PharmaSUG 2019 - Paper SS-056

Practical Guidance for ADaM Dataset Specifications and Define.xml

Jack Shostak, Duke Clinical Research Institute

ABSTRACT

The goal of this paper is to provide practical guidance for how to specify ADaM analysis datasets within the confines of Define-XML nomenclature. This paper's guidance is presented in a tool and software agnostic manner. The audience of this paper is for individuals that specify and produce ADaM datasets and the associated define.xml file and who have little to moderate prior experience in doing so. A number of practical issues are explored. Source/derivation/comment text rendering is examined. Then contrasting define file specifications versus programmer ETL specifications is addressed. ADaM source and origin type is explored, followed by a deeper dive into parameter value level metadata. How to reconcile variable and parameter value level metadata is discussed. Further issues around comments, controlled terminology, dates, lengths, object order, and results metadata are explored to close out the paper.

INTRODUCTION

CDISC's gift to the pharmaceutical research community has been the power of metadata, and its ability to be leveraged to standardize and transform work processes. When we think of the metadata that defines CDISC ADaM, we think of metadata that describes and defines the analysis datasets, as well as the metadata that describes the analysis results. How that metadata is created and sourced varies widely across the industry, with sources including spreadsheets, homegrown applications, and commercially available software. Then, those metadata sources are transformed via additional homegrown applications or commercial applications into the regulatory reviewer-ready define.xml file, which is defined by the CDISC Define-XML standard.

There are two operational processes associated with CDISC ADaM metadata and define.xml, and that is the collection of the ADaM metadata, and the rendering of that metadata in define.xml. This paper doesn't address those two specific business processes themselves. This paper is focused on how ADaM and its metadata can be best specified, in a tool agnostic fashion, so that it can be best rendered in define.xml. The task of specifying ADaM datasets is typically one of a statistician or statistical programmer. It is my hope that this paper will give you some practical tips and guidance about how to specify ADaM in a way that is define.xml friendly.

The examples in this paper use the following documents as a foundation:

- *CDISC Define-XML Specification* Version 2.0
- The latest revised Define-XML stylesheet version 2018-11-21 found at https://wiki.cdisc.org/display/PUB/Stylesheet+Library.
- *Analysis Data Model (ADaM)* version 2.1
- *Analysis Data Model Implementation Guide* Version 1.1.
- *Define-XML Version 2.0 Completion Guidelines version 1.0 Draft* PhUSE guidance

Please note that the terms "derivation" and "method" are used synonymously throughout this paper. It is also worth noting that when things are stated as fact in this paper that those are based upon these referenced CDISC or PhUSE documents. When this paper goes beyond factual assertions, those will be prefaced with "in my opinion," as those are solely the opinion of this author. Finally, the define.xml samples seen in this paper use the latest CDISC stylesheet, so they may appear different than what you are used to seeing.

"ORIGIN / SOURCE / METHOD / COMMENT" SCREEN SIZE CONSIDERATIONS

For ADaM, the latest CDISC supplied stylesheet for define.xml renders a column on the far right of the browser window for the variable and value level metadata sections called "Origin / Source / Method / Comment." Older stylesheets titled this as "Source/Derivation/Comment," but it is conceptually the same. This column is actually a merger of those four different pieces of distinct ADaM metadata. Unfortunately, because of this, you have to be careful about text length when you have both a derivation and a comment for a variable or parameter value level metadata item. Assuming we are speaking about a derived variable, then the way that this Define-XML metadata renders in a browser looks like what you see here in Figure 1:

Origin / Source / Method / Comment
Derived: Your derivation text goes here after the "Derived" source above. Your comment text goes here after the derivation text.

Figure 1 Sample Origin / Source / Method / Comment

When you are defining your derivations and comments for ADaM variables and parameter value level metadata, you will want to keep text length in mind. It isn't just a matter of keeping your derivation length reasonable, but you have to keep the overall combined text size of your derivation text and comment text at a reasonable level to ensure that it doesn't overwhelm the screen. I suggest a maximum word count for the derivation and comment text of around 300 words to maintain define file readability. Text lengths beyond that will tend to overwhelm the screen of most users and you would be better served by providing links to external documentation.

STATISTICAL PROGRAMMING ETL INSTRUCTIONS AND DEFINE.XML'S METHOD

The Define-XML rendition displays a "Method" that is used for both variable level as well as parameter value level derivation definitions for ADaM datasets. The idea is that "Method" provides a reviewer of define.xml with a derivation, method, algorithm, or reproducible rule that indicates how a variable or a set of parameters (subset of rows) in an ADaM dataset were derived. For a variable level derivation, that method might look like this gray shaded text in Figure 2 below:

Variable	Label / Description	Type	Length or Display Format	Controlled Terms or ISO Format	Origin / Source / Method / Comment
AGEGR1	Pooled Age Group 1	text	10	["<55 YEARS" = "<55 Years", ">=55 YEARS" = ">=55 Years"] <AGEGR1>	Derived: If ADSL.age<55 then "<55 years"; else if ADSL.age>=55 then ">=55 YEARS"

Figure 2 Sample "Method" Text

That "Method" text in Define-XML parlance is called a MethodDef. In the Define-XML specification, it says that:

