SDTM-ETL 5.0 User Manual and Tutorial

Author: Jozef Aerts, XML4Pharma

Last update: 2025-02-09

Tutorial: Working with hypervertical structures

Table of Contents

Introduction 1
An ODM example with the classic structure 1
An ODM example of a hypervertical structure 3
Treatment of classic ODM and of "hypervertical ODM" in SDTM-ETL 5
  Classic ODM 5
  Hypervertical ODM 7
Generating the mappings for LBTESTCD and LBTEST 10
  Generating the mapping for LBTESTCD 10
  Generating the mapping for LBTEST 27
Getting the collected values - Using a relative path as an alternative 28
Developing mappings and assigning values for LBSPEC, LBSTRESC, LBSTRESN, LBSTRESU 34
Timing Variables and date and time formatting 34
Conclusions 38

Introduction

Hypervertical structures, in database theory also entity-attributes-value (EAV) tables, with rows containing a parameter name (key), parameter label/description and parameter value, are first somewhat unusual when generating SDTM, this although SDTM itself, at least for the Findings datasets, is also a hypervertical structure (-TESTCD, -TEST, -ORRES).

Whereas in classic ODM, data is organized hierarchically in visits (StudyEvent), forms (Form), subforms (ItemGroup) and the data themselves (Item), for hypervertical structures, the visit, the form and (when applicable) the subform are just fields from the table.

Especially in Phase 1 studies, one will find such data structures, especially when the ODM is generated from e.g. CSV or SAS files, e.g. as generated using the ODMGenerator software.

In SDTM-ETL 4.2, we added new features and wizards for making it easier to work with such hypervertical structures.

An ODM example with the classic structure

In classic ODM, all data is organized per subject visit, per form, per subform, per datapoint. For example:

```
<SubjectData SubjectKey="001">
    <StudyEventData StudyEventOID="BASELINE">
        <FormData FormOID="F_BASELINE">
            <ItemGroupData ItemGroupOID="IG_COMMON">
                <ItemData ItemOID="I_SITE" Value="23"/>
                <ItemData ItemOID="I_SUBJECTID" Value="001"/>
                <ItemData ItemOID="I_VISIT" Value="2010-02-27"/>
                <ItemData ItemOID="I_VISITTIME" Value="10:27:33"/>
            </ItemGroupData>
            <ItemGroupData ItemGroupOID="IG_DM">
                <ItemData ItemOID="I_BRTHDT" Value="1957-05-07"/>
                <ItemData ItemOID="I_SEX" Value="F"/>
                <ItemData ItemOID="I_RACE" Value="CAUCASIAN"/>
            </ItemGroupData>
            <ItemGroupData ItemGroupOID="IG_SH">
                <ItemData ItemOID="I_SMOKING" Value="true"/>
                <ItemData ItemOID="I_NR_CIGARETTES" Value="LT10"/>
            </ItemGroupData>
            <ItemGroupData ItemGroupOID="IG_DH">
                <ItemData ItemOID="I_DRINKING" Value="1-2"/>
            </ItemGroupData>
            <ItemGroupData ItemGroupOID="IG_PE_BASE">
                <ItemData ItemOID="I_HEIGHT" Value="193">
                    <MeasurementUnitRef MeasurementUnitOID="MU_CM"/>
                </ItemData>
                <ItemData ItemOID="I_WEIGHT" Value="90">
                    <MeasurementUnitRef MeasurementUnitOID="MU_KG"/>
                </ItemData>
                <ItemData ItemOID="I_SYSBP" Value="120">
                    <MeasurementUnitRef MeasurementUnitOID="MU_MMHG"/>
                </ItemData>
                <ItemData ItemOID="I_DIABP" Value="80">
```

This means that for each data point, there is only a single "ItemData", with the "ItemOID" being the identifier (pointing to an "ItemDef"), and the value of the data point located in the "Value" attribute. For example:

```
<ItemData ItemOID="I_WEIGHT" Value="90">
    <MeasurementUnitRef MeasurementUnitOID="MU_KG"/>
</ItemData>
```

In this case, also the unit of measurement is within the same "ItemData" element as a child "MeasurementUnitRef" element, pointing to a "MeasurementDef" in the metadata.
The metadata then are:

```
▼<ItemDef DataType="float" Length="5" SignificantDigits="1" Name="Weight" OID="I_WEIGHT" SASFieldName="WEIGHT" SDSVarName="VSORRES">
  ▼<Question>
     <TranslatedText xml:lang="en">Weight</TranslatedText>
     <TranslatedText xml:lang="fr">Poids</TranslatedText>
     <TranslatedText xml:lang="de">Gewicht</TranslatedText>
     <TranslatedText xml:lang="ko"/>
   </Question>
   <!--  Weight is expected to come either in pounds or in kilogram   -->
   <MeasurementUnitRef MeasurementUnitOID="MU_POUNDS"/>
   <MeasurementUnitRef MeasurementUnitOID="MU_KG"/>
  ▼<RangeCheck Comparator="LT" SoftHard="Hard">
     <CheckValue>150</CheckValue>
     <MeasurementUnitRef MeasurementUnitOID="MU_KG"/>
    ▼<ErrorMessage>
       <TranslatedText xml:lang="en">The weight value should be below 150 kg</TranslatedText>
       <TranslatedText xml:lang="fr">La valeur du poids doit être en dessous de 150 kilo</TranslatedText>
       <TranslatedText xml:lang="de">das Gewicht sollte unter 150 Kg liegen</TranslatedText>
       <TranslatedText xml:lang="ko"/>
     </ErrorMessage>
   </RangeCheck>
  ▼<RangeCheck Comparator="LT" SoftHard="Hard">
     <CheckValue>300</CheckValue>
     <MeasurementUnitRef MeasurementUnitOID="MU_POUNDS"/>
    ▼<ErrorMessage>
       <TranslatedText xml:lang="en">The weight should be below 300 Pounds</TranslatedText>
       <TranslatedText xml:lang="fr">La valeur du poids doit être en dessous de 300 livres</TranslatedText>
       <TranslatedText xml:lang="de">das Gewicht sollte unter 300 Pfund liegen</TranslatedText>
       <TranslatedText xml:lang="ko"/>
     </ErrorMessage>
   </RangeCheck>
```

even showing e.g. rangechecks

and for the unit of measurement:

```
▼<MeasurementUnit Name="Kilograms" OID="MU_KG">
  ▼<Symbol>
     <TranslatedText xml:lang="en">kg</TranslatedText>
     <TranslatedText xml:lang="fr">Kilo</TranslatedText>
     <TranslatedText xml:lang="de">Kg</TranslatedText>
     <TranslatedText xml:lang="ko">킬로그램</TranslatedText>
   </Symbol>
   <Alias Context="UCUM" Name="kg"/>
 </MeasurementUnit>
```

also nicely demonstrating the "internationalization"

An ODM example of a hypervertical structure

The following figure shows a snapshot of an ODM structure which was obtained from a CSV file using the ODMGenerator software:

```
<ItemGroupDef OID="IG.DEFAULT" Name="Default subform" Repeating="Yes">
    <ItemRef ItemOID="IT.StudyID" Mandatory="No"/>
    <ItemRef ItemOID="IT.SubjectNr" Mandatory="No"/>
    <ItemRef ItemOID="IT.AssessmentNumber" Mandatory="No"/>
    <ItemRef ItemOID="IT.PlannedAssessmentTime" Mandatory="No"/>
    <ItemRef ItemOID="IT.ActualAssessmentTime" Mandatory="No"/>
    <ItemRef ItemOID="IT.AssessmDateTime" Mandatory="No"/>
    <ItemRef ItemOID="IT.AssessmDate" Mandatory="No"/>
    <ItemRef ItemOID="IT.AssessmTime" Mandatory="No"/>
    <ItemRef ItemOID="IT.AssessmPerfDatetime" Mandatory="No"/>
    <ItemRef ItemOID="IT.ActivityName" Mandatory="No"/>
    <ItemRef ItemOID="IT.ActivityComments" Mandatory="No"/>
    <ItemRef ItemOID="IT.ParameterName" Mandatory="No"/>
    <ItemRef ItemOID="IT.ParameterDescription" Mandatory="No"/>
    <ItemRef ItemOID="IT.ParameterValue" Mandatory="No"/>
    <ItemRef ItemOID="IT.NumericResult" Mandatory="No"/>
    <ItemRef ItemOID="IT.CharacterResult" Mandatory="No"/>
    <ItemRef ItemOID="IT.ParameterValueNumericFormatted" Mandatory="No"/>
    <ItemRef ItemOID="IT.CharacterFormattedResult" Mandatory="No"/>
    <ItemRef ItemOID="IT.Unit" Mandatory="No"/>
    <ItemRef ItemOID="IT.SasFormat" Mandatory="No"/>
    <ItemRef ItemOID="IT.SasInformat" Mandatory="No"/>
```

describing the metadata of what was originally a row in the CSV file.

The codelist generated by the ODMGenerator for the parameter name is then e.g.:

```
<CodeList OID="CL.IT.ActivityName" Name="CL.IT.ActivityName" DataType="text">
    <CodeListItem CodedValue="Arrival">
        <Decode>
            <TranslatedText xml:lang="en">Arrival</TranslatedText>
        </Decode>
    </CodeListItem>
    <CodeListItem CodedValue="COVID_prot">
        <Decode>
            <TranslatedText xml:lang="en">COVID-19 protection</TranslatedText>
        </Decode>
    </CodeListItem>
    <CodeListItem CodedValue="Cov_Precheck">
```

A data record is then e.g.:

```
<ItemGroupData ItemGroupOID="IG.DEFAULT" ItemGroupRepeatKey="91">
    <ItemData ItemOID="IT.StudyID" Value="MyStudyHV"/>
    <ItemData ItemOID="IT.SubjectNr" Value="3"/>
    <ItemData ItemOID="IT.AssessmentNumber" Value="0"/>
    <ItemData ItemOID="IT.PlannedAssessmentTime" Value="22"/>
    <ItemData ItemOID="IT.ActualAssessmentTime" Value="U"/>
    <ItemData ItemOID="IT.AssessmDateTime" Value="09NOV22:16:02:00"/>
    <ItemData ItemOID="IT.AssessmDate" Value="09NOV2022"/>
    <ItemData ItemOID="IT.AssessmTime" Value="16:02:00"/>
    <ItemData ItemOID="IT.AssessmPerfDatetime" Value="09NOV2022:16:02:00"/>
    <ItemData ItemOID="IT.ActivityName" Value="BsChem_A"/>
    <ItemData ItemOID="IT.ParameterName" Value="ALAT"/>
    <ItemData ItemOID="IT.ParameterDescription" Value="Alanine Aminotransferase (GPT)"/>
    <ItemData ItemOID="IT.ParameterValue" Value="13"/>
    <ItemData ItemOID="IT.NumericResult" Value="13"/>
    <ItemData ItemOID="IT.ParameterValueNumericFormatted" Value="13"/>
    <ItemData ItemOID="IT.Unit" Value="U/L"/>
```

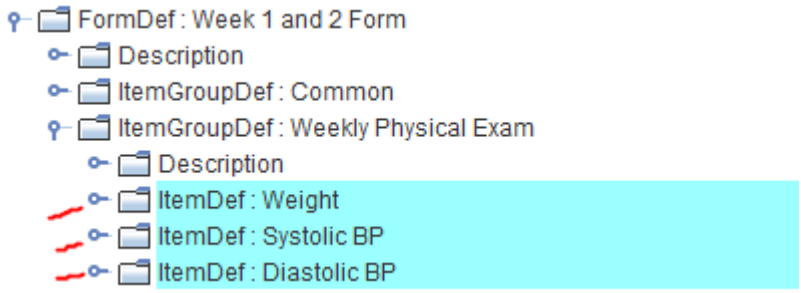
Where the parameter name is "Alanine Aminotransferase", coming from a group "BsChem_A" (Blood Serum Chemistry), with a result of 13 U/L.

This means that the information about a single data point is now not within a single "ItemData" anymore as in classic ODM, but spread over different "ItemData" elements within the same "ItemGroupData", in our case over "Activity Name", "Parameter Name", "Parameter Value" and "Unit". Typical for such structures is also that information is redundant. E.g. "Study ID", and "Subject Number" will also be found at higher levels of the ODM, i.e. in "ClinicalData/@StudyOID" and in "SubjectData/@SubjectKey".

