PharmaSUG 2019 - Paper HT-171

Value-Level Metadata Done Properly

Sandra Minjoe, PRA Health Sciences;
Mario Widel, Independent

ABSTRACT

Value-level metadata is nothing mysterious. It is simply a way to describe how a variable is derived when that derivation differs based on some circumstances. When done properly, value-level metadata make a define.xml more reviewer-friendly.

A common use in ADaM for value-level metadata is when AVAL is derived based on PARAM, but this is not the only time to use value-level metadata.

This hands-on training will include examples and exercises of value-level metadata in SDTM Findings, in ADaM BDS, and more. Additionally, it will provide guidance to help attendees decide when to use value-level metadata.

INTRODUCTION

CDISC SDTM and ADaM both include dataset-level, variable-level, and value-level metadata.

Dataset-level metadata is information about each dataset. Examples of dataset-level content include the dataset name, description (or dataset label), class, and structure. Dataset-level metadata helps us understand what type of data is in a dataset, and how one dataset differs from another. In a submission, every dataset must have dataset-level metadata.

Variable-level metadata is information about each variable within each dataset. Examples of variable-level content include the variable name, description (or variable label), type, format, terminology, source, and derivation. Variable-level metadata helps us understand what can be found in each variable within a dataset. In a submission, every variable within every dataset must have variable-level metadata.

Value-level metadata is information about each value (or set of values) within some variables, within some datasets. Unlike dataset-level and variable-level metadata, value-level metadata is not always required. As long as variable-level metadata is sufficient to describe the contents of a variable, value-level metadata is not needed. So how do we know when to include value-level metadata?

This paper first walks through some examples where value-level metadata can be beneficial. It summarizes common features of these examples, so that in other situations you will be able to determine when to make use of value-level metadata.

VALUE-LEVEL METADATA USED IN SDTM FINDINGS

As described in SDTMIG v3.3, SDTM LB is the laboratory findings domain. It is based around the topic variable LBTESTCD, and includes a number of qualifiers that further explain the test. For many collected laboratory tests, all records for a specific laboratory test name and test code contain identical information on many, if not all, qualifiers.

Glucose laboratory data, however, can be collected in several different ways, including:

- Numeric values from a chemistry test, such as 4, in units of mmol/L
- Numeric values from a urinalysis test, such as 0.54, in units of mg/dL
- Categorical values from a urinalysis test, such as +4

In each case, the SDTM LBTESTCD will be “GLUC” and LBTEST will be “Glucose”, but there can be differences in qualifier variables:

- LBCAT might be “CHEMISTRY” or “URINALYSIS”
- LBSPEC might be “SERUM”, “BLOOD”, or “URINE”
- LBMETHOD might be “TEST STRIP” or missing
- LBSTRESN might be populated for numeric tests but missing for categorical ones
- LBORRESU might be “mmol/L” or “mg/dL” for numeric tests, but missing for categorical ones

Table 1 shows an example of some SDTM variables that demonstrate different kinds of glucose data.

LBTEST	LBCAT	LBORRES	LBORRESU	LBSTRESN	LBLOINC	LBSPEC	LBMETHOD
Glucose	CHEMISTRY	72	mg/dL	72	14749-6	SERUM	
Glucose	CHEMISTRY	89	mg/dL	89	2341-6	BLOOD	TEST STRIP
Glucose	URINALYSIS	+4			25428-4	URINE	TEST STRIP
Glucose	URINALYSIS	0.54	mg/dL	0.54	5792-7	URINE	TEST STRIP

Table 1: Example SDTM Glucose Data with LOINC Added

In addition to some differences in qualifier values LBCAT, LBSPEC, LBMETHOD, LBORRESU, and LBSTRESN, the variable LBLOINC also differs. The meaning of the LBLOINC values from Table 1 are:

- 14749-6: A serum chemistry test, with units
- 2341-6: A blood chemistry test strip test, with units
- 25428-4: A urine urinalysis test strip test, with no units
- 5792-7: A urine urinalysis test strip test, with units

Most SDTM variables have straightforward variable metadata. Table 2 shows a snippet of what a define.xml might look like for a few of these LB variables:

Variable	Label	Key	Type	Length	Controlled Terms or Format	Origin	Derivation / Comment
LBTESTCD	Lab Test or Examination Short Name	5	text	7	Laboratory Test Code	Assigned	
LBTEST	Lab Test or Examination Name		text	22	Laboratory Test Name	eDT	
LBCAT	Category for Lab Test	3	text	10		eDT	
LBORRESU	Original Units		text	7	Unit (LBRESU)	eDT	

Table 2: Example define.xml Variable Metadata

Variable-level metadata is sufficient for the variables shown in Table 2. However, variables such as LBORRES and LBSTRESN need different metadata: specifically, the number of significant digits for these variables is based on the units. In this case we must use the combination of variable-level metadata and value-level metadata.

Table 3 shows variable-level metadata for LBORRES:

Variable	Label	Key	Type	Length	Controlled Terms or Format	Origin	Derivation / Comment
LBORRES	Result or Finding in Original Units		text	8		eDT	

Table 3: Example define.xml Variable Metadata that References Value-Level Metadata

Notice the blue underlined variable name LBORRES in Table 3. In a define.xml, this would link to value-level metadata, such as shown in Table 4:

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
LBORRES	LBTESTCD = "GLUC" (Glucose) and LBCAT = "CHEMISTRY" and LBSPEC = "SERUM"	integer	2		eDT	
LBORRES	LBTESTCD = "GLUC" (Glucose) and LBCAT = "CHEMISTRY" and LBSPEC = "BLOOD" and LBMETHOD = "TEST STRIP"	integer	2		eDT	
LBORRES	LBTESTCD = "GLUC" (Glucose) and LBCAT = "URINALYSIS" and LBSPEC = "URINE" and LBMETHOD = "TEST STRIP" and LBLOINC="25428-4"	text	8		eDT	
LBORRES	LBTESTCD = "GLUC" (Glucose) and LBCAT = "URINALYSIS"and LBSPEC = "URINE" and LBMETHOD = "TEST STRIP" and LBLOINC="5792-7"	float	3		eDT	

Table 4: Example define.xml Value-Level Metadata

The value-level metadata shown in Table 5 has one row per unique value within each variable. Much of the information is the same across each row, but the values in the columns titled **Type** and **Length / Display Format** are different on some rows. Note that because the values for both **Type** and **Length / Display Format** are the same on the first two rows, if desired these could be combined into one row of value-level metadata.

Also note that the maximum of all the lengths in Table 4 is what is shown for the variable-level **Length** in Table 3. Define-XML explains that in cases where both numeric and text data are captured in the same variable, as is the case for LBORRES here, at the variable-level the broader value of “text” is to be used.