> "To enhance traceability users are encouraged to provide descriptions that include accurate and consistent references to source variables and derivations."
>
> "For cases where the algorithm description is longer than a few lines, the MethodDef can include a def:DocumentRef element to link to a section in a supplemental document containing the additional details."
>
> "A MethodDef can also link to a text file that contains the specific programming code to be executed." (CDISC 2013, p. 44)

The ADaM model document also describes “Source/Derivation” as:

> “Provides details about the variable’s lineage – what was the predecessor, where the variable came from in the source data (SDTM or other analysis dataset) or how the variable was derived. This field is used to identify the immediate predecessor source and/or a brief description of the algorithm or process applied to that source and can contain hyperlinked text that refers readers to additional information. The source / derivation can be as simple as a two level name (e.g., ADSL.AGEGR) identifying the data file and variable that is the source of the variable (i.e., a variable copied with no change). It can be a simple description of a derivation and the variable used in the derivation (e.g., “categorization of ADSL.BMI”). It can also be a complex algorithm, where the element contains a complete description of the derivation algorithm and/or a link to a document containing it and/or a link to the analysis dataset creation program.” (CDISC 2009, p. 15)

What we see here both in the ADaM and Define-XML documentation is that a source/derivation can be:

1. A simple pointer to a source variable.
2. A “brief description” that is no “longer than a few lines” or a “complete description.”
3. A link to program code or external documentation.

That leaves quite a bit of latitude to you as you create your derivations. You also have two business needs to address for your ADaM derivations. On one hand, you need detailed and precise ETL (Extract, Transform, Load) instructions for the programmer, analyst, or even machine to create the derived ADaM content. On the other hand, you want something in the define.xml file that a human reviewer can readily understand and digest. Often times those two things can be synonymous. However, sometimes those two things may be a bit different when the derivation is very complex. You may need a more terse definition for your define.xml “Method” text so that may be easily consumed, but need a more precise detailed and lengthy definition for your ETL instructions.

In my opinion, I would also strongly suggest that you not place raw programming code into the define.xml derivation like you see here in Figure 3:

Variable	Label / Description	Type	Length or Display Format	Controlled Terms or ISO Format	Origin / Source / Method / Comment
AVAL	Analysis Value	float	10		Derived: proc sort data=sdtm.lb out=lb; by usubjid visitnum lbdtc; run; data calculate_aval; set lb; *…. and on and on with SAS code….*

Figure 3 Raw Code as a Method

Even for a programmer, that is not very helpful. For a non-programmer it is essentially worthless and potentially frustrating. At another extreme, I think placing copious text into the derivation isn’t completely helpful either as you see here in Figure 4:

Variable	Label / Description	Type	Length or Display Format	Controlled Terms or ISO Format	Origin / Source / Method / Comment
AVAL	Analysis Value	float	10		Derived: First, the SDTM LB laboratory domain data is sorted by visit and laboratory collection date. Then, that data is joined with the SDTM EX exposure dataset so that you

					fetch the last standardized result of alkaline phosphatase available before the last injection of study medication. Then, you take that alkaline phosphatase value and perform a Cowhill's transformation of that value which involves…… and on and on and on….

Figure 4 Copious Text as a Method

In my opinion, when it comes to derivation text for the define file, it is best to create a balanced mixture of pseudocode and prose to guide the reader for define.xml derivations that end up in the "Origin / Source / Method / Comment" cell. The following, in my opinion, shows a nice mixture of pseudocode logic and prose for the method in Figure 5:

Variable	Label / Description	Type	Length or Display Format	Controlled Terms or ISO Format	Origin / Source / Method / Comment
AVAL	Analysis Value	float	4		Derived: Derived from ADEF.ADY (where PARAMCD="XPPAIN" and ADY>1) or ADAE.ASDY (where CQ01NAM="PAIN EVENT"). AVAL is the earliest day when (ADEF.CHG<0) for an event, or (ADEF.CHG>0 or ADAE.ASDY>0) for subjects censored due to pain worsening, or the last record (where ADEF.CHG=0) for subjects censored due to no improvement or worsening by the time of the last assessment.

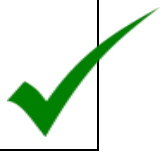

Figure 5 An Example of Balanced Method Text

The "Origin / Source / Method / Comment" define.xml column is dynamically sized in your browser, so it is hard to speak in absolutes, but in the event that the derivation is greater than 300 words or you have the need for textual formatting (e.g., bulleting, font effects) beyond simple ASCII text in your derivation, then I suggest that you insert a link to an external specification document for your derivation. That linked specification document could be to the Analysis Data Reviewer's Guide (ADRG) itself, an appendix to the ADRG, or a separate specifications document entirely. It is worth noting that the latest CDISC supplied stylesheet from 2018-11-21 honors whitespace and line breaks for derivations.

Of course you can also add a link to the ETL program itself from the derivation in define.xml. I do not recommend using the program itself as sole sufficient documentation for a derivation as that is the same as just placing programming code in your define file. The reviewer would then have to hunt through your program to look for the specific derivation parts of interest, and they would have to understand your program which would likely take precious time. A link to the program that creates the derivation should be a secondary consideration when it comes to documenting derivations.