Treatment of classic ODM and of "hypervertical ODM" in SDTM-ETL

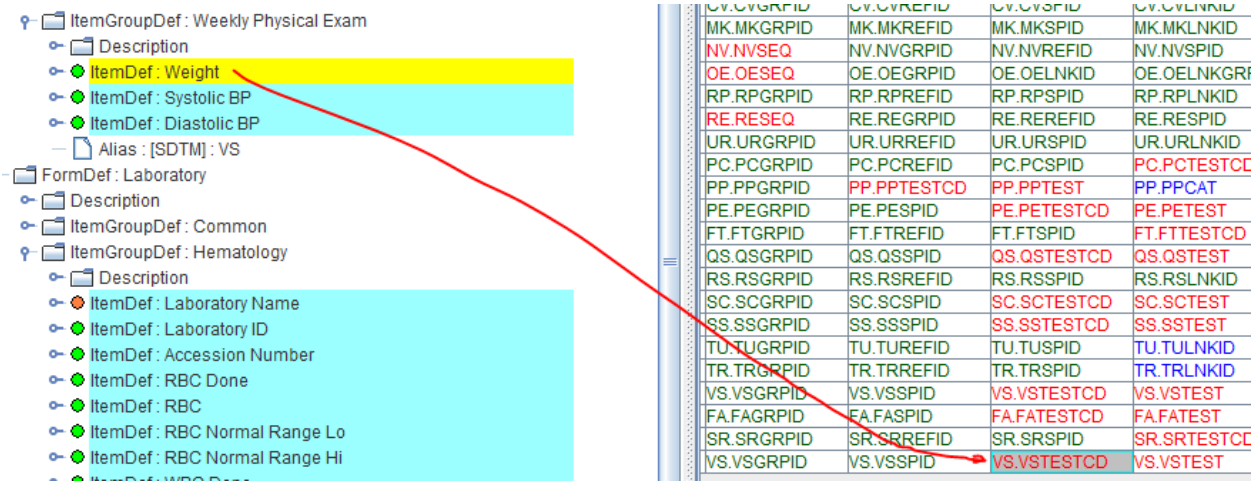
Classic ODM

In "**classic**" ODM, one will see a single tree item for each data point definition on the left side. For example:

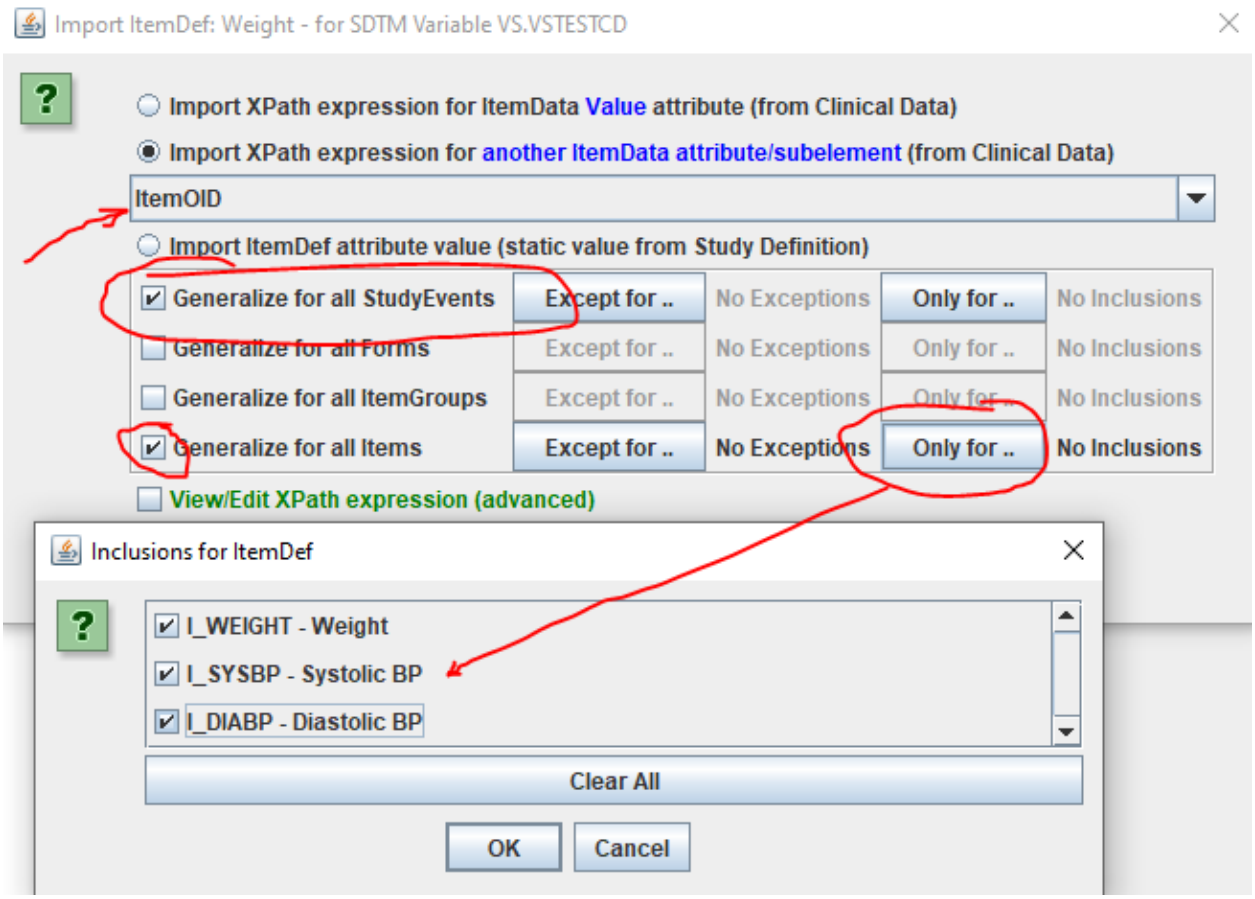


defining 3 data points "weight", "systolic blood pressure" and "diastolic blood pressure".
For each of them, when applicable, one or more allowed units of measure can be defined:

For mapping to SDTM. one can then simply drag-and-drop one of the tree items to the "VSTESTCD" cell:

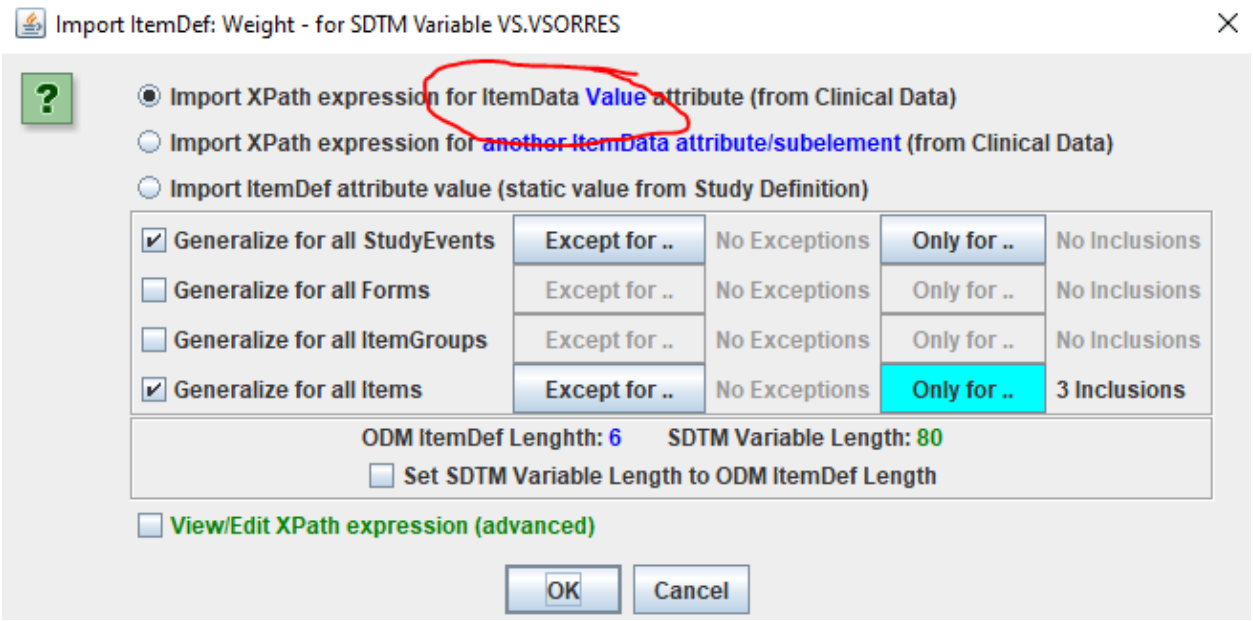


and then, with the wizard that is showing up, indicate that one want to have an SDTM row for every visit and also for "Systolic BP" and for "Diastolic BP", with the identifier of the item ("ItemOID") being mapped to the controlled terminology for VSTESTCD:



This then starts the "Mapping Wizard", as explained in many other of our tutorials.

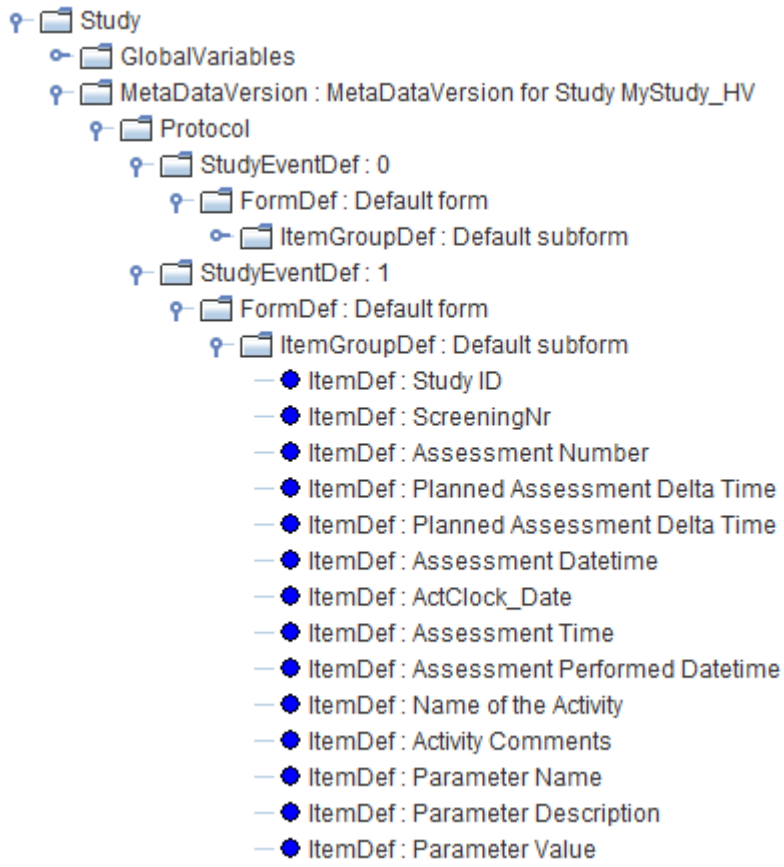
For VSORRES, one will then do the same drag-and-drop, but then wanting to retrieve the "Value" from the item, not the identifier:



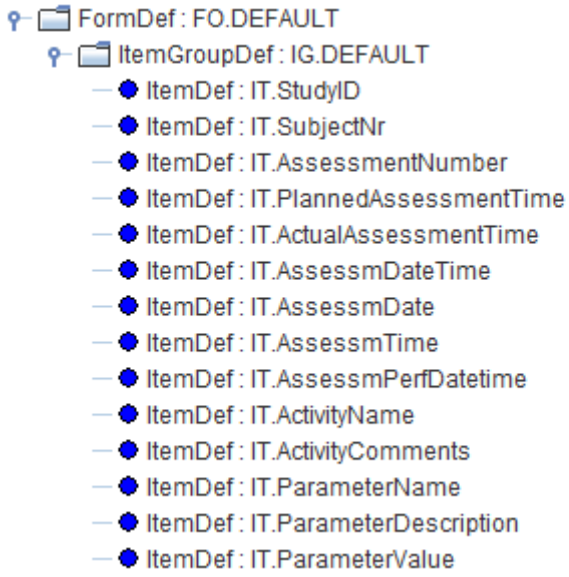
The further treatment of SDTM/SEND generation for "classic" ODM is explained in the many tutorials that can also be found on our website.