VALUE-LEVEL METADATA USED IN ADAM BASIC DATA STRUCTURE

The ADaM Basic Data Structure (BDS) is one of the most common CDISC structures to make use of value-level metadata. This is because within a single dataset, one piece of metadata can be very different for each parameter or set of parameters. We’ll walk through some of the ways we can make use of value-level metadata within BDS: parameter-based derivations, derived vs. copied parameters, criteria values that differ based on parameter, and the use of multiple baselines.

PARAMETER-BASED DERIVATIONS

Have you ever created a complicated parameter derivation like the below, in Table 5?

Variable	Label	Type	Length / Display Format	Controlled Terms or Format	Source / Derivation / Comment
PARAM	Parameter	text	100	PARAM_ADLB	Assigned: If (LBTESTCD = GLUC and LBCAT = CHEMISTRY and LBSPEC = SERUM and LBSTRESU = mg/dL) then set PARAM = “Serum Glucose (mg/dL)”; Else if (LBTESTCD = GLUC and LBCAT = CHEMISTRY and LBSPEC = BLOOD and LBMETHOD = TEST STRIP and LBSTRESU = mg/dL) then set PARAM = “Blood Glucose Test Strip (mg/dL)”; Else if (LBTESTCD = GLUC and LBCAT = URINALYSIS and LBMETHOD = TEST STRIP and LBSTRESU is null) then set PARAM = “Urine Glucose Test Strip Categorical”; Else if (LBTESTCD = GLUC and LBCAT = URINALYSIS and LBMETHOD = TEST STRIP and LBSTRESU = mg/dL) then set PARAM = “Urine Glucose Test Strip (mg/dL)”; Else if (LBTESTCD = GFRBSCRT and LBMETHOD = BEDSIDE SCHWARTZ) then set PARAM = “Calculated EGFR Bedside Schwartz”; Otherwise...

Table 5: Example of a Complicated Parameter Derivation

Not only is this difficult to read in a narrow column, some of the tools used to create define.xml won’t even allow character strings this long, or they strip out the formatting to look like the below, in Table 6:

Variable	Label	Type	Length / Display Format	Controlled Terms or Format	Source / Derivation / Comment
PARAM	Parameter	text	100	PARAM_ADLB	Assigned: If (LBTESTCD = GLUC and LBCAT = CHEMISTRY and LBSPEC = SERUM and LBSTRESU = mg/dL) then set PARAM = “Serum Glucose (mg/dL)”; Else if (LBTESTCD = GLUC and LBCAT = CHEMISTRY and LBSPEC = BLOOD and LBMETHOD = TEST STRIP and LBSTRESU =

Variable	Label	Type	Length / Display Format	Controlled Terms or Format	Source / Derivation / Comment
					mg/dL) then set PARAM = “Blood Glucose Test Strip (mg/dL)”; Else if (LBTESTCD = GLUC and LBCAT = URINALYSIS and LBMETHOD = TEST STRIP and LBSTRESU is null) then set PARAM = “Urine Glucose Test Strip Categorical”; Else if (LBTESTCD = GLUC and LBCAT = URINALYSIS and LBMETHOD = TEST STRIP and LBSTRESU = mg/dL) then set PARAM = “Urine Glucose Test Strip (mg/dL)”; Else if (LBTESTCD = GFRBSCRT and LBMETHOD = BEDSIDE SCHWARTZ) then set PARAM = “Calculated EGFR Bedside Schwartz”; Otherwise…

Table 6: Example of a Complicated Parameter Derivation without Formatting

Now imagine you’re a regulatory reviewer trying to make sense of how AVAL is derived. Would you want to try to decipher the content of Table 6?

Value-level metadata allows us to split up those “if” and “else” statements, making it much more readable. Instead of putting all the if statements into a single cell, as was shown in Table 5 and Table 6, variable metadata, such as shown below in Table 7, can instead reference the value-level metadata, such as shown in Table 8:

Variable	Label	Type	Length / Display Format	Controlled Terms or Format	Source / Derivation / Comment
PARAM	Parameter	text	100	PARAM_ADLB	Assigned: See Value Level Metadata

Table 7: Example of a Parameter Reference to Value-Level Metadata

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
PARAM	LBTESTCD=GLUC and LBCAT=CHEMISTRY and LBSPEC=SERUM and LBSTRESU=mg/dL	text	21	PARAM_ADLB	Assigned	Set to "Serum Glucose (mg/dL)"
PARAM	LBTESTCD=GLUC and LBCAT=CHEMISTRY and LBSPEC=BLOOD and LBMETHOD=TEST STRIP and LBSTRESU=mg/dL	text	32	PARAM_ADLB	Assigned	Set to "Blood Glucose Test Strip (mg/dL)"

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
PARAM	LBTESTCD=GLUC and LBCAT=URINALYSIS and LBMETHOD=TEST STRIP and LBSTRESU is null	text	36	PARAM_ADLB	Assigned	Set to "Urine Glucose Test Strip Categorical"
PARAM	LBTESTCD=GLUC and LBCAT=URINALYSIS and LBMETHOD=TEST STRIP and LBSTRESU = mg/dL	text	33	PARAM_ADLB	Assigned	Set to "Urine Glucose Test Strip (mg/dL)"
PARAM	LBTESTCD= GFRBSCRT and LBMETHOD= BEDSIDE SCHWARTZ	text	32	PARAM_ADLB	Assigned	Set to “Calculated EGFR Bedside Schwartz”

Table 8: Example of a Complicated Parameter Derivation as Value-Level Metadata

Notice that in Table 8, the only columns that vary are **Length / Display Format** and **Derivation / Comment**. Since these are all text strings, the largest value across all parameters should be put into the variable-level **Length / Display Format** for PARAM.

The **Derivation/Comment** in the first three rows of Table 8 could be modified to describe it as the concatenation of the mixed case version of LBCAT, LBTEST, LBMETHOD, and, when present, LBSTRESU. However, this derivation does not work for the last row of Table 8, where the word “Calculated” is not part of any existing SDTM variable, and LBSTRESU is not included as part of the parameter even though it exists in SDTM. This means that, if desired, Table 8 could be represented with only two rows instead of the four rows shown. Value-based derivations, whether or not there are differences in other variable metadata, can benefit from the use of value-level metadata, if only to make it more readable.

DERIVED VS. COPIED PARAMETERS

Within a BDS dataset, we may create parameters in different ways. For example, when creating a BDS structure for vital signs data, AVAL might be a copy of SDTM test results for height (PARAMCD = HEIGHT) and weight (PARAMCD = WEIGHT), but derived for Body Mass Index (PARAMCD = BMI) by using those corresponding height and weight values. An example is shown below, with variable-level metadata in Table 9 and value-level metadata in Table 10.