ADAM SOURCE AND DEFINE-XML ORIGIN TYPE

The CDISC ADaM document defines the metadata concept of "source," and the equivalent concept in CDISC Define-XML is "origin." These concepts tell you from where the origin, or source, a variable/column or parameter-level item comes from. The Define-XML standard defines a specific origin "type" which includes: "CRF," "Derived," "Assigned," "Protocol," "eDT," and "Predecessor." The types of "CRF," "Protocol," and "eDT" are essentially owned by the SDTM. That leaves us to use "Derived," "Assigned," and "Predecessor" origin types for ADaM variables and parameters. Let's look at defining each one, with increasing level of complexity:

- **Derived** – If the variable or set of parameter analysis values is calculated, then origin type is "Derived." That is rendered like this in the define.xml file:

Origin / Source / Method / Comment
Derived: Your derivation text goes here.

- **Predecessor** – If a variable is copied from another dataset into an ADaM dataset "as-is," then the origin is "Predecessor." That renders like this in the define.xml file:

Origin / Source / Method / Comment
Predecessor: DATASET.VARIABLE

Where "DATASET.VARIABLE" is the predecessor object such as "DM.USUBJID." This seems simple enough, but there are a few gray areas here.

In the event that a set of column data from a SDTM or ADaM dataset is copied into a set of ADaM BDS parameter analysis values "as-is" without any change beyond perhaps sub setting the records, then those parameter rows could be set to origin type of "Predecessor." You can see this in the following Figure 6 where some SDTM LB lab records are copied into an ADaM BDS dataset:

SDTM LB Dataset	
LBTESTCD	**LBSTRESN**
ALT	20.5
ALT	21.0
ALT	21.5
ALT	22.0

ADaM Basic Data Structure Dataset	
PARAMCD	**AVAL**
ALT	21.0
ALT	21.5
ALT	22.0

Figure 6 SDTM to ADaM Predecessor Content

In the event that a SDTM supplemental qualifier entity is copied into an ADaM column variable using the value of QNAM as the ADaM variable name, or as a set of BDS parameter analysis values "as-is" with no transformation of the data, then in my opinion this copied data could be set to origin type of "Predecessor." You can see an example of this in the following Figure 7 where the SDTM SUPPDM QNAM of "CHRSTIC2" has its QVAL row value of "Y" copied into the ADaM ADSL covariate variable CHRSTIC2:

SDTM SUPPDM Dataset	
QNAM	**QVAL**
CHRSTIC1	Bedroom
CHRSTIC2	Y
CHRSTIC3	Bicycles
CHRSTIC4	Baths

ADaM ADSL Dataset
CHRSTIC2
Y

Figure 7 SDTM SUPPQUAL Predecessor to ADaM ADSL

In the event that an SDTM variable such as VISIT and VISITN are copied into the ADaM BDS variable called AVISIT and AVISITN "as-is" and without derivation, then the origin type could be set to "Predecessor" as you see in Figure 8:

SDTM LB Dataset	
VISIT	**VISITN**
Week 1	1.0
Week 2	2.0
Week 3	3.0

ADaM Basic Data Structure Dataset	
AVISIT	**AVISITN**
Week 1	1.0
Week 2	2.0
Week 3	3.0

Figure 8 SDTM Variable Copied into ADaM Variable

In each of these cases, all that was done is that data was copied "as-is" from one place to another. Yes, the structure may have changed in some cases either from column to row or from row to column, but the actual data content did not change which seems to legitimize the use of setting origin type to "Predecessor." Note that a subset of data copied from one place to another satisfies the origin type of "Predecessor," but if any of the ADaM row or column content is derived in the target row or column, then the origin type should be "Derived."

- **Assigned** – In the Define-XML specification it states that assigned values can be, "Data that is determined by individual judgment (by an evaluator other than the subject or investigator)... This may include third party attributions by an adjudicator." (CDISC 2013, p. 80) Often clinical endpoints or outcomes in research are adjudicated by experts to determine if a meaningful endpoint was achieved or not. The complication is how that adjudication data gets collected. In my work, I am used to complete and detailed adjudication data that gets collected in case report forms, databased, and then included in my SDTM data. In my opinion, the origin type of that data in ADaM is "Predecessor." However, if your adjudication information is not stored in the SDTM, and somehow that adjudication happens directly in your ADaM datasets, then origin type could be considered to be "Assigned."