Hypervertical ODM

In ODM with an "hypervertical" structure, an Item will no longer represent a single data point any more, but just representing one of the many attributes of the "entity-attribute-value" structure:

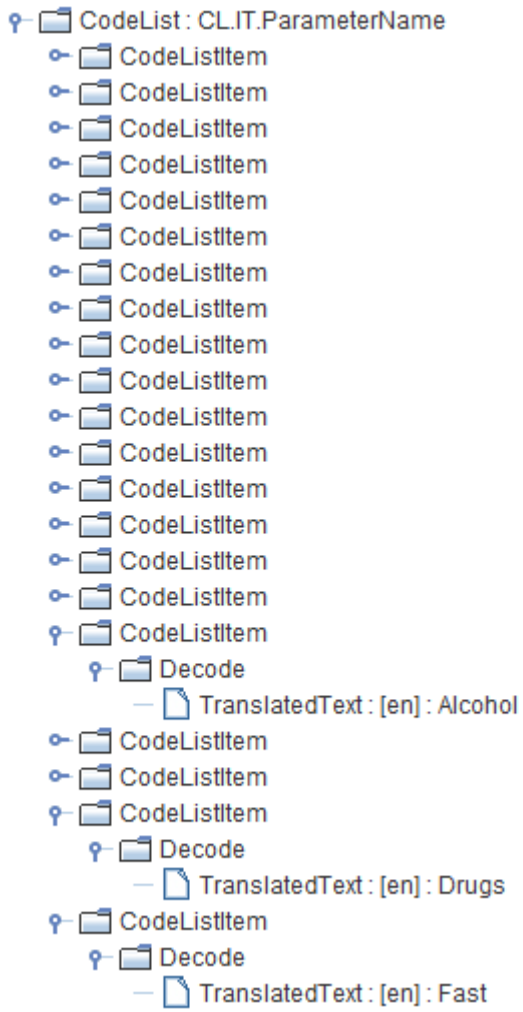


Or when one chooses to display with the OID identifier (menu "View - ODM tree with OIDs"):

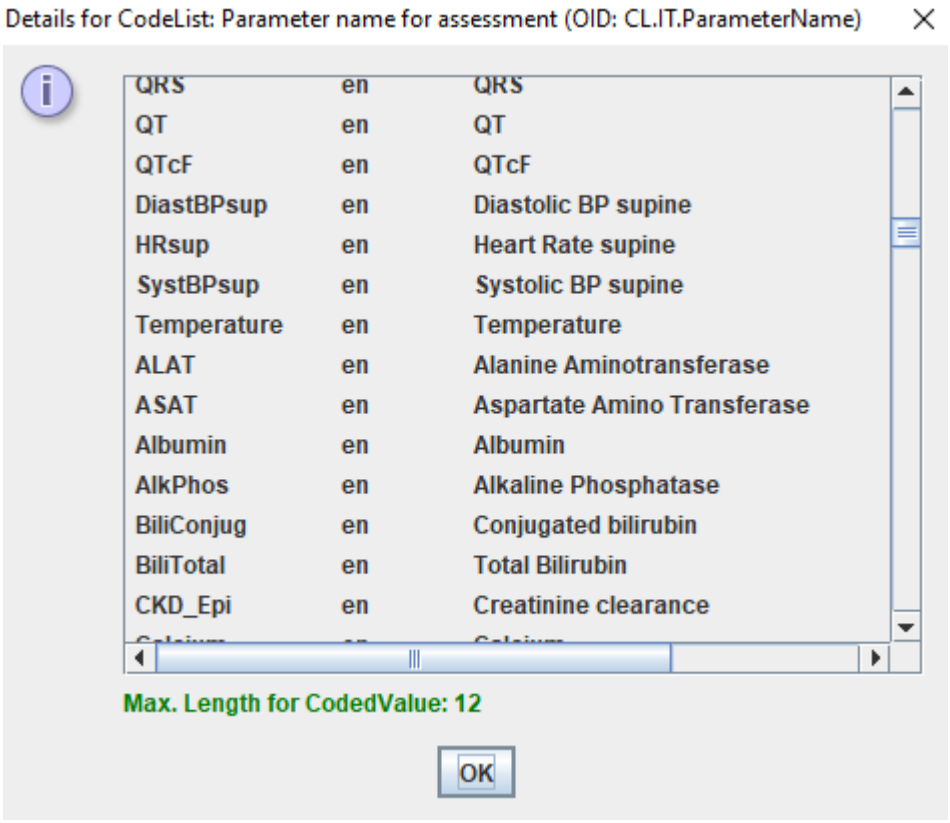


with the **values** of "Parameter Name" (OID "IT.ParameterName") essentially representing the data point definitions, i.e. the "entities".

We can see the list of the entities by navigating to the codelist "CL.IT.ParameterName" in the ODM tree:



or, more easily, select "Parameter Name" (IT.ParameterName) and then use the menu "View - Item CodeList Details", leading to e.g. :



Generating the mappings for LBTESTCD and LBTEST

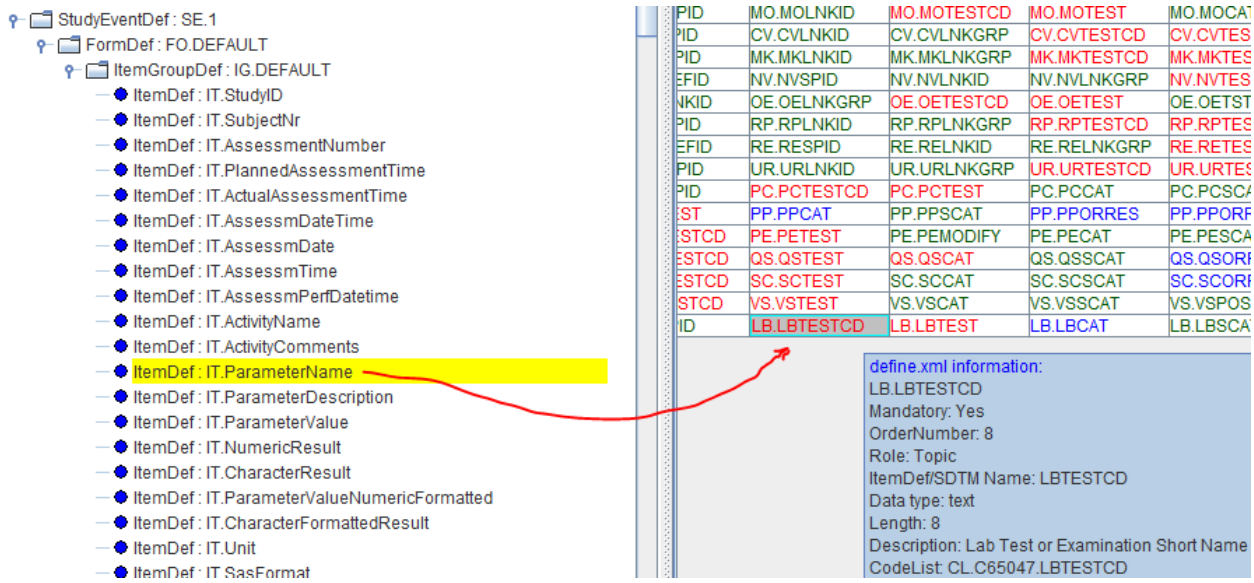
All the following only applies to the case of ODM with an hypervertical structure. The "classic ODM" case is explained in many other of our tutorials.

Now, let's start with trying to generate an SDTM dataset for LB (Laboratory Test Findings).

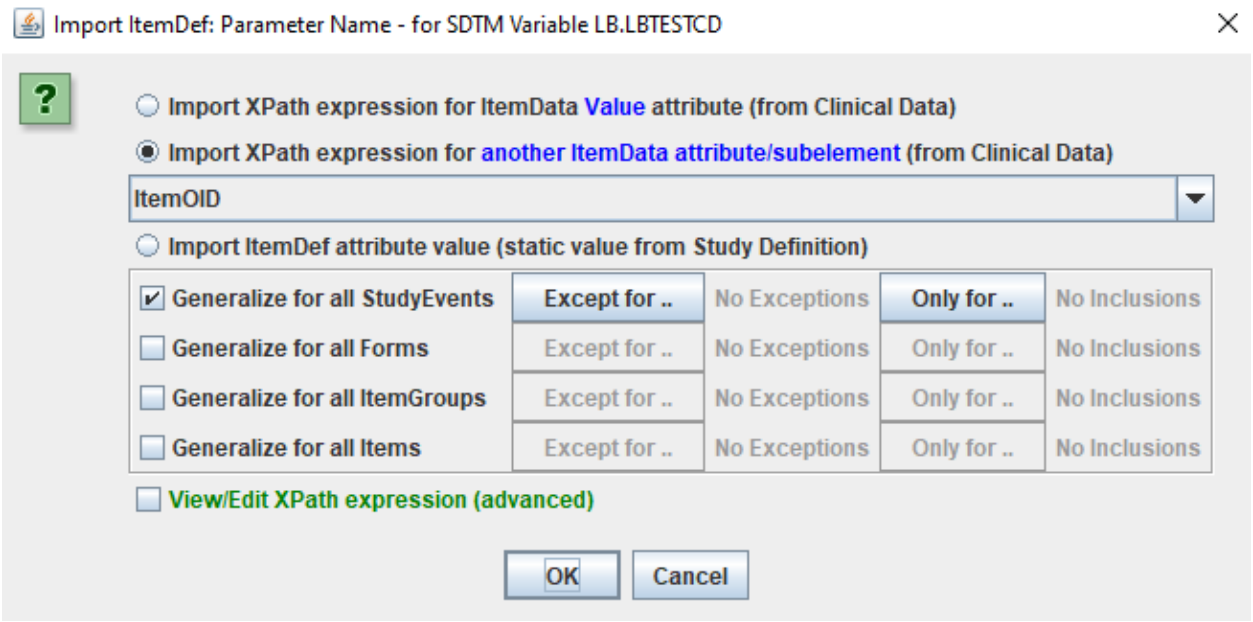
Generating the mapping for LBTESTCD

We first generate the mapping for LBTESTCD, which usually is the "looping variable", i.e. we will iterate over all the values for which there is an entry in the ODM or "one record per lab test result per subject". In the SDTM table, we can see that LBTESTCD is the "looping variable" as it has a light-blue border on the cell:

So, for obtaining a value for LBTESTCD, we must drag-and-drop the "Parameter Name" from the ODM tree to the SDTM "LBTESTCD" cell, after we created a study-specific instance of LB:



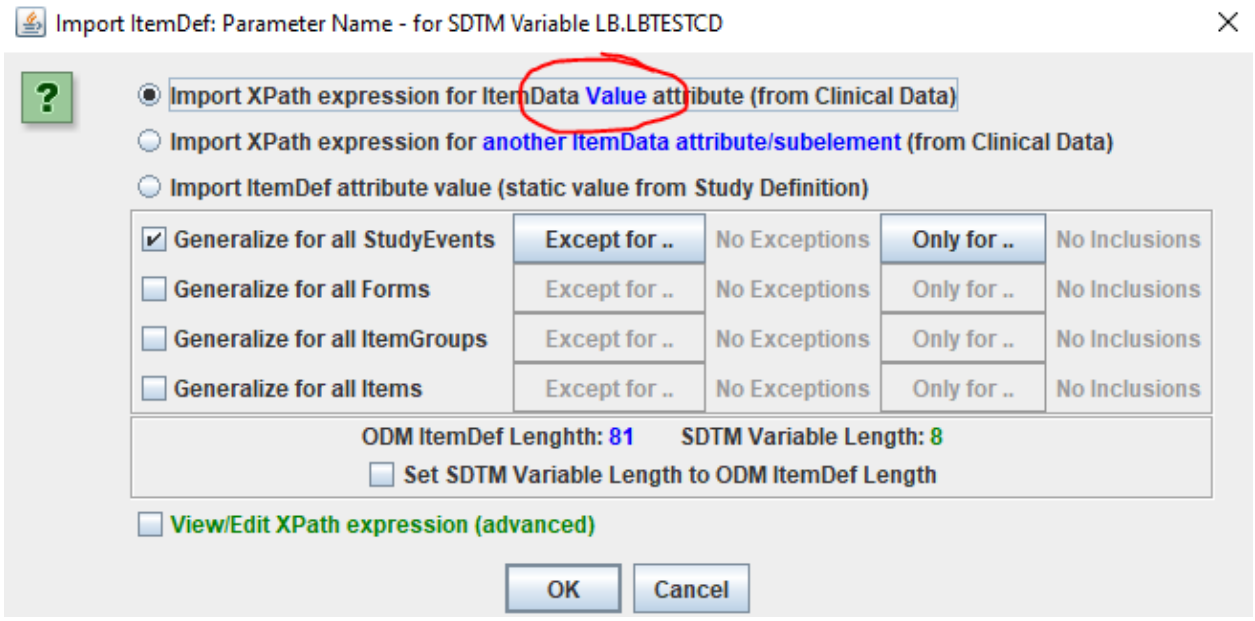
leading to the first wizard:



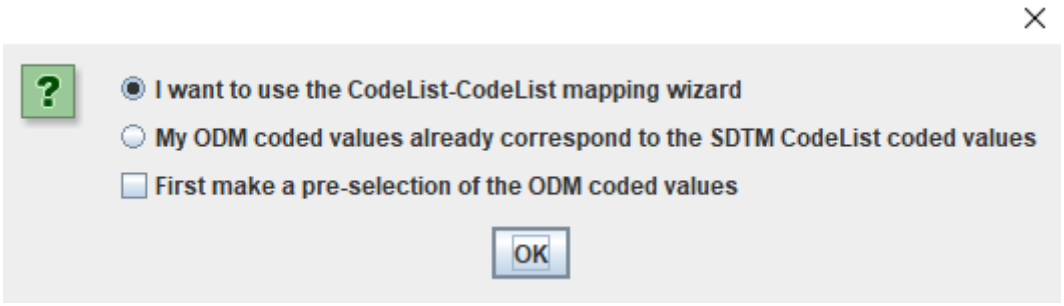
We immediately check the checkbox "Generalize for all StudyEvents", as we want to capture lab data for all the visits, and not for one single visit.