Variable	Label	Type	Length / Display Format	Controlled Terms or Format	Source / Derivation / Comment
AVAL	Analysis Value	integer	8		Derived: See Value Level Metadata

Table 9: Example Vital Signs AVAL Variable-Level Metadata

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
AVAL	PARAMCD in (HEIGHT, WEIGHT)	integer	3		Derived	Set to VS.VSSTRESN where VSTESTCD=PARAMCD
AVAL	PARAMCD=BMI	integer	3		Derived	Set to weight in kg over the square of height in m, using AVAL from PARAMCDs HEIGHT and WEIGHT on the same ADT.

Table 10: Example Vital Signs AVAL Value-Level Metadata

Note that the **Where** clause for first row in Table 10 here is used for both the height and weight parameters ("PARAMCD in (HEIGHT, WEIGHT)"). These parameters are derived the same way, so there is no need to split them out into separate rows. However, if preferred, two separate rows could be used, one for HEIGHT and another for WEIGHT.

Also, because height, weight, and BMI are all generally seen as integers, the **Length / Display Format** column has the same value for all parameters. This means value-level metadata, in this case, is not even required. If no value-level metadata were used, the variable-level metadata would look something like Table 6 or Table 7, with "if" statements to describe each parameter or set of parameters. However, value-level metadata as shown in Table 10 is allowed and can be helpful.

CRITERIA VALUES THAT DIFFER BASED ON PARAMETER

The BDS standard variables CRITy and CRITyFL are used to define a criterion and denote whether that criterion was met on the row. They are always used in pairs. The "y" in the variable names CRITy and CRITyFL is to be replaced with a number, creating variable pairs such as CRIT1 and CRIT1FL. There can be many pairs of these variables within a dataset.

The CDISC Notes in ADaMIG v1.1 for CRITy state that it is "A text string identifying a pre-specified criterion within a parameter, for example SYSBP > 90." If CRIT1 were "SYSBP > 90" then CRIT1FL would be set to "Y" on every row with value of SYSBP greater than 90.

Also explained in ADaMIG v1.1, "CRITy can vary across parameters within a dataset … In other words, CRITy for one parameter can be different than CRITy for a different parameter in the same dataset." This means that we can define CRIT1 a "SYSPBP > 90" for the parameter SYSBP, but for parameter DIASBP we can define CRIT1 as "DIASBP > 150".

There are two different ways to make use of variables CRITy and CRITyFL described in ADaMIG v1.1. Before getting into our examples, let's examine each option.

Option 1: Summarizing Only Records that Meet the Criteria

Let's consider an analysis need showing how far outside the normal range (X, Y) a result is, where X is referred to as the lower limit of normal and Y is the upper limit of normal". One example summary table might be something like shown below in Table 11:

	Trt A	**Trt B**	**Total**
	N=xxx	N=xxx	N=xxx
Number of Subjects with ABC >= 2X upper limit of normal	xxx (xx.x%)	xxx (xx.x%)	xxx (xx.x%)

Table 11: Example Lab Summary Table using Y/null Criteria and Flag

In Table 11, we will only analyze records that meet the criteria. A simple way to create the results for Table 11 is to include, on all records that meet the criteria, variables CRITy with value "ABC >= 2X upper limit of normal" and a corresponding CRITyFL with the value of "Y". Other records for the parameter (those that don't meet the criteria) would have neither CRITy nor CRITyFL populated.

Option 2: Summarizing Records that Meet or do not Meet the Criteria

A different type of analysis is to create a summary table such as shown in Table 12:

	Trt A N=xxx	**Trt B** N=xxx	**Total** N=xxx
ABC >= 2X upper limit of normal			
Yes	xxx (xx.x%)	xxx (xx.x%)	xxx (xx.x%)
No	xxx (xx.x%)	xxx (xx.x%)	xxx (xx.x%)

Table 12: Example Lab Summary Table using Y/N/null Criteria and Flag

In Table 13, we will analyze both the records are at least 2-times the upper limit of normal, and also the records we know are not at least 2-times the upper limit of normal. Thus the CRITyFL used for Table 12 needs to have possible values of "Y", "N" and null, where a null value means we didn't have the data to determine if the result was more than 2-times the upper limit of normal. Here, CRITy would be populated on all rows for the parameter, not just those that meet the criteria.

Example using both Criteria Variable Options

Now let's apply these two different ways of using CRITy and CRITyFL into a more robust example. In this example, we want to analyze the following:

- Count (percent) of subjects with Glucose >=2 times upper limit of normal
- Count (percent) of subjects with Calcium < 0.5 times lower limit of normal
- Count (percent) of subjects with Glucose between 3 and 5 times upper limit of normal or not
- Count (percent) of subjects with Calcium between 3 and 5 times upper limit of normal or not

Even though there are 4 different analyses to be done, two are on Glucose and two are on Calcium. Therefore, only 2 pairs of variables (CRIT1, CRIT1FL, CRIT2, and CRIT2FL) are needed.

Table 13 shows the ADLB variable-level metadata for CRIT1, CRIT1FL, CRIT2, and CRIT2FL.

Variable	**Label**	**Type**	**Length / Display Format**	**Controlled Terms or Format**	**Source / Derivation / Comment**
CRIT1	Analysis Criterion 1	text			Assigned: See Value Level Metadata
CRIT1FL	Criterion 1 Evaluation Result Flag	text	1	["Y" = "Yes"] <No Yes Response - Y subset	Derived: Set to 'Y' when CRIT1 is populated
CRIT2	Analysis Criterion 2	text	48		Assigned: See Value Level Metadata
CRIT2FL	Criterion 2 Evaluation Result Flag	text	1	["N" = "No", "Y" = "Yes"] <No Yes Response>	Derived: See Value Level Metadata

Table 13: Example Variable-Level Metadata for Criteria and Criteria Flags

Notice that most of the variables in Table 13 reference value-level metadata. Let's look at the value-level metadata for CRIT1 (in Table 14), CRIT2 (in Table 15), and CRIT2FL (in Table 16), plus code lists for CRIT1 (in Table 18) and CRIT2 (in Table 19), before seeing why CRIT1FL has no value-level metadata.