  The Define-XML specification also says, "Values that are set independently of any subject-related data values in order to complete SDTM fields such as DOMAIN and --TESTCD are considered to have an origin type of 'Assigned.' " (CDISC 2013, p. 81) Although that is written in SDTM-centric logic, that instructive text taken in the same spirit for ADaM variables in my opinion could allow for the following candidate variables to be set to the origin type of "Assigned":

  > ASEQ, PARAM, PARAMCD, PARAMN, PARAMTYP, PARCATy, PARCATyN, BASETYPE, CRITy, MCRITy, DTYPE, AWRANGE, AWTARGET, AWLO, AWHI, AWU, SRCDOM, SRCVAR, SRCSEQ, ATOXGR, BTOXGR, ANRLO, ANRLOC, ANRHI, ANRHIC, AyLO, AyLOC, AyHI, AyHIC

  Note that I say "could" be set to origin type of "Assigned." If this content is derived by some method algorithm, then setting origin type to "Derived" may make more sense. Also, per the draft Define-XML Version 2.0 Completion Guidelines document, it states that secondary or "counterpart" ADaM variables can be considered to be assigned. (PhUSE 2018, p. 32) So, for example, the secondary variables in the table below could be of origin type "Assigned" in the dataset in which they are introduced:

Primary Variable	Secondary of type "Assigned"
SEX, RACE, etc.	SEXN, RACEN
SITEGRy	SITEGRyN
REGIONy	REGIONyN
AGEGRy	AGEGRyN
*FL	*FN
TRT*P, TRT*A	TRT*PN, TRT*AN

TRTSEQP, TRTSEQA	TRTSEQPN, TRTSEQAN
TRxxPGy, TRxxAGy	TRxxPGyN, TRxxAGyN
TSEQPGy, TSEQAGy	TSEQPGyN, TSEQAGyN
TRCMPGy	TRCMPGyN
DTHCGRy	DTHCGRyN
TRTPGy, TRTAGy	TRTPGyN, TRTAGyN
AVISIT	AVISITN
ATPT	ATPTN
APHASE	APHASEN
APERIOD	APERIODC
ASPER	ASPERC
AVALCATy	AVALCAyN
BASECATy	BASECAyN
CHGCATy	CHGCATyN
PCHGCATy	PCHGCAyN
SHIFTy	SHIFTyN
MCRITyML	MCRITyMN

Logically synonymous and assigned variable items like PARAM, PARAMN, and PARAMCD have proven to be a source of origin type definition confusion and variability for our community. Some have set one variable, such as PARAM, to be a derived entity, and then PARAMN and PARAMCD to be assigned based on the idea that PARAM was derived in the ETL code that creates the BDS dataset. However, in my opinion, I believe it makes more sense to set all there off these origin types to be “Assigned.” The draft Define-XML Version 2.0 Completion Guidelines document agrees with this assessment as well. (PhUSE 2018, p. 32)

HOW MUCH PARAMETER VALUE LEVEL METADATA DO YOU NEED?

I have always said that an advantage that ADaM construction has over the SDTM, is that ADaM tends to have a bit less controlled terminology to manage. Unfortunately, to counterbalance that, ADaM tends to have more parameter value level metadata than the SDTM has value level metadata. The reason for this is that the ADaM Basic Data Structure is the dominant ADaM data structure, and there is quite a bit of derived content that ends up in the analysis values in those datasets. This leads to a significant need for parameter value level metadata to store those derivations.

The question becomes, how much parameter level metadata do you need? The Define-XML specification indicates that you have to have:

- A where clause specification
- Data type
- Origin type
- Length
- SignificantDigits for floating point numbers
- SASFieldName for regulatory submissions
- Where the content came from, or Description (i.e., DM.AGE), for origin type of “Predecessor”
- Derivations, or MethodDef, for any origin type of “Derived

Remember what ADaM is in essence. It is value added and often derived content layered on top of the SDTM data substrate. If you don’t fully describe your derived data content, then you are missing the point of creating ADaM datasets in the first place.

HARMONIZED VARIABLE AND PARAMETER VALUE LEVEL METADATA

One issue that causes problems with specifying ADaM BDS datasets and define.xml files is the possible conflict between variable and parameter value level metadata. In general terms, if a metadata characteristic varies per BDS parameter, then it should be specified at the parameter value level in the define.xml file metadata. Also, parent variable level metadata characteristics should be able to encompass the parameter value level metadata characteristics.

Let's look at each metadata component to see how to specify it properly across variable and value level metadata for ADaM.

NAME, LABEL, AND SASFIELDNAME

Although parameter value level metadata links to the parent variable, such as AVAL, via a where clause, the parameter value level metadata also has comparable name, label, and SAS field name components. In Define-XML version 2.0, that "label" is now referred to as the "Description," but it is still essentially the label in SAS parlance. However, these three items do not render in define.xml with the old default Define-XML 2.0 stylesheet in your browser and they are essentially "hidden" in the define.xml. Label is no longer hidden from view with the new define.xml stylesheet. As such, I would suggest one of two approaches:

1. Leave the parameter value level name, label, and SAS fieldname the same as the parent variable.
2. Distinguish the parameter value level name, label, and SAS fieldname from the parent variable. Note that this is primarily an option when using comparator "EQ" in the where condition.

If you choose the second option, it would allow you to easily transpose your ADaM parameter value level metadata items from row entities into column entities if needed. This can be useful if your ADaM BDS parameter value level content needs to quickly become variable level content. This could happen if you needed to have some BDS parameter level AVALs appear as covariates for a model in another BDS dataset, an ADaM OTHER dataset, or even ADSL. In Figure 9 you can see how the ADEFF parameter value level metadata can be leveraged via PROC TRANSPOSE to become variables in the ADOTHER dataset.