The choice "Import XPath expression for another ..." with choice for "ItemOID" is automatically selected in the case of -TESTCD variables. This is also the most obvious choice in the case of "classic", i.e. non-hypervertical ODM structures, as in that case, the OID ItemOID represents the test itself (e.g. "Albumin", "Bilirubin", ...).
In the case of a "hypervertical" structure, this is however not the case anymore, where the test name essentially is in the "Value" of the ItemData (see section "An ODM example of a hypervertical structure").
So we need to check the radiobutton "Import XPath expression for ItemData *Value* attribute ...":

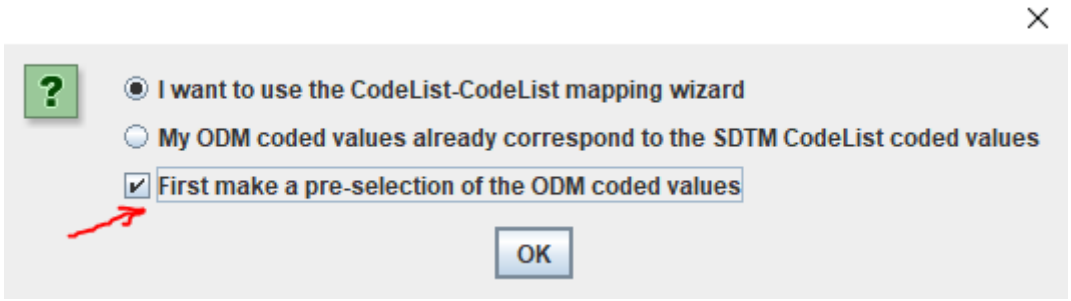
After clicking OK, the second wizard is displayed:



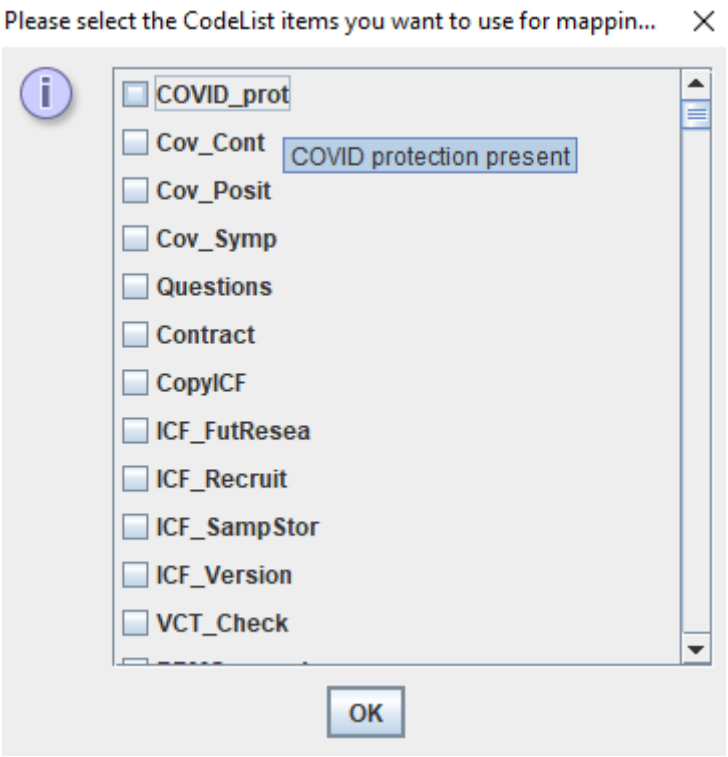
As we want to map each of the "activities" to an SDTM LBTESTCD (which is under CDISC controlled terminology), we select the radiobutton "Use CodeList from the SDTM Variable", and then click "OK". This leads to:



Very often, using the "CodeList-CodeList" mapping wizard is the smartest choice, as it allows some automation (see further on). However, we do not want to map **every** activity name to a value for LBTESTCD, as there can be hundred of such, and only a small part of it may be lab tests.
New in SDTM-ETL 4.2 is therefore the checkbox "First make a pre-selection of the ODM coded values", which we must check in order to be able to make such a pre-selection:



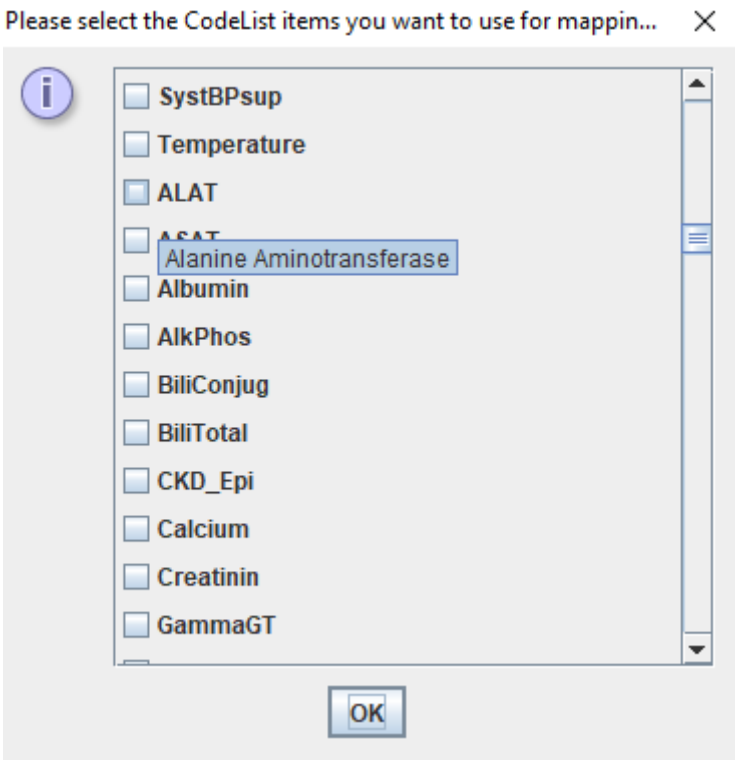
Clicking OK then leads to:



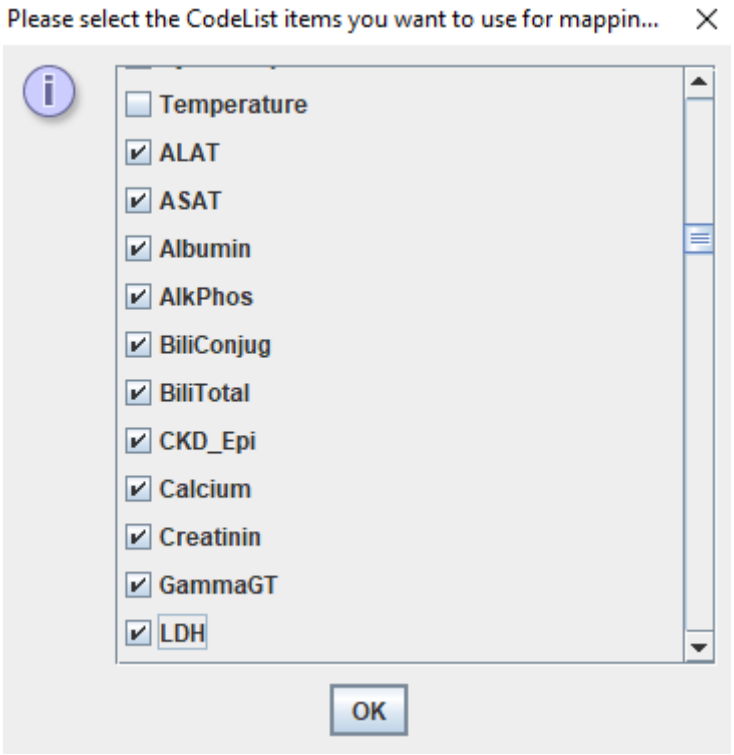
Now it is clear that the parameter "COVID protection" is not about a lab values, so we do not want to map it to LBTESTCD.

At this point, it is often clever to **not** to try to map each lab test to LBTESTCD, but to limit the selection to a group of lab tests that belong together, e.g. serum chemistry tests. A look into the annotated CRF can often help in making the right selection.
One can then generate another instance of the LB domain e.g. for "Urinalysis", "Hematology", "Coagulation", etc.. This mechanism is often called "split data sets", which essentially is a false name, as we do not split data sets, but generate different ones right from the start.
Having different data sets for different categories for lab tests is also advantageous for the (regulatory) reviewer, as LB datasets can easily grow to millions of records, making it difficult to review them.

In our case, we only want to include serum chemistry tests, and thus start selecting:

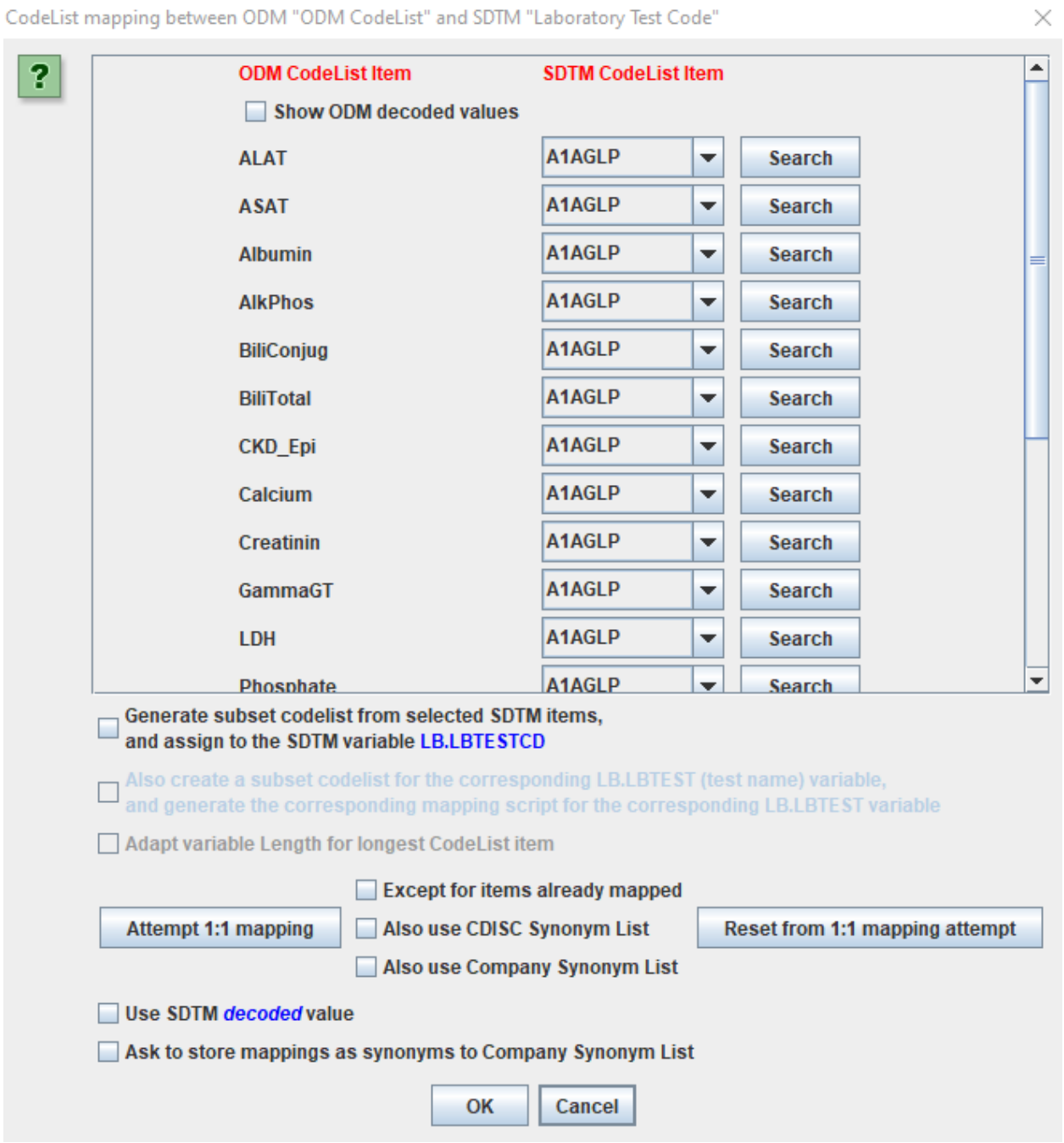


The tooltips, representing the "decoded" values often help making the right selection, as well as a look in the annotated CRF on the "Blood Serum Chemistry" page (activity with ID BSChem_A and BSChem_Gluc):



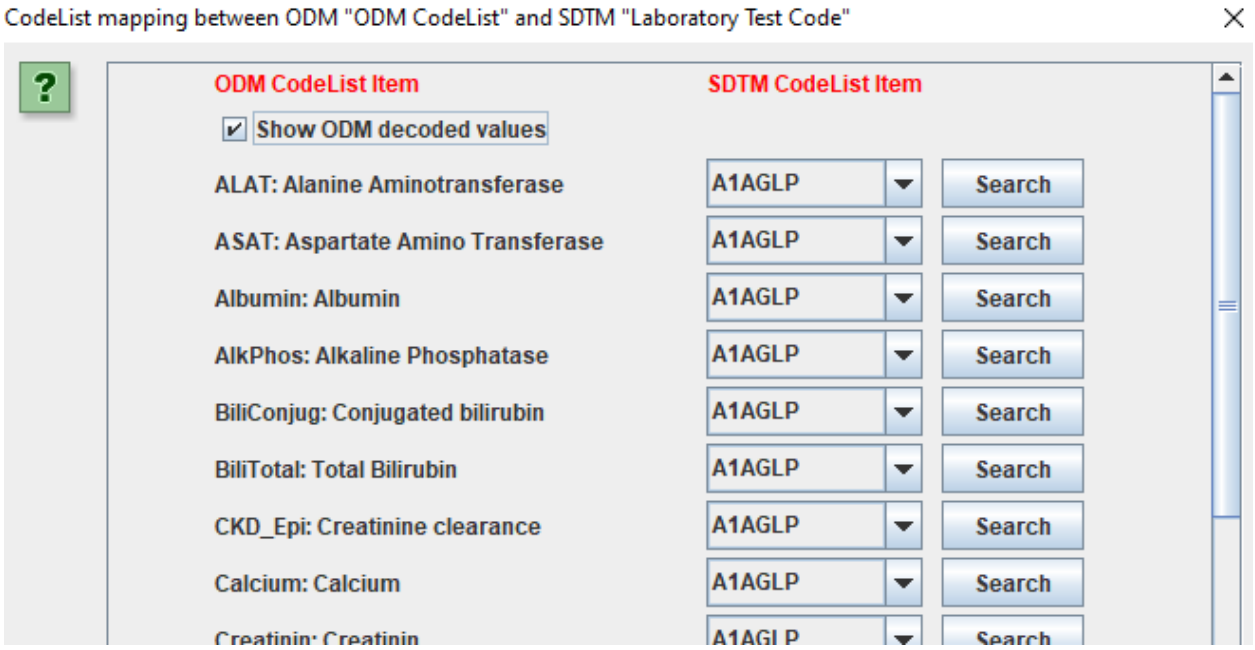
Remark that when one misses one, or has one too much, one can later still correct in the mapping script, but this requires a little bit of understanding of XPath expresssions (see later). Of course, one can also always do the drag-and-drop over again, and overwrite the prior mapping.