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
CRIT1	PARAMCD =GLUC	text	37	CRIT1_ADLB	Derived	Set to "Glucose >=2 times upper limit of normal" when AVAL >= 2 * ANRHI. Set to null otherwise
CRIT1	PARAMCD =CA	text	36	CRIT1_ADLB	Derived	Set to "Calcium < 0.5 times lower limit of normal" when AVAL < 2* ANRLO. Set to null otherwise

Table 14: Example Value-Level Metadata for CRIT1

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
CRIT2	PARAMCD =GLUC	text	48	CRIT2_ADLB	Assigned	Set to "Glucose between 3 and 5 times upper limit of normal"
CRIT2	PARAMCD =CA	text	48	CRIT2_ADLB	Assigned	Set to "Calcium between 3 and 5 times lower limit of normal"

Table 15: Example Value-Level Metadata for CRIT2

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
CRIT2FL	PARAMCD =GLUC	text	1	["N" = "No", "Y" = "Yes"] <No Yes Response>	Derived	If AVAL is missing then set to blank Set to Y if 3 * ANRHI <= AVAL <= 5 * ANRHI ; Otherwise set to N
CRIT2FL	PARAMCD =CA	text	1	["N" = "No", "Y" = "Yes"] <No Yes Response>	Derived	If AVAL is missing then set to blank Set to Y if 3 * ANRLO <= AVAL <= 5 * ANRLO ; Otherwise set to N

Table 16: Example Value-Level Metadata for CRIT2FL

Permitted Value (Code)
Glucose >=2 times upper normal value
Calcium < 0.5 times lower normal value

Table 17: Example Code List for CRIT1

Permitted Value (Code)
Glucose between 3 and 5 times upper normal value
Calcium between 3 and 5 times lower normal value

Table 18: Example Code List for CRIT2

Let’s now consider the CRIT1 and CRIT1FL pair. The analysis need here would be similar to Table 11, where we only care about rows where the criteria is met. As shown in Table 17, the value in CRIT1 will be populated with different text, depending on parameter. Here we populate CRIT1 for the glucose parameter only when the result in AVAL is more at least 2 times upper limit of normal, and we populate it for the calcium parameter only when the result in AVAL is less than 0.5 times lower limit of normal. To clarify these different values and derivations, we create value-level metadata for CRIT1 shown in Table 14. The corresponding CRIT1FL, as shown in Table 13, is exactly the same for both parameters: it is Y whenever CRIT1 is populated, and it is missing otherwise. Because all the variable metadata is the same across both parameters, there is no need to create value-level metadata for CRIT1FL.

The CRIT2 and CRIT2FL pair analysis is different, since we not only need to know records that met the criteria, but also those that did not and those where we don’t have enough information to determine. This analysis need is similar to Table 12. As shown in Table 15, CRIT2 values are set differently for each parameter, but within the parameter the values are the same on all rows. This is unlike CRIT1 from Table 11, where some rows within the parameter are populated and some are not. CRIT2FL has value-level metadata, where the values of “Y”, “N”, or null (missing) are assigned based on the comparison described by parameter in CRIT2. Having a value of “Y”, “N”, or null (missing) on every row, allows us to create the counts needed for a table such as Table 12.

MULTIPLE BASELINES

When more than one baseline is needed for the same parameter, we are not allowed to include as columns variables for more than one baseline, more than one change from baseline, etc. Instead, there are 2 BDS-compliant options:

1. Add a row for each additional baseline and for each row that will make use of that additional baseline.
2. Create a separate dataset for each baseline.

The authors use the following rule to decide which method to use:

- Use Option 1 when the baseline changes as you move through the data, such as in a crossover study where the first treatment period uses one baseline and the second treatment period uses a different baseline.
- Use Option 2 when different baselines are used for all rows, such as one baseline that is the last non-missing value prior to treatment, a second baseline that is the average of all pre-treatment values, and a third baseline that is the “best” pre-treatment value.

Value-level metadata can be useful for Option 1. This option is showcased in ADaMIG v1.1 section 4.2.1 Rule #6. Table 19, below is a similar example:

Row	LBSEQ	EPOCH	APHASE	ABLFL	BASETYPE	AVAL	BASE	CHG	DTYPE
1	111	RUN-IN	RUN-IN	Y	RUN-IN	34.5	34.5	0.0	
2	168	RUN-IN	RUN-IN		RUN-IN	11.6	34.5	22.9	
3	200	RUN-IN	RUN-IN		RUN-IN	13.1	34.5	-21.4	
4	200	RUN-IN	DOUBLE-BLIND	Y	DOUBLE-BLIND	13.1	13.1		LOCF

Row	LBSEQ	EPOCH	APHASE	ABLFL	BASETYPE	AVAL	BASE	CHG	DTYPE
5	295	DOUBLE-BLIND	DOUBLE-BLIND		DOUBLE-BLIND	13.7	13.1	0.6	
6	300	DOUBLE-BLIND	DOUBLE-BLIND		DOUBLE-BLIND	19.7	13.1	6.6	
7	300	DOUBLE-BLIND	OPEN-LABEL	Y	OPEN-LABEL	19.7	19.7		LOCF
8	350	OPEN-LABEL	OPEN-LABEL		OPEN-LABEL	28.1	19.7	8.4	

Table 19: Example of a New Baseline for each Analysis Period

In Table 19, we see multiple records within three different periods, each period blocked with a heavy dark line. Notice that:

- These eight analysis records for a subject parameter were taken from six SDTM records. Rows 3 and 4 are both from SDTM record with LBSEQ = 200, and rows 6 and 7 are both from the SDTM record with LBSEQ = 300.
- SDTM EPOCH is different than ADaM APHASE on the “duplicate” rows. While EPOCH is “RUN-IN” on rows 3 and 4, APHASE is “RUN-IN” on row 3 but “DOUBLE-BLIND” on row 4. Similarly, while EPOCH is “DOUBLE-BLIND” on rows 6 and 7, APHASE is “DOUBLE-BLIND” on row 6 but “OPEN-LABEL” on row 7. In each case, the first of these “duplicate” records represents the last value in one analysis phase, and the second record represents the baseline value for the next analysis phase.
- There are 3 baseline flags for this subject parameter. The first baseline is for the run-in phase, the next baseline is for the double-blind phase, and the last baseline is for the cross-over phase.

Table 20 shows variable-level metadata for the variables shown in Table 19:

Variable	Label	Type	Length / Display Format	Controlled Terms or Format	Source / Derivation / Comment
LBSEQ	Sequence Number	integer	8		Predecessor: LB.LBSEQ
EPOCH	Epoch	text	12		Predecessor: LB.EPOCH
APHASE	Analysis Phase	text	16	<APHASE>	Derived: See Value Level Metadata
ABLFL	Baseline Record Flag	text	1	["Y" = "Yes"] <No Yes Response - Y subset>	Derived: Set to Y for record with last non-missing value of AVAL within the APHASE
BASETYPE	Baseline Type	text	20	BASETYPE_ADLB	Derived: Set to APHASE
AVAL	Analysis Value	float			Predecessor: LB.LBSTRESN
BASE	Baseline Value	float			Derived: AVAL when ABLFL=Y within the same APHASE

Variable	Label	Type	Length / Display Format	Controlled Terms or Format	Source / Derivation / Comment
CHG	Change from Baseline	float			Derived: AVAL - BASE
DTYPE	Derivation Type	text	8		Assigned: Populated with LOCF when the row is derived. See APHASE Value Level metadata for a description of derived rows.