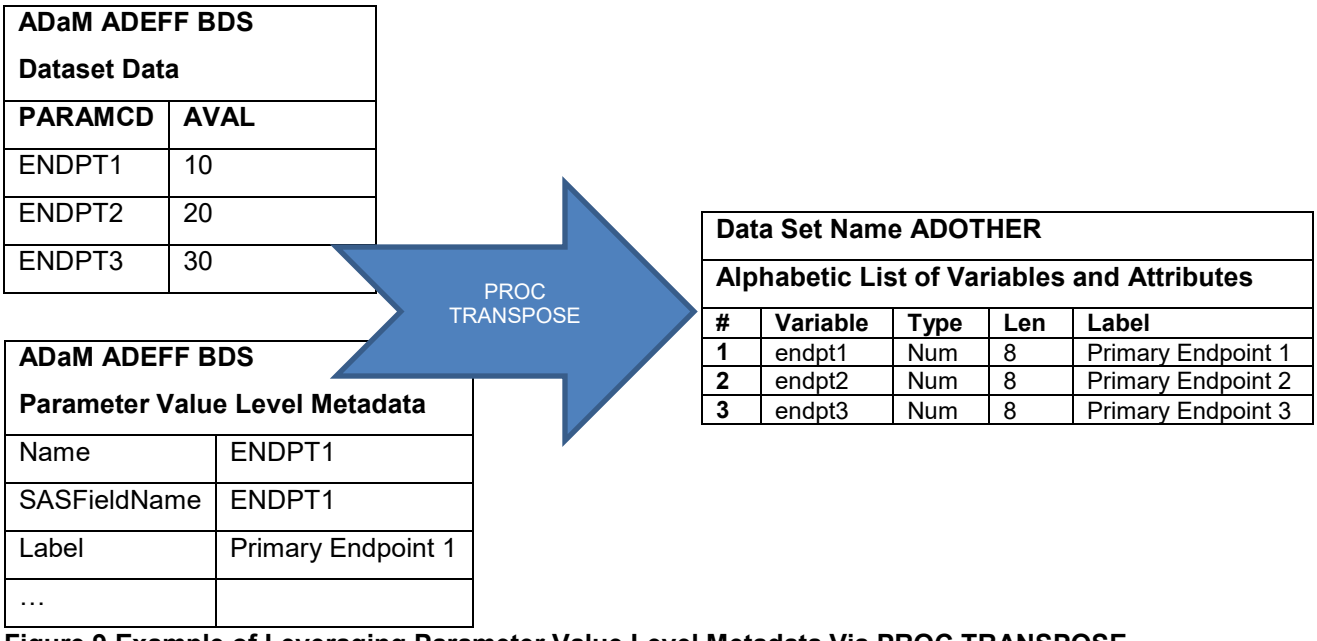

Figure 9 Example of Leveraging Parameter Value Level Metadata Via PROC TRANSPOSE

TYPE

Type has historically been a confusing thing to specify for ADaM at the parameter value level when it is different from the parent level variable. Fortunately, the *Define-XML Version 2.0 Completion Guidelines* document gives us timely guidance for how to resolve these type conflicts. In brief, the parent level variable type must be able to "contain" the parameter value level type, and the type of "text" is all encompassing. That can be best represented by the following Figure 10 from that guideline (PhUSE 2018 p. 40):

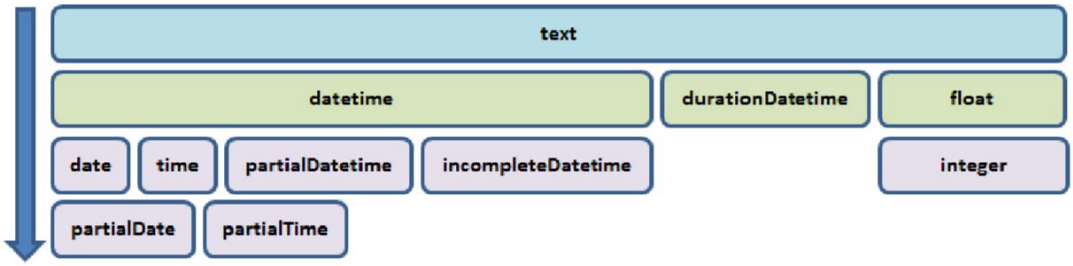

Figure 10 Data Type Compatibility

So, if the AVAL variable is of type "float," then a set of parameters within it could be of type "integer," because float is further up the tree above and can "contain" an integer parameter value. As you can see above, a variable type of "text" can encompass all parameter value level types.

LENGTH AND DISPLAY FORMAT

Length must be specified for the parameter value level content, and it cannot exceed the length at the parent variable level. If display format is defined, the variable level display format must be able to encompass and also display the parameter value level formats. So, if AVAL has a display format of "6.2," then the parameter value level could be a smaller display format of perhaps "4.1." Just like length, the parent variable display format must be able to "contain" any parameter value level display formats.

ORIGIN / SOURCE

Knowing that in ADaM a origin must be "Predecessor," "Assigned," or "Derived," if the sources at the parameter value level vary at all, then it cannot be specified at the variable level and should remain specified at the parameter value level only.