Clicking "OK" then leads to:



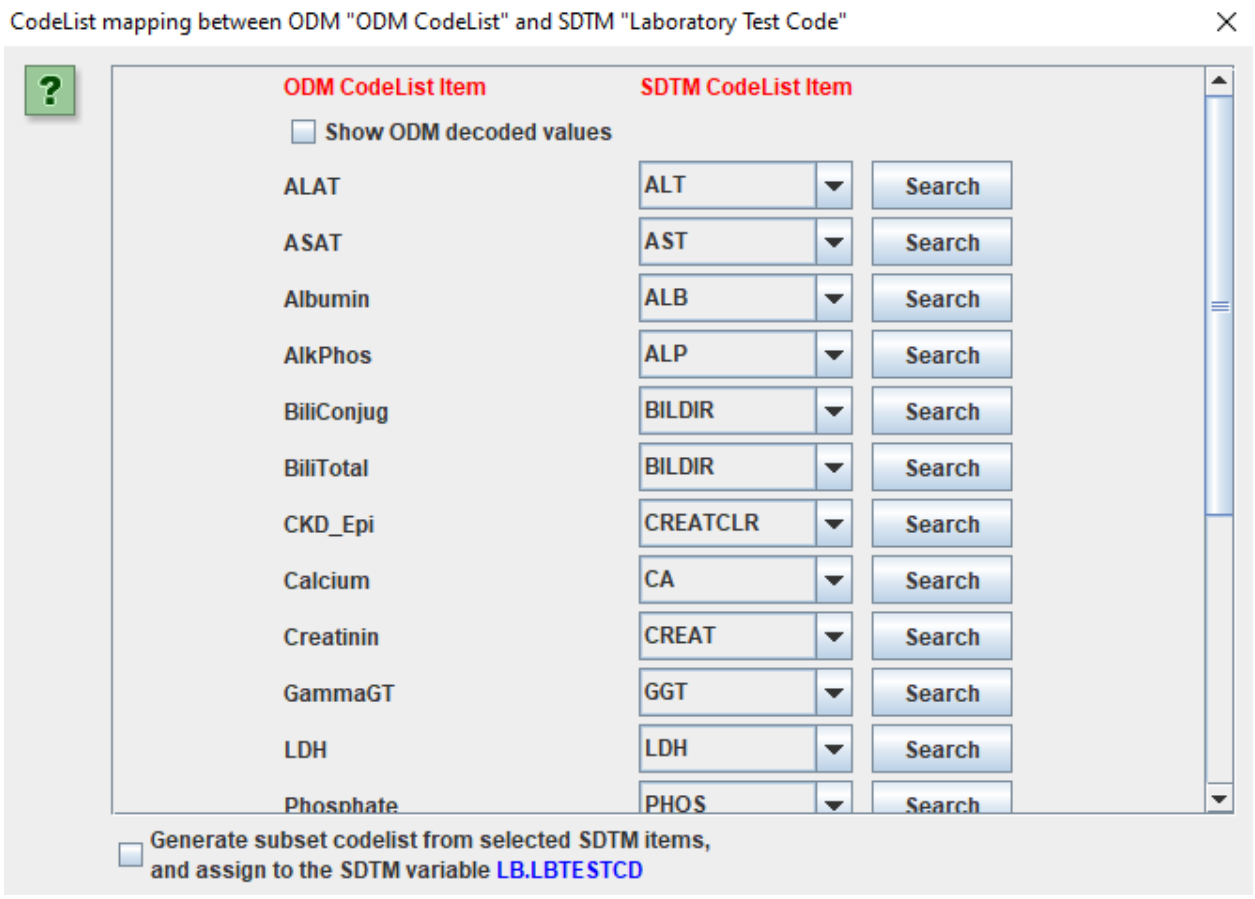
showing the "CodeList-CodeList Mapping Wizard". There here are over 20 codes from the ODM side to match.

Depending on how the ODM codelist was constructed, one can also get more details by checking the checkbox "Show ODM decoded values":

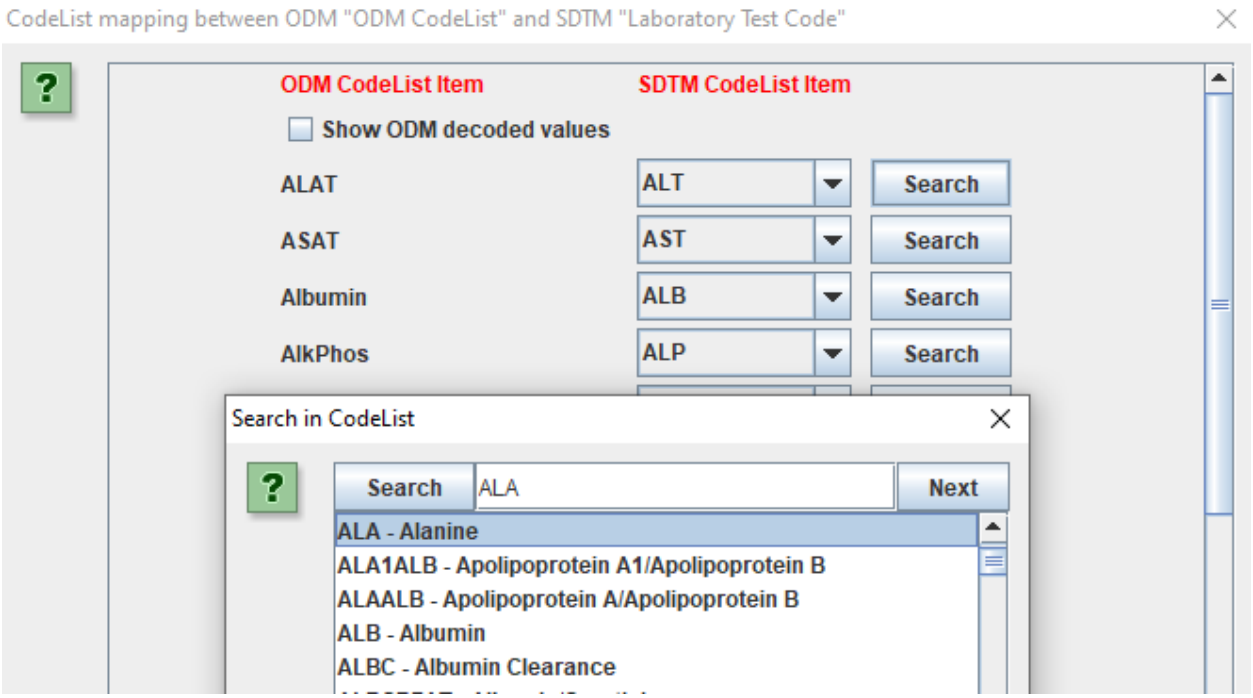


As there are over 2,000 lab test codes for LBTESTCD, selecting the right one can be a bit difficult, as it requires a good understanding about CDISC controlled terminology and how it works.
One can however always try the "Attempt 1:1 mapping" button, which allows to partially automate the process, i.e. the wizard will then propose mappings based on word similarity (for coded and decoded value). So, we will give it a try. Click the "Attempt 1:1 mapping" button, and see what is happening. It will take some time, so maybe this is a good time to go for a cup of tea or coffee ...
One can also check the "Also use CDISC Synonym List" to increase the probability of a good match, but at the cost of searching time.

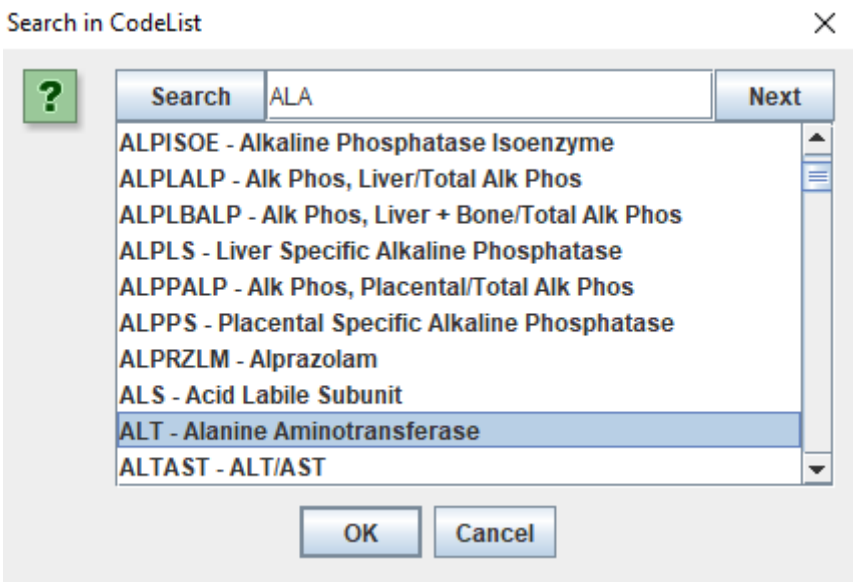
The result is:



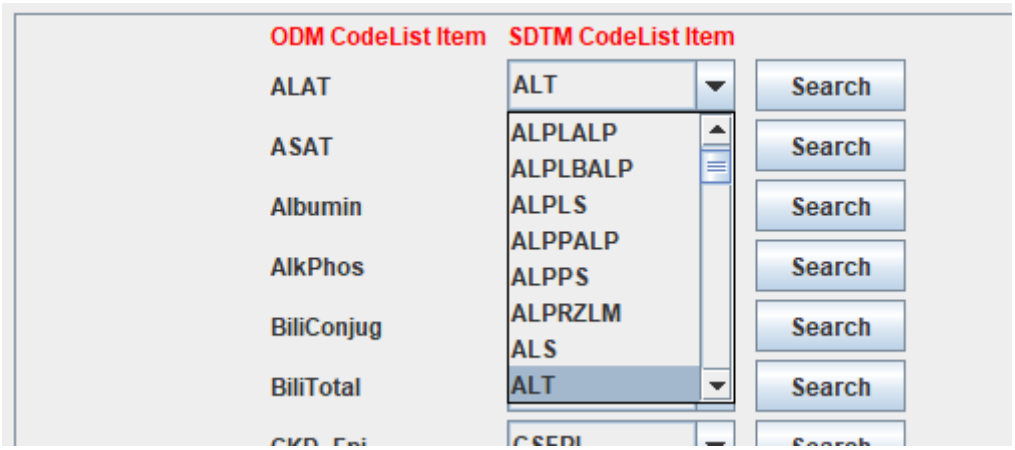
One should now carefully check each of the proposed mappings. For example, for "ALAT", the SDTM code "ALA" is proposed, for which one may have some doubts. Using the "Search" button on it, one can then look for what "ALA" means and finds:



which is clearly is not what one wants. Further searching leads to:

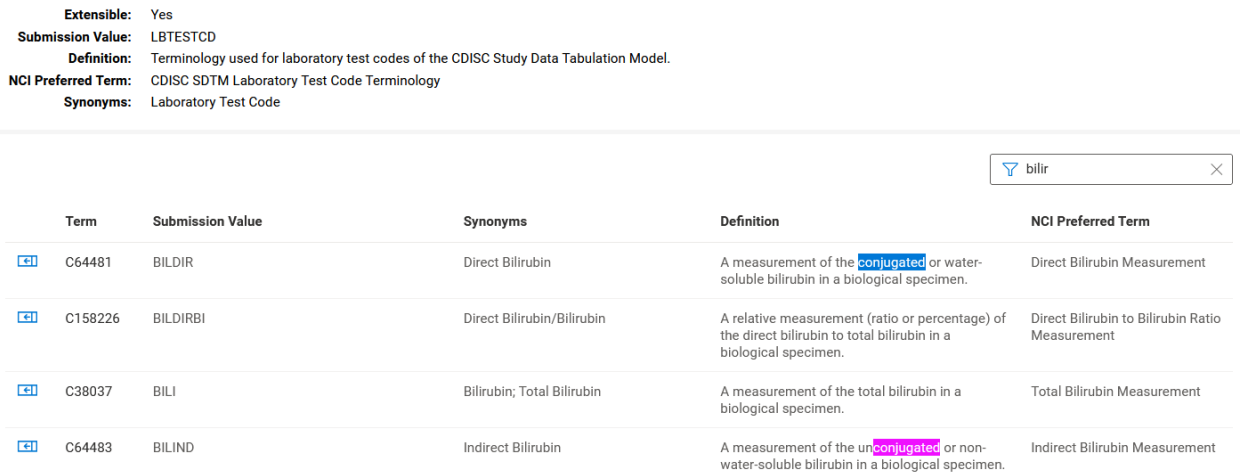


which looks to be the better choice. Clicking "OK", and then wait 1-2 seconds, leads to:



now selecting the correct choice and mapping.