Table 20: Example Variable-Level Metadata Multiple Baseline Example

Most of the variables in Table 19 can be sufficiently described using variable-level metadata in Table 20. It's the creation of additional rows that needs further explanation in value-level metadata. Table 21 shows the value-level metadata for variable APHASE, the only variable in Table 20 that needs it.

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
APHASE	EPOCH = RUN-IN					Set to RUN-IN. For the last record with a non-missing AVAL within the EPOCH, create a new record, copying all content, but with APHASE of DOUBLE-BLIND rather than RUN-IN.
APHASE	EPOCH = DOUBLE-BLIND					Set to DOUBLE-BLIND. For last record with a non-missing AVAL within the EPOCH, create a new record, copying all content, but with APHASE of OPEN-LABEL rather than DOUBLE-BLIND.

Table 21: Example Value-Level Metadata for APHASE in a Multiple Baseline Dataset

This example doesn't include other complications, such as imputations of AVAL for missing data, or the derivation of AVISIT, however, it still demonstrates that putting multiple baselines within a single analysi dataset, including deriving rows and documenting with metadata, can add complexity. In this example, where a maximum of 2 SDTM rows per subject would be "duplicated" (same SDTM data but some differences in analysis variables), the authors think that this method is the best way to convey the progression of subject parameter values through the study.

Now let's consider a different multiple-baseline analysis need, where, as you move through the dataset, each record needs to be analyzed against all possible baselines. In other words

- All run-in records (LBSEQ values of 111, 168, and 200) are to be analyzed using the RUN-IN baseline value of 34.5.
- All double-blind records (LBSEQ values of 295 and 300) are to be analyzed using the RUN-IN baseline value of 34.5 and the DOUBLE-BLIND baseline value of 13.1
- All open-label records (LBSEQ value of 350) need to be analyzed using the RUN-IN baseline value of 34.5, the DOUBLE-BLIND baseline value of 13.1, and the OPEN-LABEL value of 19.7.

In this case, the dataset would need additional rows added for each of these baselines, and would look like Table 22:

Row	LBSEQ	APHASE	ABLFL	BASETYPE	AVAL	BASE	CHG
1	111	RUN-IN	Y	RUN-IN	34.5	34.5	0.0
2	168	RUN-IN		RUN-IN	11.6	34.5	22.9
3	200	RUN-IN		RUN-IN	13.1	34.5	-21.4
4	295	DOUBLE-BLIND		RUN-IN	13.7	34.5	-20.8
5	300	DOUBLE-BLIND		RUN-IN	19.7	34.5	-14.8
6	350	OPEN-LABEL		RUN-IN	28.1	34.5	-6.4
7	200	DOUBLE-BLIND	Y	DOUBLE-BLIND	13.1	13.1	0.0
8	295	DOUBLE-BLIND		DOUBLE-BLIND	13.7	13.1	0.6
9	300	DOUBLE-BLIND		DOUBLE-BLIND	19.7	13.1	6.6
10	350	OPEN-LABEL		DOUBLE-BLIND	28.1	13.1	15.0
11	300	OPEN-LABEL	Y	OPEN-LABEL	19.7	19.7	0.0
12	350	OPEN-LABEL		OPEN-LABEL	28.1	19.7	8.4

Table 22: Example Dataset Structure for Change from All Possible Baseline Values

Notice that Table 23 has 12 rows, twice as many rows as found in the SDTM data! Also notice that:

- Rows 1-6 contain all 6 of the original SDTM records, and are used for the change from RUN-IN baseline. Row 1 AVAL is the baseline for each of these rows.
- Rows 3-6 are “duplicated” as rows 7-10 respectively, with the value of AVAL from row 3 (now row 7) acting as the baseline for this set of rows.
- Rows 5-6 are “duplicated” again as rows 11-12 respectively, with the value of AVAL from row 5 (now row 11) acting as the baseline for this set of rows.

If creating a dataset such as this, value-level metadata would be needed for BASETYPE, using the value of EPOCH to describe how to create each set of rows. This would look similar to Table 21. This metadata is not shown here, because the authors don’t think this is the best approach for this particular analysis need. “Why not?”, you may ask. Here are our reasons:

- It can be confusing to a naïve user to understand why there are so many more records in ADaM than in the corresponding SDTM data.
- It would be unusual to need data about these multiple baselines in the same dataset for analysis. If separate analysis tables are to be created for summaries of change from Run-in Baseline, change from Double-Blind Baseline, and change from Open-label Baseline, then why not instead create separate datasets for each of these?

Shown below is that split data, with Table 23 containing all rows using the run-in baseline, Table 24 containing all rows using the double-blind baseline, and Table 25 containing all rows using the open-label baseline.

Row	LBSEQ	APHASE	ABLFL	AVAL	BASE	CHG
1	111	RUN-IN	Y	34.5	34.5	0.0
2	168	RUN-IN		11.6	34.5	22.9
3	200	RUN-IN		13.1	34.5	-21.4
4	295	DOUBLE-BLIND		13.7	34.5	-20.8

Row	LBSEQ	APHASE	ABLFL	AVAL	BASE	CHG
5	300	DOUBLE-BLIND		19.7	34.5	-14.8
6	350	OPEN-LABEL		28.1	34.5	-6.4

Table 23: Example Dataset with just Change from Run-in Baseline Data

Row	LBSEQ	APHASE	ABLFL	AVAL	BASE	CHG
1	200	DOUBLE-BLIND	Y	13.1	13.1	0.0
2	295	DOUBLE-BLIND		13.7	13.1	0.6
3	300	DOUBLE-BLIND		19.7	13.1	6.6
4	350	OPEN-LABEL		28.1	13.1	15.0

Table 24: Example Dataset with just Change from Double-Blind Baseline Data

Row	LBSEQ	APHASE	ABLFL	AVAL	BASE	CHG
1	300	OPEN-LABEL	Y	19.7	19.7	0.0
2	350	OPEN-LABEL		28.1	19.7	8.4

Table 25: Example Dataset with just Change from Open-Label Baseline Data

Note that because there is exactly one baseline for each of these tables, BASETYPE is not needed in any of the datasets shown in Table 23, Table 24, and Table 25. Also note that:

- Table 23 contains exactly the same number of rows (six) as in SDTM. Here the first row is the baseline value for all the rest of the rows.
- Table 24 contains only the row used as baseline (LBSEQ = 200), plus all rows in the double-blind (two rows) and open-label (one row) periods of the trial.
- Table 25 contains only the row used as baseline (LBSEQ = 300) plus the one open-label row.

