data set, a variable, of a parameter related variable.	DESCRIPTION (the name of the description)

SOME PRINCIPLES OF METADATA

The metadata data design must be rigorously standardized and not deviated from when defining any data standard or study data requirements. If a need arises to change the metadata structure for different purposes then the design is inadequate. Any changes to metadata design must take backward compatibility issues into account, just like changes to reusable code modules. In fact, metadata blurs the line between code and data, so the principles of code change control apply to metadata too.

The metadata data design must facilitate programmatic access. So, a highly structured representation of the standard attributes is essential. For example, when defining what the primary key variables are for each data set, it is much better to create a flag variable in the variable level metadata (the COLUMNS metadata set) rather than creating a character variable in the data set level metadata that contains a delimited list of primary key variable names. The latter, character variable list, design has the following types if inferiorities – 1) the variable names are entered twice in the metadata and may become inconsistent with each other, 2) programmatic access is more difficult because the scan function and merge is required to pull out each variable name and 3) a character metavariable that contains a list of variable names imposes an unnecessary limit on the number of primary key variables.

When designing metadata, eliminate the need for entry of the same information more than once. For example, a valid value list may be applied to multiple variables. Also descriptions may also apply to more than one variable. Separating the values and descriptions from the variable level metadata data design allows multiple use of the values lists and descriptions. So, creating a values list, naming that list and then entering that name into the variable level metadata, supports reuse of that values list in any number of variables. If you need to change that values list, then you have only one place to change it in, rather than needing to make the same change for each variable that is associated with the list.

The placement of valid values in a variable level metadata design is a common but inefficient design. This inefficient design defines code – decode relationships in a single character variable, with a list of <codevar> = <decode value>; e.g. 1=Male, 2=Female. This has the same problem as the primary key design just described. This inefficient design is often the result of the failure to separate metadata design from publication format, as described above. Another reason for this inefficient metadata design is the failure to separate the metadata design from the metadata entry format. A graphical user interface should be created to allow easy population of metadata, shielding the end user from the metadata design and presenting an easy data entry format.

One somewhat controversial principal of metadata design has to do with how derived variable algorithms are represented. There are at least two camps of thought – to represent the details of the algorithms in structured metadata or to represent these details in plain language text that is to be used as a requirement for a programmer. In the first camp, xml schemas that represent mathematical formulas, such as MathML, are often brought up. In the second camp, the more efficient approach is seen to be to define the algorithms in a format useful to a programmer to create re-usable code from. I am in the second camp, since I don't see an adequate metadata design to represent the complex derivations we are required to create and even if this did exist it would be very complex to populate this kind of metadata and the ROI is not evident. Once standard algorithms are described, SAS macros can be created that implement these. This may seem to violate the structured metadata principle, and I recognize the appeal of representing algorithms in metadata, but the technology is just not mature enough for that today. Specifically, what can best be automated in the data flow with metadata-driven macros are the data transformations; the data derivations are automated with SAS macros or other reusable software modules.

THE OBJECTIVES OF A STANDARD METADATABASE

It is important to understand the objectives of a metadatabase design, so that you can test your design against these objectives to determine whether the objectives are fulfilled by the design. The definition of objectives is often not done and left to implicit assumptions that lead to disagreements about implementation. Many of the more controversial discussion points about metadata implementation can be traced back to differing and implicit assumptions about the purpose of metadata. We need a clear definition of the objectives of metadata in order to define a shared set of assumptions. Disagreements about metadata implementation can be often resolved by referring back to these assumptions and objectives. If your objective is to simply create a retrospective define file after the database is created, in order to submit a data description to a regulatory agency, that is a very different project from one whose objective is to define prescriptive metadata at the start of the study to automate data flow and generate sections of the protocol and SAP. The following are some objectives I recommend.

- Store the definition of database standards and also the data requirements for any study specific database using a single metadatabase design.
- Do not require re-entry of the same content in different parts of the metadatabase.
- Include enough attributes in the metadata design to support data flow automation. This is a major reason for creating metadatabases.
- Do not assume any single data standard or set of data standards when designing metadata. Metadata, and the associated automation, designed only for SDTM or only for ADaM will not yield the value that is possible with other standards-neutral designs.
- The standard set of attributes described in the metadatabase should optimize programmatic access to support automation. Publication for programmers and data reviewers should be a separate consideration. Create SAS macros to publish metadata content in different formats and with different levels of detail. Publication formats can include define.xml, define,html, excel, SAS metadata, etc. Recognize the people with different backgrounds and objectives need to access the data, so the publication of the metadata content should support different levels of detail for different users. For example, a programmer needs the detailed derivation descriptions but another user may need this large volume of detail. The publication macro should offer an option to include this detail or not.
- Create SAS macros that automate data flow in a general purpose way. Separate macro code from the specifics of the database requirements. Metadata should be used to inform a general-purpose macro what data base to create and what database attributes to implement. Rather than writing a macro to create the specific SDTM or ADaM data sets, create a macro that can create any data set define by metadata. For example, a general purpose macro can implement all the attributes defined in the variable level metadata, rather than hard-coding assumed attributes in the macro that will need to be changed every time the database requirement changes.