CODELIST / CONTROLLED TERM

The *Define-XML Version 2.0 Completion Guidelines* document does a nice job of explaining how to handle codelists for parameter value level data. In general, codelists should contain all categorical levels of the data that were possible for the study, for a given variable or parameter. Codelists should also be custom named to the item at hand. For example, having a single codelist called UNITS that handles laboratory data, vital signs, medications, and the like is a bad idea in my opinion. With this in mind, if you do have parameter value level metadata for a given parameter, then the associated codelist, if you have one, should be limited to the values that were possible for that specific parameter. I would likely not include a variable level codelist in this case, but if you do, in my opinion you need to make sure that your variable level codelist encompasses all of the categorical values of the parameter value level codelists.

DERIVATION

In general, if a derivation applies to all rows in a dataset, then it can be defined at the variable level, and if it varies at the parameter value level, then it should be specified there. I have seen cases where in a BDS dataset, a variable level derivation is presented as well as additional parameter value level derivations. The idea there is that the parameter value level derivations are additive to the parent level variable derivations. However, in my opinion, this is a bit dangerous. If a reviewer looks at the variable level derivation and stops looking further, then they will not truly understand how AVAL is derived throughout

the dataset. In this case, I suggest that you set the variable level comment text to, “See parameter value level metadata” and then fully specify the derivations at the parameter value level.

COMMENTS ARE NOT DERIVATIONS

Please keep in mind that comments are not derivations. Sometimes, there may be a desire to put variable or parameter value level derivations into the comment metadata element, but please don’t do that. As it says in the Define-XML specification, “Comments are not intended to replace a properly defined computational algorithm, which is expected for derived variables.” (CDISC 2013, p. 46) Those algorithms belong in the derivation and not in the comments.

WHEN TO CREATE CONTROLLED TERMINOLOGY

In many places in the ADaM Implementation Guide you will find that “Codelist/Controlled Terms” is left blank and is not prescribed for you. The ADaM doesn’t provide much guidance about when you should optionally define controlled terminology, but that doesn’t mean that you don’t need to define controlled terminology for ADaM. In my opinion, you should have ADaM controlled terminology for categorical data items that have a prescribed and limited set of values. This is also supported by the *Define-XML Version 2.0 Completion Guidelines* document which states that, “In addition to variables subject to controlled terminology as per CDISC IGs and sponsor-specific controlled terminology, codelist should also be provided for all other variables and value-level definitions which have a predefined and finite set of categorical allowable values.” (PhUSE 2018, p. 51)

These ADaM variables that can optionally have controlled terminology, but in my opinion, it should probably be mandatory that you define codelists for these variables:

> PARAM/PARAMCD/PARAMN and AVISIT/AVISITN

These ADaM variables also typically lend themselves to creating codelist controlled terminology as well:

> **ADSL**
>
> SITEGRy/SITEGRyN, REGIONy/REGIONyN, AGEGRy/AGEGRyN, RACEGRy/RACEGRyN, TRTxx*/TRxx*, DOSExxU, APHASE*, EOSSTT, DCSREAS, DCTREAS, DTHCAUS, DTHCAUSN, DTHCGRy/DTHCGRyN
>
> **BDS**
>
> TRT*, DOSEU, ATPT/ATPTN, APHASE/APHASEN, APERIOD/APERIODC, ASPER/ASPERC, ARELTMU, PARCATy/PARCATyN, AVALCATy/AVALCAyN, BASECATy/BASECAyN, BASETYPE, CHGCATy/CHGCATyN, PCHGCATy/PCHGCAyN, SHIFTy/SHIFTyN, CRITy, MCRITy, MCRITyML/MCRITyMN

Remember that when you copy SDTM variables into ADaM dataset variable such as ARM/ARMCD/ACTARM/ACTARMCD, that you also need to copy the controlled terminology set as well. This is based on the ADaM principle of harmonization found in the ADaM document that states to, “Maintain the values and attributes of SDTM variables if copied into analysis datasets.” (CDISC 2009, p. 10)

Often people wonder what they should use for a name their controlled terminology. For ADaM variables that have prescribed controlled terminology, you may use the NCI/CDISC CodeList name submission value for the codelist name that is given to you and can be found in the CDISC Library. For example, ADaM variable ADTF, “Analysis Date Imputation Flag,” has the controlled terminology CodeList CDISC submission value name “DATEFL” assigned to it. For user-defined names, in my opinion, it is easiest to create a CodeList name that matches the object as a default when feasible and specific enough. Here are a couple additional considerations when naming your codelists for ADaM objects:

1. Use caution creating a single codelist for a variable that is used across different BDS datasets. For example, instead of creating a single “PARAM” codelist that spans the parameter values of all of your BDS datasets, consider creating dataset specific codelist names such as “PARAMEFF” for a primary efficacy PARAM codelist name.

2. Some may use the optional Define-XML codelist attribute SASFormatName. If you use that, it is often easier if it can match the codelist name itself, but remember that the SASFormatName must be a legal SAS format name. So, “PARAMEF” wouldn’t be a legal codelist name because PARAM is a character variable, but “$PARAMEF” is a legal PARAM based SASFormatName. Also keep in mind the 8 character restriction on SAS format names.
3. Remember, as it states in the *Define-XML Version 2.0 Completion Guidelines* document, your codelists should be comprised of the subset of categorical responses that are possible for the item being presented. So, if your ADaM variable is a logical subset of an SDTM variable, even if origin type is set to “Predecessor,” the codelist in ADaM should be a subset of the SDTM codelist for the remaining values that could have occurred.