Table 26 shows how dataset-level metadata can explain the differences and purpose of each dataset:

Dataset	Description	Class	Structure	Purpose	Keys	Location	Documentation
ADLBRI	Laboratory Results Run-In Analysis	BASIC DATA STRUCTURE	one record per subject per parameter per timepoint	Analysis	USUBJID, PARAM, AVISIT	adlbri.xpt	Uses run-in Baseline for change-from-baseline analyses
ADLBDB	Laboratory Results Double-Blind Analysis	BASIC DATA STRUCTURE	one record per subject per parameter per timepoint	Analysis	USUBJID, PARAM, AVISIT	adlbdb.xpt	Uses double-blind Baseline for change-from-baseline analyses
ADLBOL	Laboratory Results Open-Label Analysis	BASIC DATA STRUCTURE	one record per subject per parameter per timepoint	Analysis	USUBJID, PARAM, AVISIT	adlbol.xpt	Uses open-label Baseline for change-from-baseline analyses

Table 26: Example Dataset Metadata for Multiple Lab Analysis Datasets

Another common case for multiple baselines is in a simple parallel design study, where all records after the first dose of study drug are analyzed two ways: once using baseline as the last non-missing value prior to treatment, and another using the average of all pre-treatment values. Similar to the method described for creating Table 23, Table 24, and Table 25 above, the authors recommend a separate dataset for each of these datasets (rather than basically doubling the amount of records to push it into a single dataset, plus creating value-level metadata to explain how to derive BASETYPE for each baseline). Similar to Table 26, dataset-level metadata such as in Table 27 would be used to explain the differences and purpose of each dataset.

Dataset	Description	Class	Structure	Purpose	Keys	Location	Documentation
ADLBLAST	Laboratory Results Analysis - BL = Last	BASIC DATA STRUCTURE	one record per subject per parameter per timepoint	Analysis	USUBJID, PARAM, AVISIT	adlblast.xpt	Uses last non-missing value prior to treatment as Baseline for change-from-baseline analyses
ADLBAVG	Laboratory Results Analysis - BL = Avg	BASIC DATA STRUCTURE	one record per subject per parameter per timepoint	Analysis	USUBJID, PARAM, AVISIT	adlbavg.xpt	Uses average of all values prior to treatment as Baseline for change-from-baseline analyses

Table 27: Second Example Dataset Metadata for Multiple Lab Analysis Datasets

Where multiple datasets are used, each with a single definition of baseline, dataset-level metadata such as in Table 26 and Table 27, rather than value-level metadata, is used to describe the different baselines.

VALUE-LEVEL METADATA USED IN ADAM OCCURRENCE DATA STRUCTURE

Our prior examples included value-level metadata for SDTM finding and ADaM BDS, so it might be tempting to think that value-level metadata can only apply to tests and parameters. While tests and parameters are common needs for value-level metadata, it is possible to use value-level metadata for other breakdowns. In this section we apply it to an Adverse Event dataset in the ADaM OCCDS structure.

BDS variables like PARAM and AVAL are not used in ADaM OCCDS. OCCDS, in the simplest case, can be thought of “SDTM-plus”, because it looks a lot like an SDTM Events or Interventions domain, but adding supplemental qualifiers, ADSL content, and a few derived variables.

For adverse events, we might copy in SDTM AE data, such as the MedDRA hierarchy and timing variables, transpose and merge any useful SUPPAE content, merge core ADSL variables, and derive things like numeric dates and treatment-emergent flag. Analysis is done using the unchanged MedDRA coding variables, and subset by variables such as flags for population and treatment-emergent AEs.

Things can get trickier when, in addition to the primary coding hierarchy, analysis is needed using a secondary hierarchy. Figure 1 is a copy of a diagram from OCCDS v1.1, which shows that the term “dizziness” falls within three different MedDRA hierarchy paths.

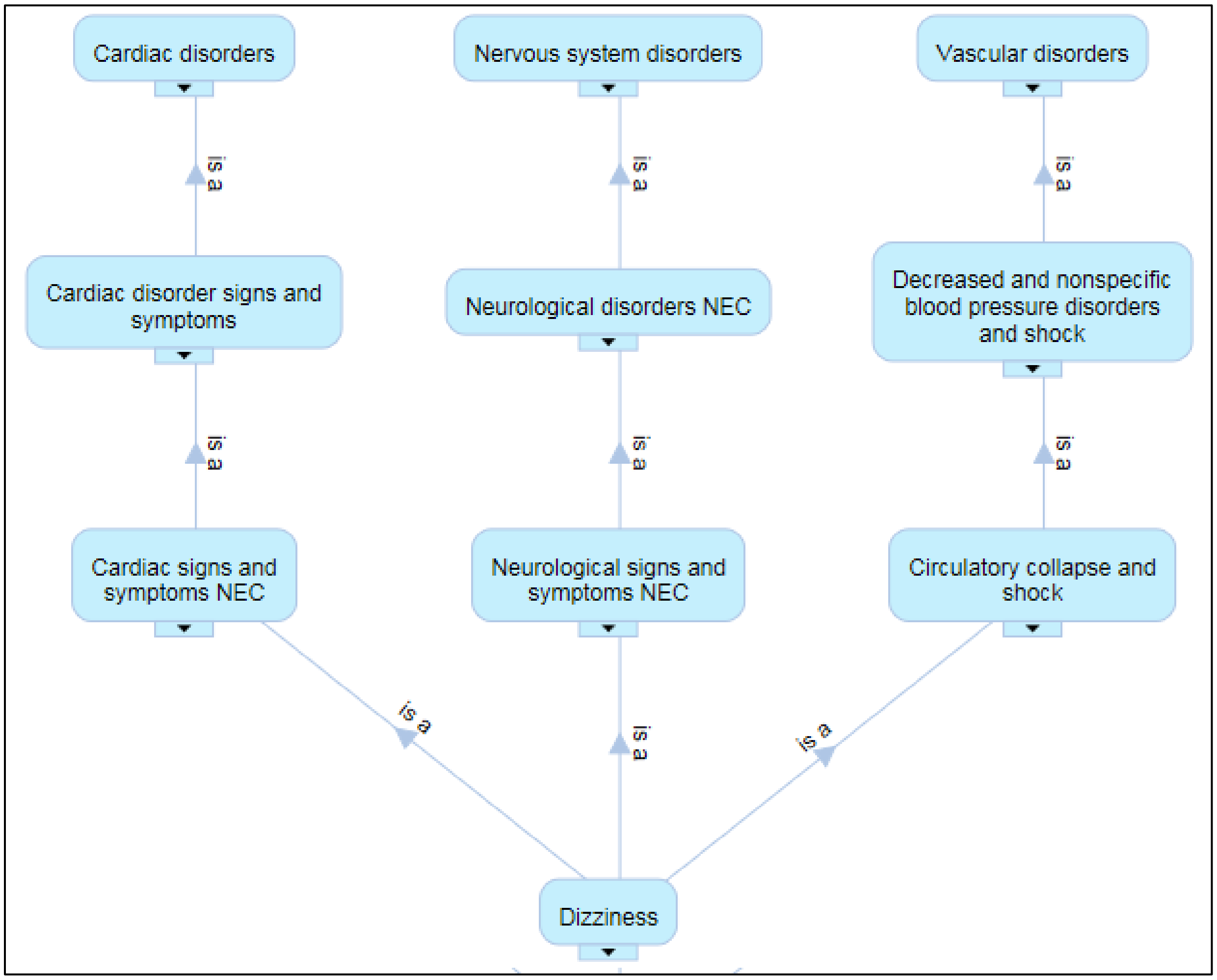
Cardiac disorders
Nervous system disorders
Vascular disorders
is a
is a
is a
Cardiac disorder signs and symptoms
Neurological disorders NEC
Decreased and nonspecific blood pressure disorders and shock
is a
is a
is a
Cardiac signs and symptoms NEC
Neurological signs and symptoms NEC
Circulatory collapse and shock
is a
is a
is a
Dizziness