SOME OF THE AUTOMATION THAT IS ENABLED BY METADATA

On primary reason for implementing standards is to automate. Some of the automation has been described above. Here is a list of some of the automation that exists now.

- Creation and storage of data standards in standard metadata

- Creating study data requirements by programmatically copying subsets of the standard metadata to a study or existing study metadata to a new study.
- Publish data requirements in various formats and with varying levels of detail with simple macro calls. Define.xml. define,html,. Pdf, rft, excel, SAS and ODS destinations are just some of the examples of publication formats. Note that all hyperinlinks within the define file and to external files, such as transport data sets and blankcrf.pdf, are all automated
- Implement all database attributes defined in the metadata with one macro call. This includes changing the type of variables, creating SAS date and datetime variables, dropping variables that should not be in the database, adding variables that should be in the database, assigning values to single-value variables, variable labels, length, format, etc.
- Renaming data sets and variables to names less than or equal to 8 characters, so that the can be included in a version 5 transport file for submission to the FDA. Reducing the lengths of character variables so that they also fit into the version 5 transport file.
- Create integrated databases from individual study databases
- Create all decode variables with one macro call
- Copy all header variable to all appropriate data sets with one macro call
- Create supplementary variable data sets with a simple macro call per data set.
- Validate a database to its requirements
- Compare a study data requirement to a standard to ensure compliance
- Compare one study to another to quickly determine the differences in structure and controlled terminology
- Proc compare two SAS libraries to each other with one macro call, including the ID statement. This useful, for example, when using replication programming to validate derived variables.
- Create zero observation data sets from the metadata-resident data requirements
- Transforms a database from one structure to another, with the DTE and map metadata

THE NEXT STEP – MAP METADATA AND THE DATA TRANSFORMATION ENGINE

So far, we have defined an approach that leads to a good deal of automation, including implementation of database attributes with a single macro call, creating study data requirements from a data standard, comparison of a study database requirement to a standard or another study to help ensure consistency and compliance, publishing data requirements in different formats and at different levels of detail and creating suppqual data sets. This automation implements database attributes. But there is still more automation that can be attained related to the transformation of the contents of a source database into a differently-structured target database. This includes data transformations from a simple variable rename to a complex rekeying of the data. The objective here is to automate data transformations with SAS macro code that does not make assumptions about the data it is transforming. This kind of automation does not make assumptions that a study data standard is complied with, the automation assumed only that the metadata describing the source and target data sets exists and that a new kind of metadata is created – map metadata.

Map metadata is an extension of the metadata model that describes databases. This extension defines the relationships between two metadatabases. Metadata describes databases; map metadata describes the relationships between databases. Once these relationships are populated in map metadata, SAS macros, that constitute the data transformation engine, can read this and generate code that automates the transformations of the source database into the target database. Examples include the creation of target SDTM from the source CRF database and the creation of integrated databases from individual study databases.

The following table describes the map metadata structures that correspond to the above-described metadata.

Map Metadata Set Name	**Primary Key Metavariables**
TABLES_MAP	SOURCE_TABLE TARGET_TABLE
COLUMNS_MAP	SOURCE_TABLE SOURCE_COLUMN TARGET_TABLE

	TARGET_COLUMN
COLUMNS_PARAM_MAP	SOURCE_TABLE SOURCE_COLUMN SOURCE_PARAM SOURCE_PARMREL TARGET_TABLE TARGET_COLUMN TARGET_PARAM TARGET_PARMREL
VALUES_MAP	SOURCE_FORMAT SOURCE_START SOURCE_END TARGET_FORMAT TARGET_START TARGET_END

Map metadata makes the assumption that the source and target databases are described with the standard metadata. Map metadata then supplies the definition of a relationship between each observation in the target metadatabase with any one or more observations in the source metadatabase. For example, any variable in the source can be mapped to the target. Variable and paramrel values can be mapped in the values_map, to generate all the code that changes the values of the source database to conform to the target database requirements.

The DTE macros determine the primary key variables in the source and target databases to determine how to copy the value of the source variable into the target database. The DTE assigns a transformation type based on the relationship between the source data set primary keys and the target data set primary keys. These categories are: same, superset, subset, supersub, wide2thin, thin2wide and none.