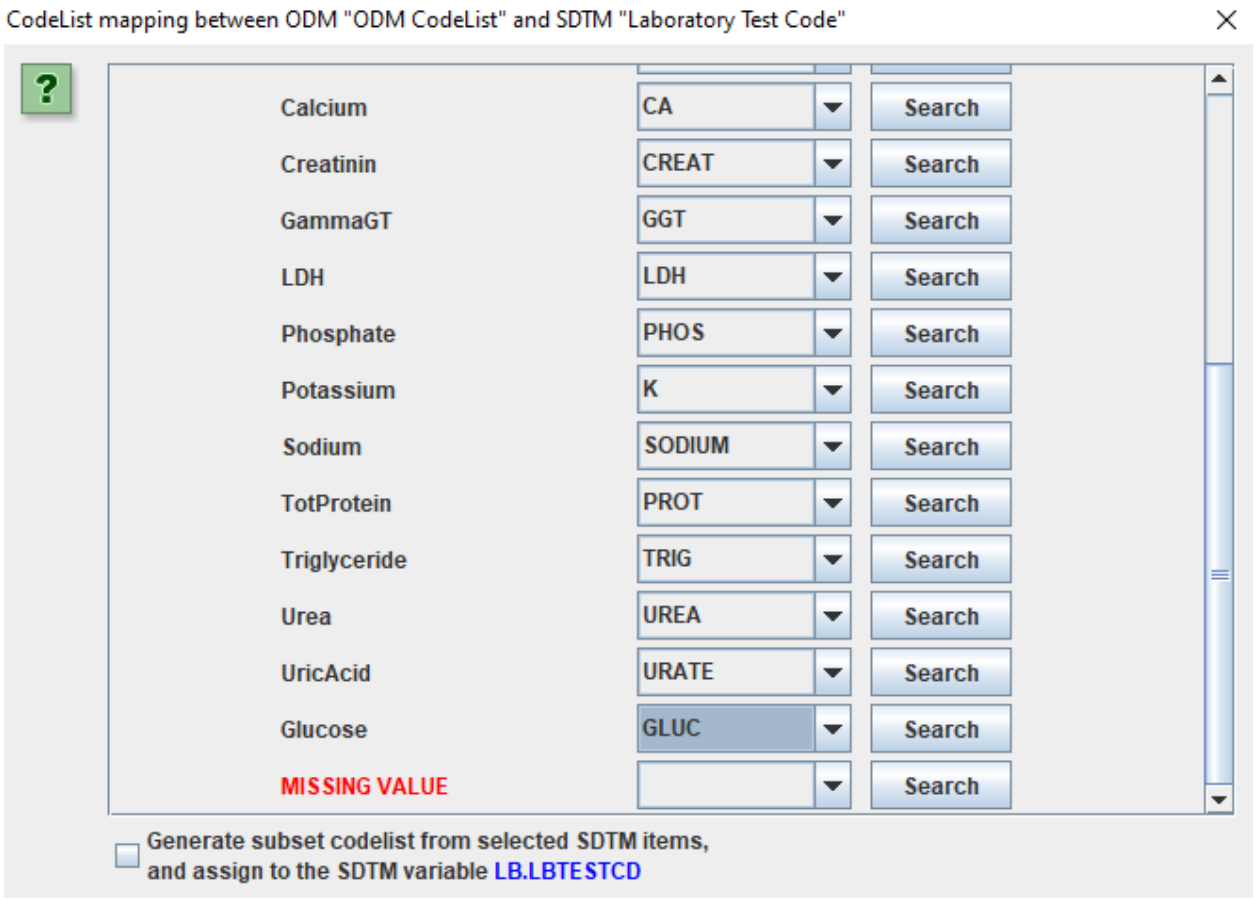
Also using the CDISC-Library browser (https://library.cdisc.org/browser/) can help a lot in making the right mapping decisions, e.g.:



Generating all these mappings between "local" lab codes and CDISC-CT can be tedious. Don't

blame us - blame CDISC[1]!

At the end, we have the following mappings:

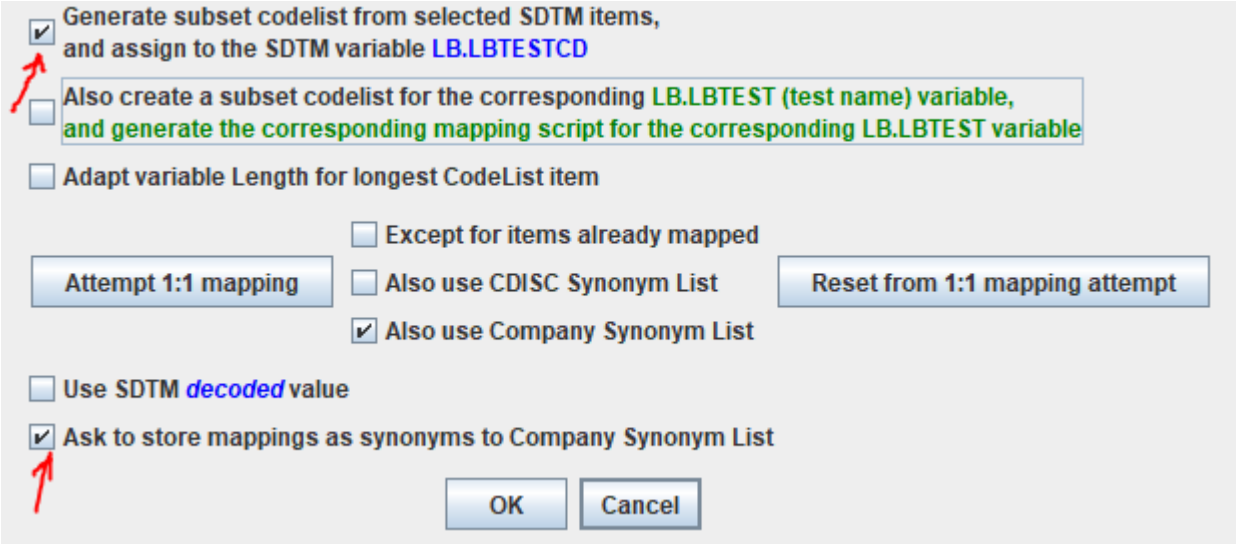


It might be that one cannot find a mapping for a certain test, as there is no CDISC controlled terminology (yet). In that case, just select "blank" (empty), and assign an own invented code (8 characters maximum) later in the mapping script itself˙.

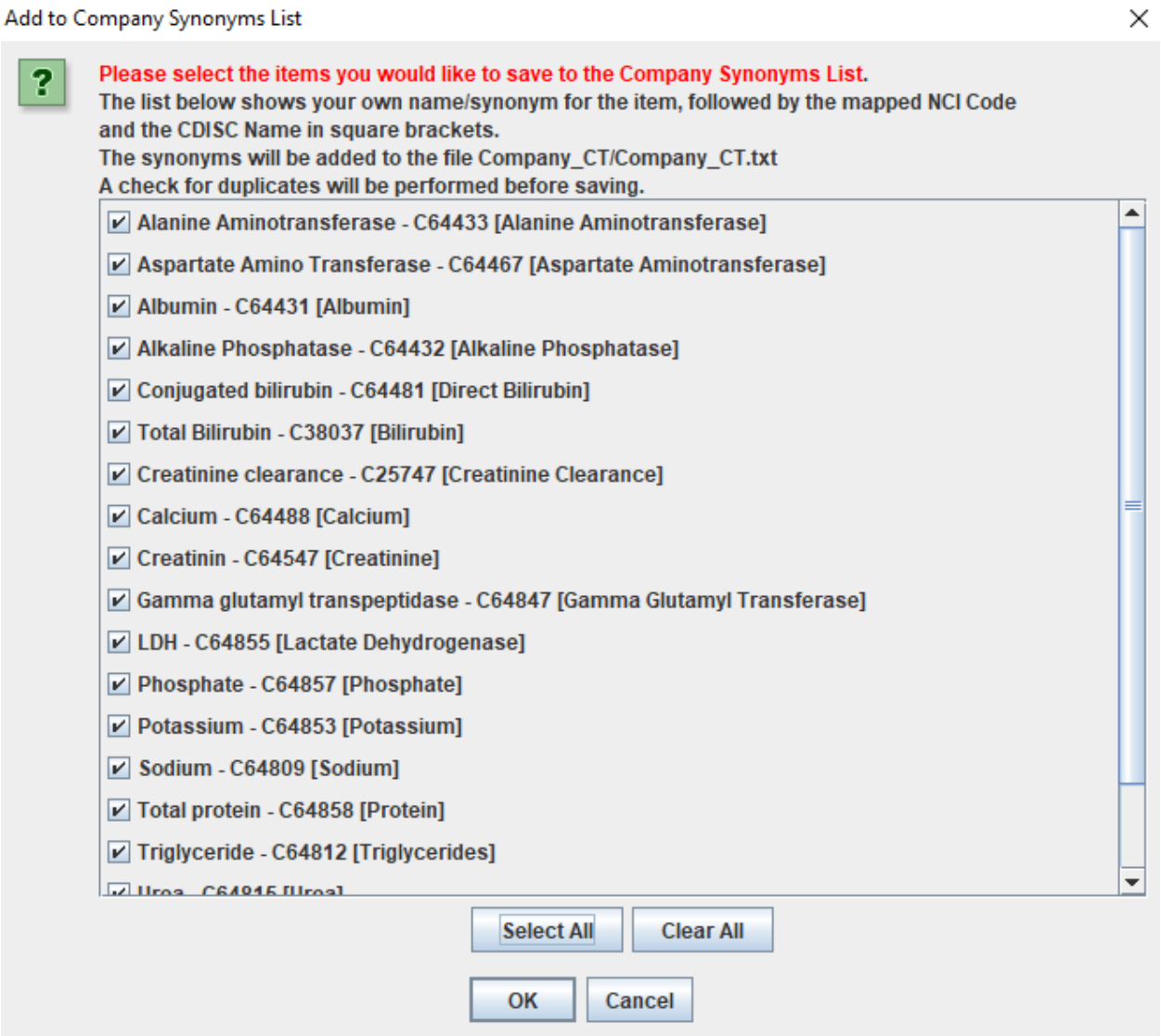
At this point, it is very interesting to also check the checkbox "Generate subset codeList ...", which will generate a LBTESTCD codelist with only, the LBTESTCD codes that are used in this mapping, and that is stored in the underlying define.xml. This will clearly show the reviewer which of the SDTM codes were used for this variable.
In the case of different datasets for different types of lab tests (chemistry, hematology, urinalysis, ...) one will later than assign that codelist to the "value list" with e.g. a "where clause" with "WHERE LBCAT=CHEMISTRY".

Also, if one often has these local lab test codes, one can store the mappings to a "Company synonyms list", so that one can reuse these mappings later studies:

[1] CDISC still refusing to use the LOINC code as the real identifier for lab, microbiology and vital signs tests.