Figure 1: Possible Coding Paths for Dizziness

As shown in Figure 1, dizziness could be related to the vascular, cardiac, or nervous system. The primary path for dizziness is **Nervous system disorders**. If the study indication or drug class is known to affect a different area of the body, such as if the indication under study is something involving the cardiac system, then dizziness might need to be analyzed in two ways, once for the Nervous System path, and separately for the Cardiac Symptom path.

SDTM allows for the capture of a second path. Basically, AESOC and the bulk of the MedDRA variables are for the primary path, but AEBODSYS is used to contain the body system (also know as system organ class) for the secondary path. The rest of the hierarchy for the secondary path is not included in SDTM AE, but this can be put into SUPPAE.

Table 28 shows example SDTM data with Dizziness mapped in the primary MedDRA path (AESOC) to Nervous system disorders and in the secondary MedDRA path (AEBODSYS) to Cardiac disorders:

USUBJID	AESEQ	AEDECOD	AEBODSYS	AESTDTC	AESOC
XYZ-1-001	1	Autoimmune thyroiditis	Endocrine disorders	2008-05-13	Endocrine disorders
XYZ-1-001	2	Dizziness	Cardiac disorders	2008-06-13	Nervous system disorders
XYZ-2-002	3	Dizziness	Cardiac disorders	2008-09-13	Nervous system disorders
XYZ-3-003	4	Thyroid atrophy	Endocrine disorders	2008-09-13	Endocrine disorders

USUBJID	AESEQ	AEDECOD	AEBODSYS	AESTDTC	AESOC
XYZ-4-004	5	Dizziness	Cardiac disorders	2008-09-09	Nervous system disorders

Table 28: Example Input SDTM AE Dataset with Secondary MedDRA Coding

There are different ways we can arrange this data in order to perform the analysis of the primary path and the analysis of the secondary path. Here are the options, as we see them:

1. Short-wide dataset
2. Separate dataset for each hierarchy
3. Tall-skinny dataset

Let’s talk through the details of each, and where it would be valuable to have value-level metadata.

Option 1: Short-wide dataset

This is probably the most intuitive. Assembly and use of the dataset would look something like this:

- Bring in data as-is from SDTM AE
- Transpose and attach all needed SUPPAE secondary path hierarchy variables
- Add ADSL variables and derive any other variables
- Use AESOC and the standard hierarchy variables for the primary path analysis
- Use AEBODSYS and the hierarchy brought in from SUPPAE for the secondary path analysis

In this case, variable derivations provide good traceability between SDTM and ADaM, and no additional value-level metadata is needed. However, traceability between ADaM and the analysis tables is going to be challenging. Which sets of variables are used for what tables? Analysis Results Metadata would help, but most companies are not currently implementing it. Which brings us to our next option.

Option 2: Separate dataset for each hierarchy

Instead of putting hierarchy for each MedDRA path together in the same dataset, we split them out: one dataset per hierarchy. Here we would create a primary path dataset, using most of the standard SDTM AE hierarchy variables, and a secondary hierarchy dataset using the AEBODSYS and SUPPAE hierarchy variables. Table 29, with no secondary hierarchy variables, would work as the basis of the primary path dataset. Table 30, with no primary hierarchy variables, would work as basis the secondary path dataset.

USUBJID	AESEQ	AEDECOD	AESOC	AESTDTC
XYZ-1-001	1	Autoimmune thyroiditis	Endocrine disorders	2008-05-13
XYZ-1-001	2	Dizziness	Nervous system disorders	2008-06-13
XYZ-2-002	3	Dizziness	Nervous system disorders	2008-09-13
XYZ-3-003	4	Thyroid atrophy	Endocrine disorders	2008-09-13
XYZ-4-004	5	Dizziness	Nervous system disorders	2008-09-09

Table 29: Example Analysis Dataset with only MedDRA Primary Hierarchy Data

STUDYID	USUBJID	AESEQ	AEDECOD	AEBODSYS	AESTDTC
XYZ	XYZ-1-001	2	Dizziness	Cardiac disorders	2008-06-13
XYZ	XYZ-2-002	3	Dizziness	Cardiac disorders	2008-09-13
XYZ	XYZ-4-004	5	Dizziness	Cardiac disorders	2008-09-09

Table 30: Example Analysis Dataset with only MedDRA Secondary Hierarchy Data

Assembly and use of the primary path dataset would look something like this:

- Bring in data as-is from SDTM AE, other than AEBODSYS (a secondary hierarchy variable)
- Do not transpose and attach any secondary hierarchy variables from SUPPAE
- Add ADSL variables and derive any other variables
- Use AESOC and the standard hierarchy variables for the primary path analysis

Assembly and use of the secondary path dataset would look something like this:

- Bring in data as-is from SDTM AE, including AEBODSYS, other than primary hierarchy variables
- Transpose and attach all needed SUPPAE secondary hierarchy variables
- Add ADSL variables and derive any other variables
- Decided whether to drop or keep any record with the same value in AEBODSYS and AESOC. (Note: we dropped them in our example, but they may be kept if needed for analysis.)
- Use AEBODSYS and hierarchy from SUPPAE for the secondary path analysis

In this case, we have 2 datasets, and would need to make use of dataset-level metadata to explain the differences. Unlike the short-wide options, traceability between ADaM and the analysis tables is straightforward, since we simply need to know is which dataset was used.