   For example, examine this SDTM define.xml file snip for AE data:

Variable	Label / Description	Type	Role	Length or Display Format	Controlled Terms or ISO Format	Origin / Source / Method / Comment
AESEV	Severity	text	Record Qualifier	8	AESEV • "MILD" = "Grade 1; 1" • "MODERATE" = "Grade 2; 2" • "SEVERE" = "Grade 3; 3"	CRF Annotated Case Report Form [6]

   You see that the AE.AESEV variable has the associated AESEV codelist attached to it. Then, imagine that you had an OCCDS adverse event dataset with only severe events where AESEV was mapped to ASEV. However, the codelist for this analysis dataset would be a subset of AESEV, so we apply the ASEV unit codelist for this dataset as follows:

Variable	Label / Description	Type	Length or Display Format	Controlled Terms or ISO Format	Origin / Source / Method / Comment
ASEV	Analysis Severity/Intensity	text	6	ASEV • "SEVERE" = "Grade 3; 3"	Predecessor: AE.AESEV

DATE, TIME, AND DATETIME FORMATTING AND ATTRIBUTES

We know that ADaM dates, times, and datetimes are numeric and that the ADaM Implementation Guide says “Numeric dates, times and datetimes should be formatted, so as to be human-readable with no loss of precision.” (CDISC 2016, p. 16) In the Define-XML specification document under section 4.2.1 “Data Type Considerations” it says, “ADaM date variables, are provided as integers” and also it says that integer length should be, “The largest allowable integer width.” (CDISC 2013, p. 20) So, from that we know that:

1. ADaM dates, times, and datetimes are numeric integers.
2. We need to format the dates, times, and datetimes.
3. The length of an ADaM date, time, and datetime should be the largest allowable integer width.

However, it isn’t clear what format should be applied to these date objects. Also, the concept of integer length is a bit confusing here.

First, let’s look at defining the length for these date, time, and datetime objects. Since we are using SAS transport format files for our submission datasets, I will go ahead and assume for the sake of argument that we are using SAS date, time, and datetime integer values. I will also assume for the sake of

argument that we are looking at data on humans for the past 120 years through the end of this year. If that is the case, reasonable length settings for these date components in define.xml are:

Integer item	Suggested length
Time	5
Date	6
Datetime	11

Note that in most of the CDISC supplied define.xml examples, that these date components have a documented length of 8 set for them. That might be over or under the suggested lengths above. The Define-XML specification for version 2.0 didn't mention what to do with negative signs, but the newer draft Define-XML 2.1 specification makes it clearer that the negative sign needs to be part of the length.

Secondly, if we look at how these dates, times, and datetimes are formatted, we see that these objects are governed by the Define-XML DisplayFormat attribute. The examples of ADaM date formatting in the Define-XML specification typically use "date9." as a date format. That said, there is no rule as to which formats you must use. I am actually not a proponent of "date9." as a date format because I don't find dates formatted such as "31DEC2019" to sort very well nor are they aesthetically pleasing. I prefer date formats that are more logical, such as YYYYMMDD or even the ISO 8601 date formats such as IS8601DT that can mimic the text date formatting of the SDTM --DTC text dates. Whatever date, time, and datetime formats that you choose to use for a define.xml file, I strongly suggest that you remain consistent in your date formatting across the submission. You may even want to create a company standard for this.

ORDER OF DATASETS IN DEFINE.XML FILES

Interestingly, although there is a prescribed order to your ADaM datasets in define.xml, there isn't actually a metadata object to assign dataset order in define.xml. We do have an order attribute for variables within a dataset, where clauses, and codelist however. We also have guidance on how to order datasets as found in the Define-XML version 2.0 specification section 3.4.2:

> 3.4.2. Other Order Considerations for Elements
> For regulatory submissions of ADaM datasets, a standard order of display has not been established. However, the following Class order seems reasonable, as it is consistent with the ordering of Class values in ADaM 2.1:
> - SUBJECT LEVEL ANALYSIS DATASET
> - ADVERSE EVENTS ANALYSIS DATASET – not yet added to the NCI/CDISC Controlled Terminology as of the time of writing this document
> - BASIC DATA STRUCTURE - datasets in alphabetical order by dataset name
> - ADAM OTHER
>
> As additional ADaM dataset classes are defined, it would be reasonable to insert them between Basic Data Structure and ADaM Other. Within each class the datasets should be displayed in ascending alphabetic order by Name as maintained in the value of the ItemGroupDef Name attribute. (CDISC 2013, p. 13)

The above guidance was written before ADaM OCCDS was invented. It might make sense in that case to replace the adverse events analysis dataset reference above with OCCDS. The way define.xml works is that it will present datasets in the order in which they appear in the data stream of the define.xml file. In my opinion, this is another case where it makes sense to have a business rule outside of define.xml metadata for your ADaM dataset specifications where you also specify dataset order. This order attribute at the dataset level, which is outside of what is required for define.xml, will ensure that you sort your datasets properly by the above guidance as well as by what you intend to see in define.xml.