Primary Key Relationship Category	**Description of Category**
SAME	Source and target are keyed the same so a simple merge is done
SUPERSET	Target has a superset of key variables as compared to the source. A merge of the common key variables is done.
SUBSET	Target data set has a subset of the key variables as compared to the source. Each mapped target variable must create an array in the target.
SUPERSUB	The target data set has both more primary key variables and less than the source data set. A transformation is not supported here because the primary key relationship does not allow it.
WIDE2THIN	The target data set is tall-thin data structure and has one extra primary key as compared to the source data set. For example, a source vitals data set with one variable per vital sign and unit mapped to a target data set with the VSTESTCD primary key variable added. A short-wide to tall-thin data transformation is done.
THIN2WIDE	The target data set is short-wide data structure and has one less primary key as compared to the source data set. A tall-thin to short-wide data transformation is done.
NONE	The target data set has no common key variables with the source. A transformation is not done.

Map metadata is far simpler in structure than the source and target metadata. Map metadata supports the definition of the relationships between metadatabases by including the primary key metavariable values of the source and target metadatabases. For example, if the metadatabase includes a metadata set named “columns” that includes primary key metavariables named “table” and “column” then the map metadata simply contains the variables “source_table”, “source_column”, “target_table”, and “target_column”. This simple 4-metavariable map metadata structure allows you to select any one observation in a source metadatabase and associate it with any one observation on the target metadatabase. There is complexity in the macro that reads the map metadata and generates the data transformation code, but populating map metadata is relatively simple.

THE ADVANTAGES

The advantages of all this is to attain the goal of a highly consistent level of quality with less resources and cost. That is, to do our work faster, better and cheaper. How would you like to create an SDTM – compliant database with code like this?

```
%dtmap(source_mdlib=m,source_prefix=raw_,target_mdlib=m,target_prefix=target_,
 maplib=me,inlib=raw,outlib=sdtm,suppqual_make=yes)
```

Or publish your data requirements with code like this?

```
%mdprint(mdlib=m,inprefix=target_,html=yes,htmlfile=c:\metadata\target_define.html)
```

Or create the define.xml file with code like this?

```
%md2odm(mdlib=m,
 outxml=c:\metadata\SDTM.xml,
 StudyName=ICR_STDM_Standard 3.1.2,ProtocolName=SDTM,
 StandardName=SDTM 3.1.2,StudyDescription=SDTM Standard,
 defineVersion=1.0.0,odmversion=1.2,crt_prefix=def)
```

Standards are the means to automation and automation is the means to improved efficiency and quality. But you may ask about the amount of work required to populate the metadata. The macro calls, above, look easy enough, but how difficult is it to populate the metadata and how does that workload compare to an environment without this automation? This is where good process is required to ensure that the quality and efficiency objectives are attained. Other points to consider are study-level validation, which is greatly reduced by automation, and a change in required skill sets to create quality deliverables.

The advantage of writing less code to create the deliverables is clear. It is both inefficient and riskier to invent or even modify code for each study. The risk is to quality. Newly created or modified code has a higher risk of error and requires more validation as it is used in each study. Reusable code that is used in multiple studies vastly reduces the amount of new code that needs to be created. Reusable code should be validated independently of any specific project. This multi-use validation is more complex than the single-use code invented for a specific project, but after about three studies the validation effort comes out even and the return on investment really begins.

The reuse principle is applied to metadata too. The content of the metadata can provide information to multiple documents that are typically entered separately. Each kind of information should be entered once and used many times. This increases the value of the metadata content. Data requirements in metadata can be published in many documents and used in SAS programs instead of re-entering that information into multiple documents. We should move away from entering information into documents and towards entering that information into structured metadata and generating documents from the metadata. This facilitates the enter once, use many principle and ensure consistency across documents, since there is one source of the information in all those documents.

It is also easier to get a more consistent quality, even from junior staff, with automation. The automation is controlled and defined by the most experienced staff and the business rules are implemented in macros. Junior staff, calling these macros, generates better deliverables because these business rules are implemented in the macro and the learning curve is greatly reduced.

Metadata data can be populated in a number of ways and by staff with different skill sets, as compared to a non-metadata environment. Without automation the skill set required to transform a database include SAS programming, database design and knowledge of clinical data. With automation new divisions of labor become available. Someone familiar with clinical data can populate metadata and a SAS programmer can use that metadata when creating, validating and transforming the data.

CONCLUSION

I have shown that innovative design of metadata and an associated SAS macro library can deliver automation of a large portion of data flows, during most phases of the database creation process, from specification through build and on to validation and delivery. The standardization of metadata and the automation that it enables, can be done in a way that is data standard neutral. This protects your investment, of the time to create the automation, from the inevitable changes in data standards. In fact, standardized metadata has a larger positive and more long-lasting impact, as compared to standardizing data. This impact has to be measured against of clearly defined objectives. These objectives also guide you when designing the metadata and the automation.

CONTACT INFORMATION

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

Gregory Steffens
ICON Development Solutions
6031 University Boulevard
Ellicott City, MD 21043
Work Phone: 410-696-3194
Fax: 410-480-0776
Email: Gregory.Steffens@iconplc.com
Web: www.iconplc.com/company/service-divisions/icon-development-solution/