However, we've now lost the ability to see both sets of hierarchy data in the same dataset. If having the data together in a single dataset is needed or helpful, that brings us to our last option.

Option 3: Tall-skinny dataset

An advantage of Option 1 is that all the data are together in the same dataset. An advantage of Option 2 is that it is easy to see which hierarchy is used for each table. Option 3 is a way to achieve both, basically by stacking the 2 datasets in Option 2 and creating analysis versions of the hierarchy values. Assembly and use of the combined dataset would look something like this:

- Bring in data as-is from SDTM AE, other than the hierarchy variables
- Create analysis versions of the System Organ Class hierarchy variable and analysis flags such as shown in variable-level metadata in Table 31, plus ASOC value-level metadata such as shown in Table 32
- Add any other hierarchy variables needed, such as in Table 33 for high-level group term
- Add ADSL variables and derive any other variables
- Use the analysis hierarchy variables, such as ASOC and AHLT, and analysis flags ANL01FL for primary path analysis or ANL02FL for secondary path analysis

Variable	Label	Type	Length / Display Format	Controlled Terms or Format	Source / Derivation / Comment
AETERM	Reported Term for the Adverse Event	text	200		Predecessor: AE.AETERM
AEDECOD	Dictionary-Derived Term	text	200	MedDRA	Predecessor: AE.AEDECOD
ASOC	Analysis System Organ Class	text	100	MedDRA	Derived: See Value Level Metadata

Variable	Label	Type	Length / Display Format	Controlled Terms or Format	Source / Derivation / Comment
AHLT	Analysis High-Level Group Term	text	100	MedDRA	Derived: See Value Level Metadata
ANL01FL	Analysis Flag 01 – Primary Path	text	1	["Y" = "Yes"] <No Yes Response - Y subset>	Derived: Populated with Y when the row is not derived. See ASOC value level metadata for a description of derived rows.
ANL02FL	Analysis Flag 02 – Secondary Path	text	1	["Y" = "Yes"] <No Yes Response - Y subset>	Derived: Populated with Y when the row is derived. See ASOC value level metadata for a description of derived rows.

Table 31: Example Variable-Level Metadata for Tall-Skinny Dataset with Multiple MedDRA Hierarchies

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
ASOC	AE.AESOC = AE.AEBODSYS	text	100	MedDRA	AE.AESOC	Set to AE.AESOC
ASOC	AE.AESOC NE AE.AEBODSYS	text	100	MedDRA		Set ASOC to AE.AESOC. Create a new record, copying all content, but with ASOC set to AE.AEBODSYS.

Table 32: Example Value-Level Metadata for ASOC

Variable	Where	Type	Length / Display Format	Controlled Terms or Format	Origin	Derivation / Comment
AHLT	AE.AESOC = AE.AEBODSYS	text	100	MedDRA	AE.AEHLT	Set to AE.AEHLT
AHLT	AE.AESOC NE AE.AEBODSYS	text	100	MedDRA		For rows where ASOC=AE.AESOC, set AHLT to AE.AEHLT; Otherwise, set AHLT to QVAL for the SUPPAE record with QNAM = <sponsor-defined value>

Table 33: Example Value-Level Metadata for AHLT

In Table 34, rows 2 and 3 are from the SDTM record with AESEQ=2, rows 4 and 5 are from the SDTM record with AESEQ=3, and rows 7 and 8 are from the SDTM record with AESEQ=5.

Row	USUBJID	AESEQ	AEDECOD	ASOC	AESTDTC	ANL01FL	ANL02FL
1	XYZ-1-001	1	Autoimmune thyroiditis	Endocrine disorders	2008-05-13	Y	
2	XYZ-1-001	2	Dizziness	Nervous system disorders	2008-06-13	Y	
3	XYZ-1-001	2	Dizziness	Cardiac disorders	2008-06-13		Y
4	XYZ-2-002	3	Dizziness	Nervous system disorders	2008-09-13	Y	
5	XYZ-2-002	3	Dizziness	Cardiac disorders	2008-09-13		Y
6	XYZ-3-003	4	Thyroid atrophy	Endocrine disorders	2008-09-13	Y	
7	XYZ-4-004	5	Dizziness	Nervous system disorders	2008-09-09	Y	
8	XYZ-4-004	5	Dizziness	Cardiac disorders	2008-09-09		Y

Table 34: Example Tall-Skinny Analysis Dataset with Primary and Secondary MedDRA Hierarchies

Analysis flags are used to signify which rows are used for the primary path analysis (ANL01FL=Y) and the secondary path analysis (ANL02FL=Y). Note that the label for ANL01FL is "Analysis Flag 01 – Primary Path" and the label for ANL02FL is "Analysis Flag 02 – Secondary Path". Thus, using this option, we have good traceability back to SDTM, but also good traceability between the dataset and the tables. If secondary path analysis needed to also include the MedDRA terms that only map to a single path, such as Table 34 rows 1 and 6, then the derivation of ANL02FL would be changed to set the value to Y on these rows, with no new rows added.

The use of value-level metadata for Adverse Event and other OCCDS data is not common, but it is allowed. Use it when it will help with clarity and traceability.

CONCLUSION

Value-level metadata is nothing mysterious. It is simply a way to describe how a variable is derived when that derivation differs based on some circumstances.

Value-level metadata is required in cases where the values of metadata differ based on some condition. Examples were shown of different values of metadata for length, display format, controlled terminology, and derivation. When only the derivation differs, variable-level metadata with qualifiers, like "if", "when" or "where", can instead be used; however value-level metadata in these cases can still be used to clarify the derivation by splitting it across multiple rows.

When done properly, value-level metadata make a define.xml more reviewer-friendly.

REFERENCES

CDISC. 2019. "SDTMIG v3.3" Accessed March 17, 2019. https://www.cdisc.org/standards/foundational/sdtmig/sdtmig-v3-3.

CDISC. 2014. "Define-XML v2.0" Accessed March 25, 2019. https://www.cdisc.org/standards/data-exchange/define-xml.

CDISC. 2009. "ADaMIG v1.1" Accessed April 4, 2019. https://www.cdisc.org/standards/foundational/adam.

CDISC. 2016. "ADaM Structure for Occurrence Data (OCCDS) Version 1.0" Accessed April 23, 2019. https://www.cdisc.org/standards/foundational/adam.

CONTACT INFORMATION

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

Sandra Minjoe
minjoesandra@prahs.com

Mario Widel
mhwidel@gmail.com