VARIABLE AND PARAMETER LEVEL OBJECT LENGTH CONCERNS

There are two closing concerns for defining the length on variables.

1. How do you define the length on an origin type of “Predecessor” object when the content is a subset of the original content?
2. How do you adhere to the length truncations required in the FDA’s Technical Conformance Guide?

For the first issue, the length of the ADaM variable should be the maximum length of the applicable subset of the predecessor content. For example, even though the length of a SDTM variable may have been 20, if the content copied to ADaM is of maximum length 10, then the length of the new ADaM variable should be 10. This may seem to violate the ADaM principle of harmonization found in the ADaM model document that states to, “Maintain the values and attributes of SDTM variables if copied into analysis datasets.” However, in section 3.1.1 of the ADaM Implementation Guide it states, “To optimize file size, it is permissible that the length of the variables differ (e.g., trailing blanks may be removed).” (CDISC 2016, p. 14)

For the second issue, the current version of the FDA’s Technical Conformance Guide (October 2018) states:

> The allotted length for each column containing character (text) data should be set to the maximum length of the variable used across all datasets in the study except for suppqual datasets. (FDA 2018, p. 7)

This is an unfortunate decision and subsequently creates an operational challenge for industry. Since this paper’s focus is on ADaM specifications and define.xml, I can offer one operational approach to this problem:

1. In your ADaM specifications, specify lengths for numeric variables with the appropriate precision and be a bit liberal with your lengths for the character variables.
2. Near the time of submission, determine what the maximum lengths are on your ADaM character variables.
3. Reset your ADaM specifications for the maximum revised lengths.
4. Regenerate your define.xml (and everything else) based on the revised character variable lengths.

Obviously, the above process depends upon your define.xml and ADaM specifications being inextricably linked. If they are, then you could do most of this length work programmatically. This is yet another reason for having your ADaM specifications and define.xml metadata coming from the same source. Although, in my opinion, the character variable shortening requirement in the Technical Conformance Guide is not well conceived, comes with risk, and is just poor data structure design. I understand the issue, but it would have been better to ask submitters to set reasonable lengths (e.g., not 200) on variables instead of what has been requested.

ANALYSIS RESULTS METADATA

ADaM Analysis Results Metadata, or ARM, is a specification and define.xml challenge to touch on briefly here. ARM is the metadata that describes the statistical reporting results. As of now, ARM isn’t required by the FDA, although there is interest in starting to see it provided with submissions. The Japanese PMDA is looking at requiring it starting in 2020, however. Also, some tools on the market, such as Pinnacle 21 Community edition, do not yet support ARM. Finally, and to be fair, the definition of ARM in the ADaM 2.1 document and the analysis results metadata extension is a good start, but it is not complete enough for a fully functional operational system of analysis results metadata as it would need augmentation.

In my opinion, if you supply ARM at this point in your define.xml, I would limit it to covering your primary and key secondary endpoint results. You might also refer to these results as your “top line tables.”

CONCLUSION

It is my hope that this paper gives some practical guidance and help to those who are specifying ADaM metadata and for how to do that in a Define-XML friendly way. The factual information in this paper is culled from the references below, and I strongly recommend them to the reader as a source of guidance for how to create proper define.xml files for ADaM.

REFERENCES

CDISC. 2009. "Analysis Data Model (ADaM)" Version 2.1. Accessed March 9, 2019. https://www.cdisc.org/system/files/members/standard/foundational/adam/analysis_data_model_v2.1.pdf.

CDISC. 2013. "Define-XML" Version 2.0. Accessed March 9, 2019. https://www.cdisc.org/system/files/members/standard/foundational/define-xml/define_xml_2_0_releasepackage20140424.zip

CDISC. 2015. "Analysis Results Metadata Specification Version 1.0 for Define-XML Version 2" Accessed March 24, 2019. https://www.cdisc.org/system/files/members/standard/foundational/adam/ARM-for-Define-XML.zip.

CDISC. 2016. "Analysis Data Model Implementation Guide" Version 1.1. Accessed March 9, 2019. https://www.cdisc.org/system/files/members/standard/foundational/adam/ADaMIG_v1.1.pdf.

PhUSE. 2018. "Define-XML Version 2.0 Completion Guidelines" Version 1.0 Draft. Accessed March 9, 2019. https://www.phuse.eu/documents//working-groups/deliverables/phuse-define-xml-20-completion-guidelines-v10-draft-for-public-review-19881.pdf.

Food and Drug Administration. 2018. "Study Data Technical Conformance Guide" Accessed March 10, 2019. https://www.fda.gov/downloads/ForIndustry/DataStandards/StudyDataStandards/UCM624939.pdf.

CDISC. 2019. "Stylesheet Library" Accessed March 24, 2019. https://wiki.cdisc.org/display/PUB/Stylesheet+Library.

ACKNOWLEDGMENTS

I would like to thank the following individuals for their review of this text, their enduring patience (with me), and their continued efforts on standardization in our industry: Lex Jansen, Monika Kawohl, Karin LaPann, and Sandra Minjoe

CONTACT INFORMATION

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

Jack Shostak
Duke Clinical Research Institute
jack.shostak@duke.edu