PhUSE EU Connect 2018

Paper SI06

Define.XML-centric dataset development

Leonid Tseytlin, CS Ltd, Kharkov, Ukraine

ABSTRACT

Define.xml is a well-known and established standard for representing dataset metadata in a machine-readable format for data interchange and clinical study submission. This article advocates a wider use of define.xml in the process of dataset development and presents several cases where the use of define.xml can facilitate programming.

INTRODUCTION

Define.xml is a well-known standard, created by CDISC and supported by the FDA. As the standard authors note in the specifications, “The purpose of Define-XML is to support the interchange of dataset metadata for clinical research applications in a machine-readable format. An important use case for Define-XML is to support the submission of clinical trials data in CDISC SDTM, SEND or ADaM format to regulatory authorities”. Perhaps it is this statement of purpose by the authors that leads many teams to treat define.xml as something that is “for submission only”, and as such needs to be created at the end of the study analysis, after all datasets are finalized, almost as an afterthought.

What is often overlooked is that define.xml offers many useful opportunities in the process of dataset programming. Indeed, let us pause for the moment and think: what is the key motivation for define.xml as opposed to previous formats such as define.pdf? It is, of course, that define.xml is machine-readable, i.e. can be parsed, analyzed and processed programmatically. Why not make use of it in dataset programming?

DEFINE.XML-CENTRIC DATASET DEVELOPMENT

In the define.xml-centric paradigm, define.xml is created before any datasets are. Any programming team worthy of its name will create dataset specifications before getting down to programming. Define.xml is essentially machine-readable dataset specifications. Therefore define.xml needs to be created from the dataset specs.

Our team creates dataset specifications in an Excel template. We created a utility (implemented as Excel add-in) that takes these specifications in Excel as input and converts them to define.xml. An alternative to in-house spec template and utility is Pinnacle21 Community Edition. This popular tool can create define.xml out of Excel specifications; one can maintain their specs in Pinnacle21-compatible format and use this tool to generate define.xml. One does not need to wait for the specs to be finalized: this conversion is done on ongoing basis in the process of specs and dataset development. The current version of define.xml is available for dataset programs to use.

We further created a set of SAS macros that take define.xml as input and help in various dataset programming tasks. The following sections detail these tasks.

DECLARING VARIABLE ATTRIBUTES

How many times did you write a piece of code like this:

```
data dm;
  attrib STUDYID length=$16 label = "Study Identifier";
  attrib DOMAIN length=$2 label = "Domain Abbreviation";
  attrib USUBJID  length=$23 label = "Unique Subject Identifier";
  attrib SUBJID length=$6 label = "Subject Identifier for the Study";
```

etc.

Probably way too many. But do you have to? All this information is already contained in define.xml. Why not use it? We wrote a macro that reads these variable attributes from define.xml and translates them into appropriate SAS attrib statements. Thus the variable declaration becomes just this:

```
data dm;
  %DefineVarAttrib;
```

Where DefineVarAttrib macro expands into all the necessary attrib statements.

VERIFYING THE DATASET STRUCTURE

Quality is always of utmost importance for any SAS programmer. How do you make sure that your dataset adheres to specifications? Of course, we all do QC by double-programming and we use Pinnacle21 to check our datasets for standard compliance. But here is something that a dataset programmer can quickly use to catch errors even before the QC process begins.

We created a macro that compares the dataset against its specifications in define.xml. It verifies the following:

1. That the dataset has all the variables from the specs and does not have any extra variables
2. That all variables have the expected attributes (type, length, label, SAS format)
3. That all required variables are not missing
4. That variables with associated code lists have values compatible with these code lists
5. That date/time variables in ISO 8601 format are formatted properly
6. The trickiest part: that value-level metadata is adhered to.

If the dataset programmer violated any of the above requirements, they will get an error message and will have a chance to fix those issues as early in the development process as possible.

CONTROLLED TERMINOLOGY: CODES VS. DECODES VS. RANKS

Have you ever faced the following problem? Say, you are creating a summary of disposition. Your disposition terms (in DS.DSDECOD or perhaps ADSL.DCSREAS) are in standard terminology, something like this:

ADVERSE EVENT
LACK OF EFFICACY
PROTOCOL DEVIATION
WITHDRAWAL BY SUBJECT

But probably you don't want to present them in a table in all uppercase. Maybe you want to replace these standard terms with equivalent, but different terms in the sponsor's terminology. Something like this:

Adverse Event
Lack of Treatment Effect
Protocol Violation
Subject Withdrew Consent

Moreover, maybe you don't want to present them in the alphabetical order, but in some sponsor-defined order, e.g. in the order these options appeared on the study CRF. How do you do this? Programmers often create formats and informats to map the standard terminology of the datasets to the study-specific terminology and sort order used in tables. The creation of these formats and informats can be simplified with the use of code lists. It would be appropriate to define the following code list for this case:

Rank	Coded Value	Decode
1	ADVERSE EVENT	Adverse Event
4	LACK OF EFFICACY	Lack of Treatment Effect
2	PROTOCOL DEVIATION	Protocol Violation
3	WITHDRAWAL BY SUBJECT	Subject Withdrew Consent

Here Rank represents sort order; Coded Value is the standard terminology found in the datasets; Decode is the study-specific term to be shown in tables.

We wrote a macro that translates such code list found in define.xml into a group of SAS formats and informats that map the standard terminology to the study-specific terms and sort orders. Then programmers can use put and input functions to quickly do the translation:

```
dsterm_no = input(dsdecod, ncomplt.);
dsterm_disp = put(dsdecod, $ncomplt.);
```

SUPPLEMENTAL QUALIFIERS DATASETS

The notions of supplemental qualifiers datasets is one of the less programming-friendly features of the venerable SDTM standard. Hopefully, it is to be phased out and replaced with non-standard variables in SDTM IG 3.3, but this has not happened yet.

It is not hard to merge the supplemental qualifiers dataset back to its main dataset, and even write a macro to do it automatically, but all supplemental qualifier variables will have the same type and length. Indeed, if I have a Y/N flag, which naturally would have the length of 1 and a comment with the length of 200, my QVAL is going to have the length of 200. Therefore my 1-character Y/N flag will end up also having the length of 200. Moreover, if I have a numeric supplemental qualifier variable, it will have to be stored in the same 200-character QVAL variable.

However, define.xml can define the exact type of each supplemental qualifier using a value list associated with the supplemental qualifier dataset. In this value list we can record that my Y/N flag has the length of 1 and is associated to NY code list, while my numeric variable has integer type.

We wrote a macro that merges a supplemental qualifier dataset back to its main dataset and simultaneously converts QVAL to the appropriate type specified in define.xml. Thus different supplemental qualifier variables can have different types and lengths.

CONCLUSION

We demonstrated how define.xml can be created early in dataset programming process and can be used to facilitate programming of datasets. These techniques help ensure quality and speed up the development process. Having dataset metadata available in a machine-readable format is very valuable for programming. It opens a lot of possibilities, a sample of which was presented in this article.

REFERENCES

Define.xml standard: https://www.cdisc.org/standards/data-exchange/define-xml.

ACKNOWLEDGMENTS

The author thanks his programming team at CS Ltd for invaluable help in the development of the techniques presented in this article.

CONTACT INFORMATION

Leonid Tseytlin
CS Ltd
42a Tobolskaya street
Kharkov 61072
Ukraine
Email: lz@csltd.com.ua