Now clicking "OK" leads to:



This list is then added to and stores in the folder and file "Company_CT/Company_CT.txt, which then looks like:

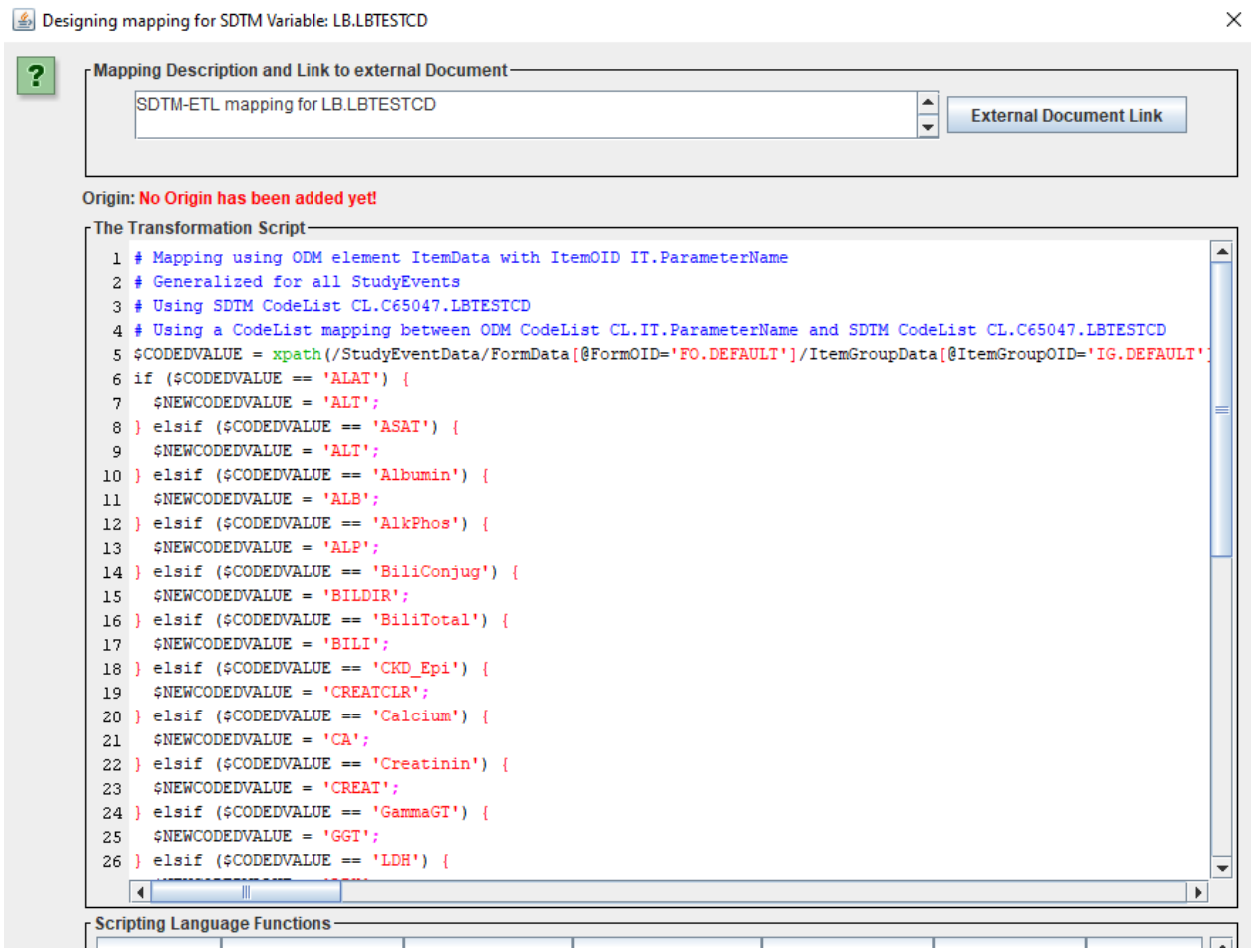
```
C38295 PR
C38300 Sublingual
C38216 Inhaled
C38304 Topical

C64433 Alanine Aminotransferase
C64467 Aspartate Amino Transferase
C64431 Albumin
C64432 Alkaline Phosphatase
C64481 Conjugated bilirubin
C38037 Total Bilirubin
C25747 Creatinine clearance
C64488 Calcium
C64547 Creatinin
C64847 Gamma glutamyl transpeptidase
C64855 LDH
C64857 Phosphate
C64853 Potassium
C64809 Sodium
C64858 Total protein
C64812 Triglyceride
C64815 Urea
C64814 Uric Acid
C105585 Glucose
```

One sees that it already contains some older entries, and that the new have been added.
These entries have the CDISC-NCI code for the SDTM controlled terminology, followed by the company synonym, as present in the ODM codelist (decoded values).
We use the CDISC-NCI code, as in some cases, the "submission term" changes between codelist versions, and the CDISC-NCI code is the real identifier.

This list can then later be used (e.g. in other studies) for searching a suitable codelist mapping by checking the checkbox "Also use Company Synonym List". This will then not only make the search much faster, but also to better mapping results.

The mapping script then generated is:



One can then still edit this mapping script, e.g. when one did not find a CDISC code for a test, and want to assign ones own one.

Now have a look at the first non-comment line in the script (one can use the button "Full-screen Transformation Script Panel" for a better experience:

```
# Mapping using ODM element ItemData with ItemOID IT.ParameterName
# Generalized for all StudyEvents
# Using SDTM CodeList CL.C65047.LBTESTCD
# Using a CodeList mapping between ODM CodeList CL.IT.ParameterName and SDTM CodeList CL.C65047.LBTESTCD
$CODEDVALUE = xpath(/StudyEventData/FormData[@FormOID='FO.DEFAULT']/ItemGroupData[@ItemGroupOID='IG.DEFAULT']/ItemData[@ItemOID='IT.ParameterName'][@Value='ALAT' or @Value='ASAT' or @Value='Albumin'
if ($CODEDVALUE == 'ALAT') {
  $NEWCODEDVALUE = 'ALT';
} elsif ($CODEDVALUE == 'ASAT') {
```

The "xpath()" function selects one or more paths to the ODM structure.
Important here is to understand that the square brackets [.....] mean a selection (or "where" statement), in XPath language called a "predicate".
So, for "StudyEventData" we see no square brackets, meaning that we do not select any visit or set of visits, i.e. we take all visits.
For "FormData", we select the form with ID "FO.DEFAULT" - due to the hypervertical structure, there is only one form, and for ItemGroupData we select the one with ID "IG.DEFAULT", as also here due to the hypervertical structure, there is only one ItemGroup (i.e. one section in the form).

For selecting which items are taken into account, we find:

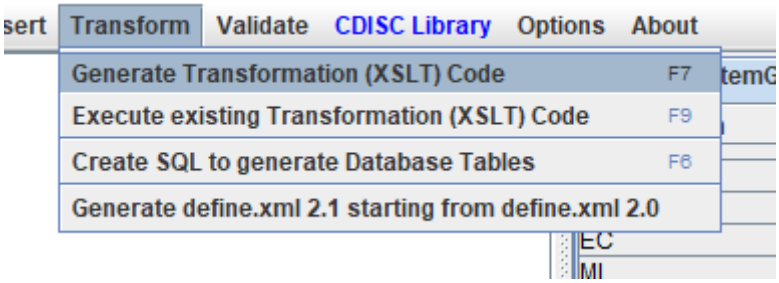
```
Name and SDTM CodeList CL.C65047.LBTESTCD
LT']/ItemGroupData[@ItemGroupOID='IG.DEFAULT']/ItemData[@ItemOID='IT.ParameterName'][@Value='ALAT' or @Value='ASAT' or @Value='Albumin' or @Valu
```

followed by, on the right:

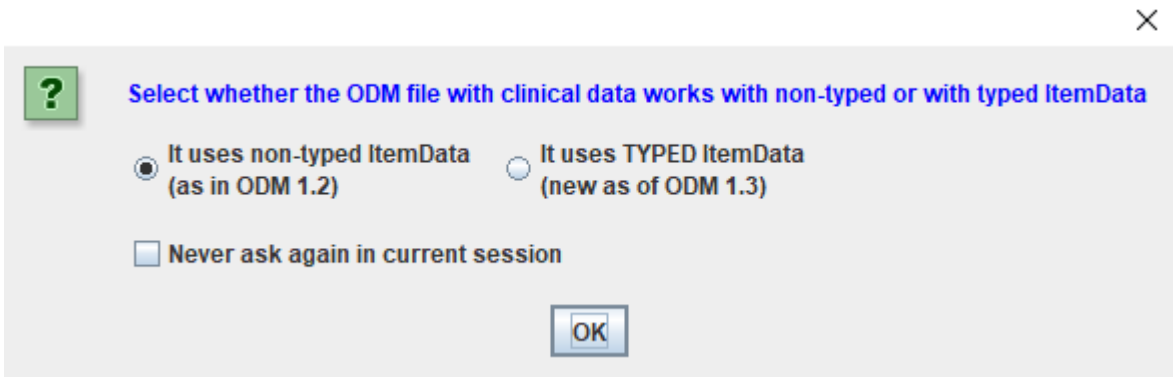
```
@Value='Potassium' or @Value='Sodium' or @Value='TotProtein' or @Value='Triglyceride' or @Value='Urea' or @Value='UricAcid' or @Value='eGFRcheck_MD' or @Value='Glucose' or @Value='LabGluc']/@Value);
```

i.e. we only take the items for which the parameter name ID is "ALAT", "ASAT", ... down to "LabGluc".
If we now still want to add or remove an item to the selection, we can simply edit the XPath selection expression by adding or removing "or @Value="..." parts.

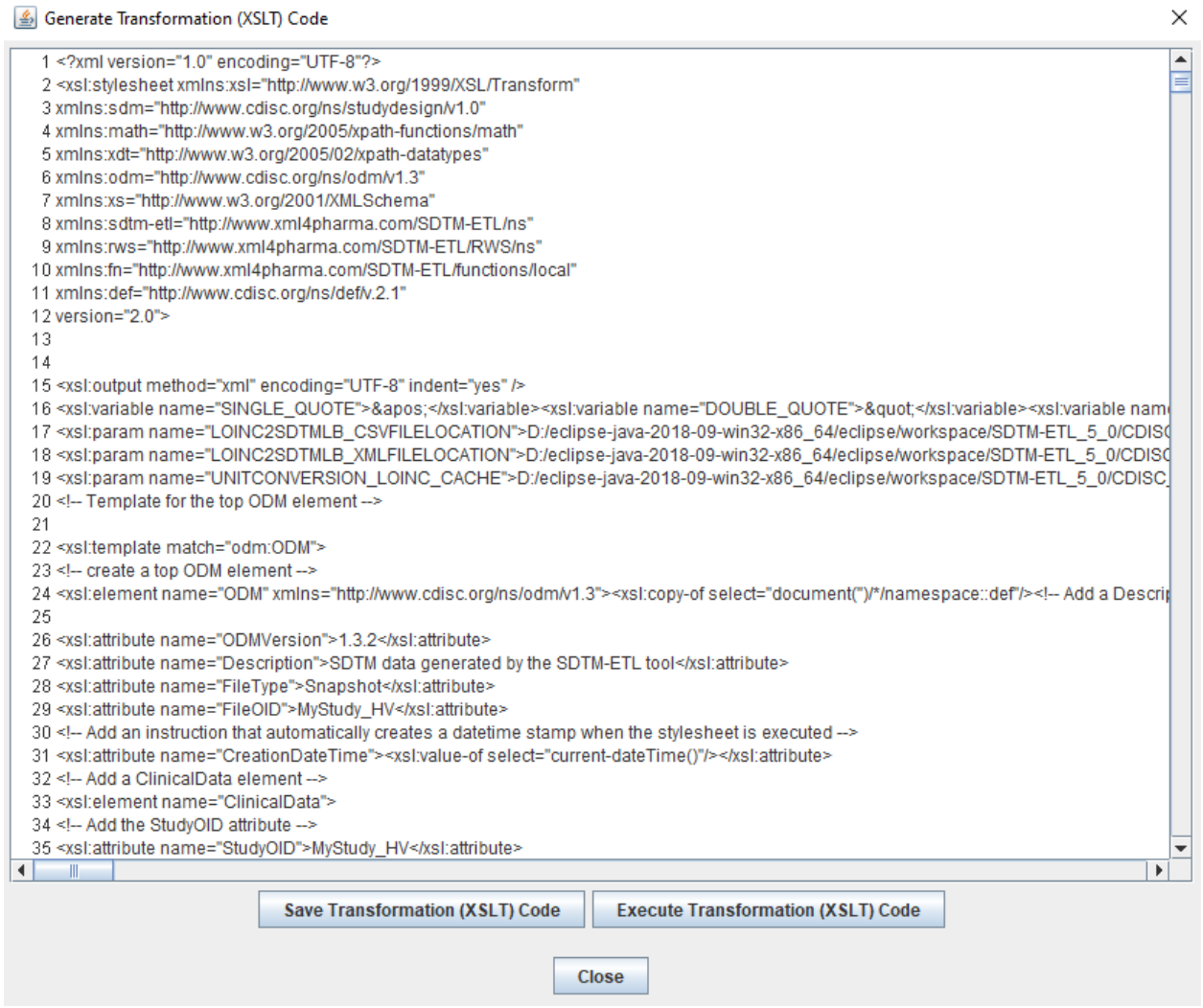
After having developed our mapping, we of course want to test it on some real data (this can also be mock data when one starts using SDTM-ETL even before the real start of the study). To do so, use the menu "Transform - Generate Transformation (XSLT) Code for SAS-XPT":



which then results in:



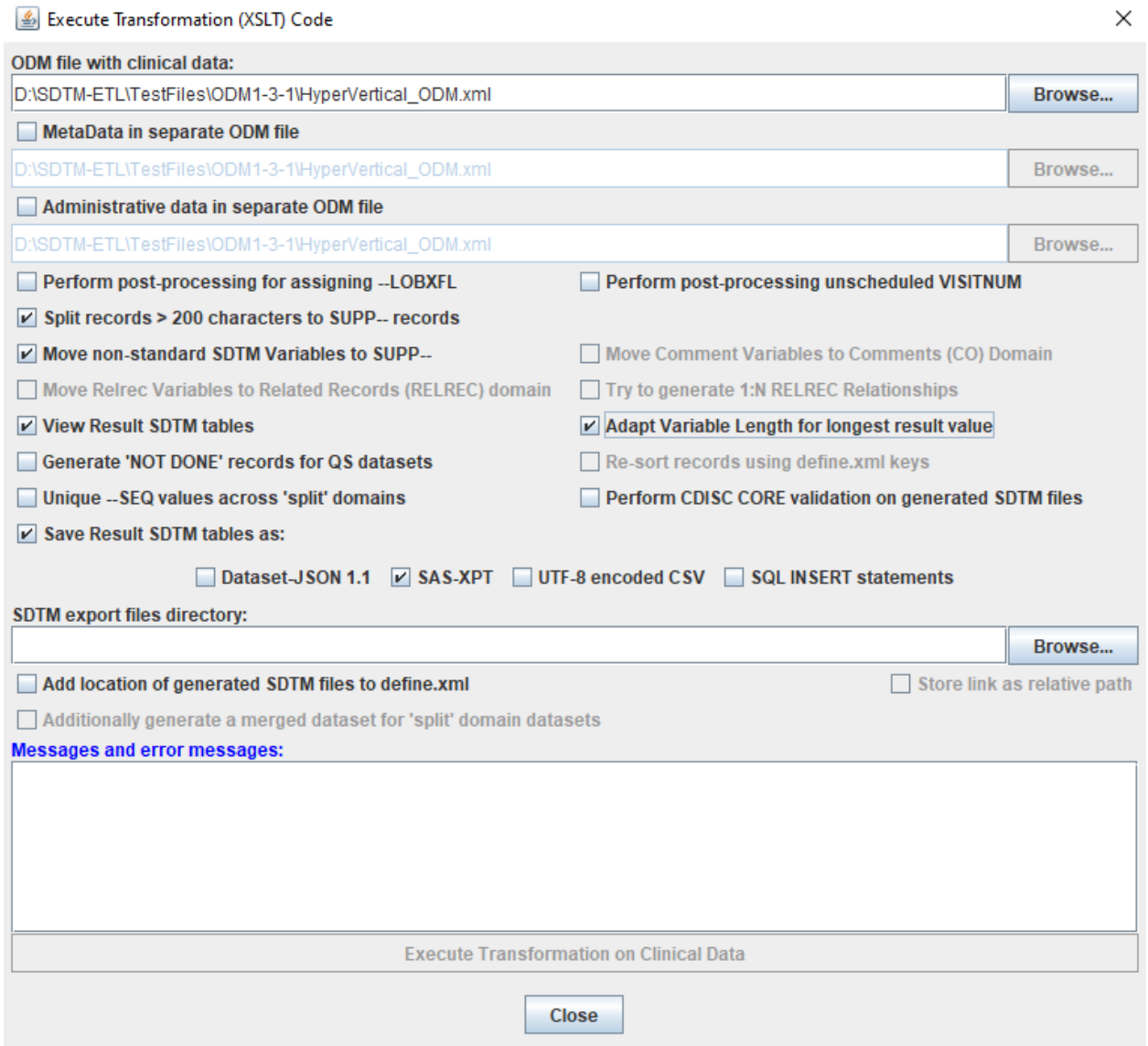
as there are 2 "flavors" of ODM files with clinical data. 90% of the users of ODM use the "non-typed ItemData". When the ODM is generated from CSV or SAS files using the "ODMGenerator", it also uses "untyped ItemData". After clicking OK, we get:



The reason that this (intermediate) XSLT script is shown, is that some users want to save it to file, to later use it in "batch" generation of SDTM datasets, sometimes even in an automated process, e.g. when new data come in every night, or once a week. There is however also the possibility to run the native scripts (as stored in the define.xml) in "batch" mode. See the tutorial "SDTM-ETL Light' and running in batch execution mode"

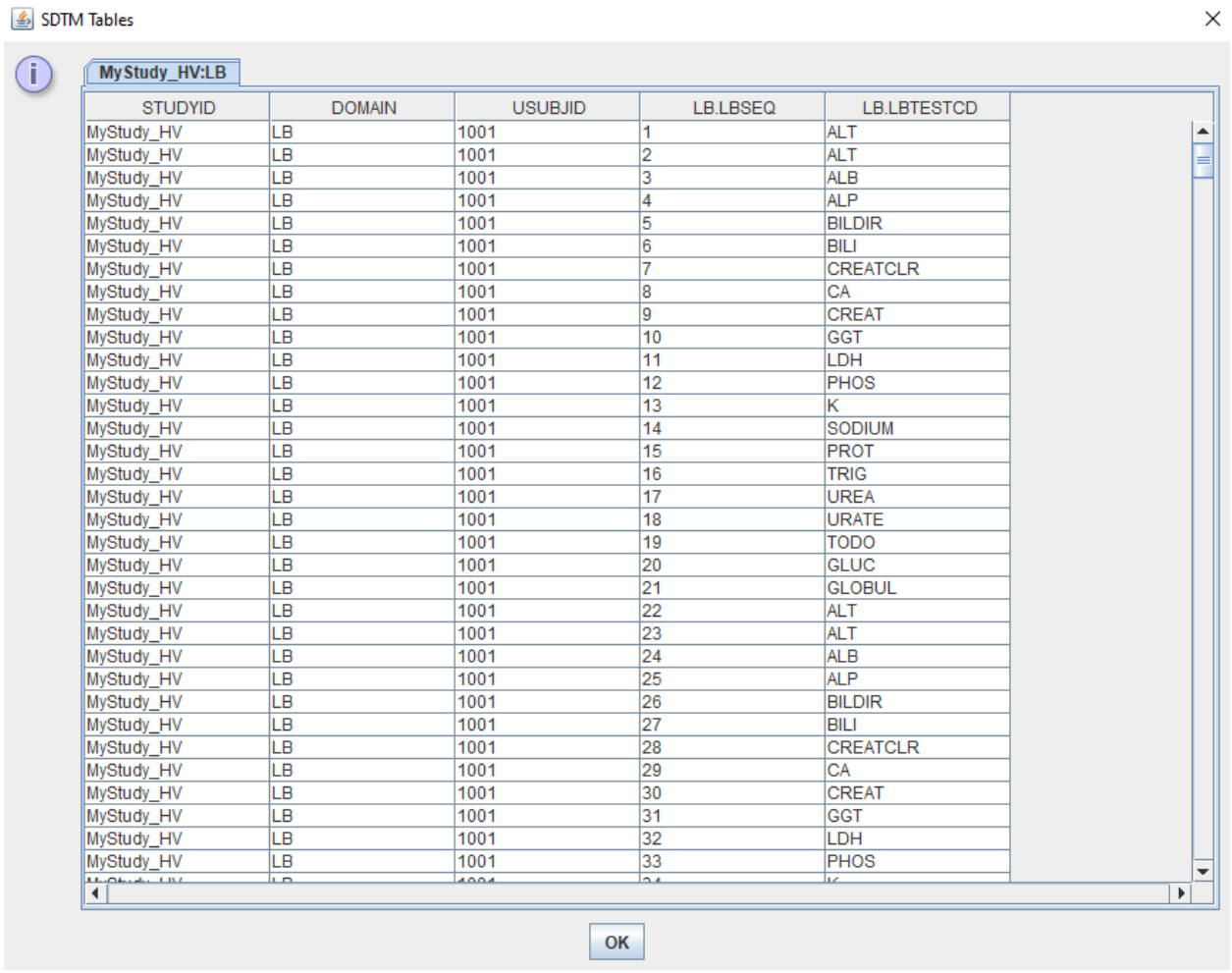
One can have the software skipped this step by using the "Options" menu, and check the box "Skip display of generated XSLT". Some users however also use it for debugging when some mappings do not deliver what they expect.

Then click "Execute Transformation (XSLT) Code", and provide an ODM file with the clinical data:



During development, one will usually not want to actually generate SAS-XPT files, so one can leave the checkbox "Save Result SDTM tables as" and "SAS-XPT" unchecked. If one wants to have the XPT files (e.g. for discussing the mapping results with colleagues"), one can of course check the checkbox and provide a directory to which the XPT files need to be written to. As of v.5.0, it is also possible to generate the datasets in the modern CDISC Dataset-JSON-1.1 format, which will probably soon become the successor of XPT at the regulatory agencies.

In our case, clicking "Execute Transformation on Clinical Data" leads to:



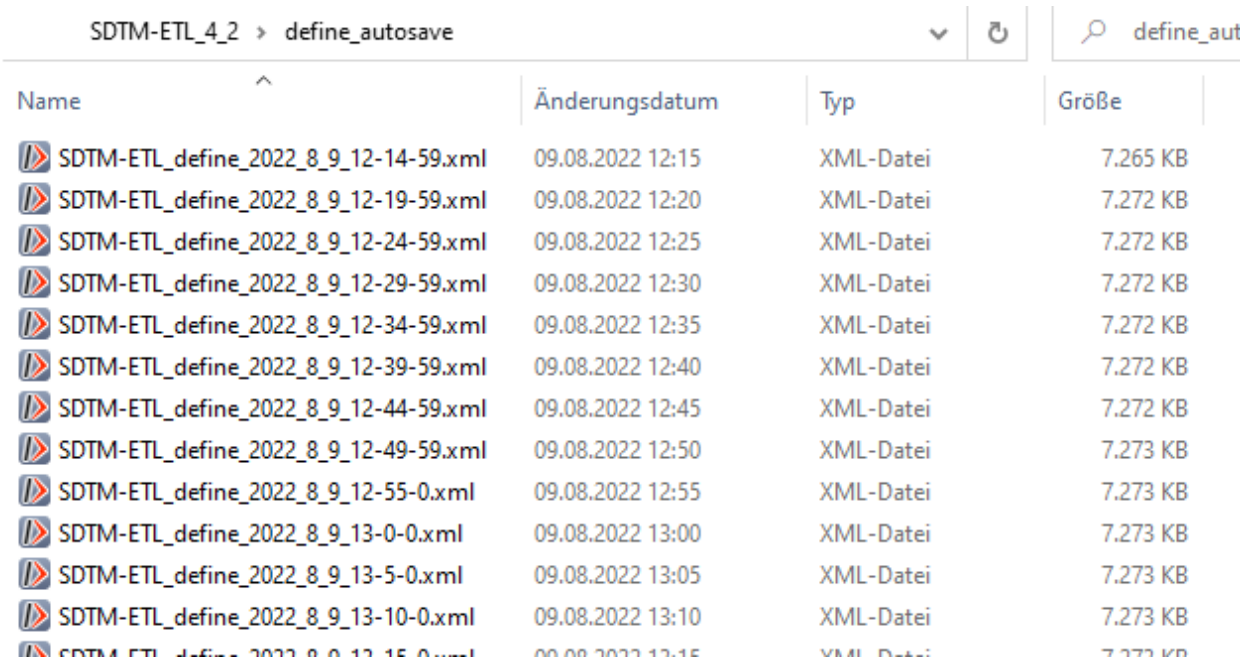
STUDYID	DOMAIN	USUBJID	LB.LBSEQ	LB.LBTESTCD
MyStudy_HV	LB	1001	1	ALT
MyStudy_HV	LB	1001	2	ALT
MyStudy_HV	LB	1001	3	ALB
MyStudy_HV	LB	1001	4	ALP
MyStudy_HV	LB	1001	5	BILDIR
MyStudy_HV	LB	1001	6	BILI
MyStudy_HV	LB	1001	7	CREATCLR
MyStudy_HV	LB	1001	8	CA
MyStudy_HV	LB	1001	9	CREAT
MyStudy_HV	LB	1001	10	GGT
MyStudy_HV	LB	1001	11	LDH
MyStudy_HV	LB	1001	12	PHOS
MyStudy_HV	LB	1001	13	K
MyStudy_HV	LB	1001	14	SODIUM
MyStudy_HV	LB	1001	15	PROT
MyStudy_HV	LB	1001	16	TRIG
MyStudy_HV	LB	1001	17	UREA
MyStudy_HV	LB	1001	18	URATE
MyStudy_HV	LB	1001	19	TODO
MyStudy_HV	LB	1001	20	GLUC
MyStudy_HV	LB	1001	21	GLOBUL
MyStudy_HV	LB	1001	22	ALT
MyStudy_HV	LB	1001	23	ALT
MyStudy_HV	LB	1001	24	ALB
MyStudy_HV	LB	1001	25	ALP
MyStudy_HV	LB	1001	26	BILDIR
MyStudy_HV	LB	1001	27	BILI
MyStudy_HV	LB	1001	28	CREATCLR
MyStudy_HV	LB	1001	29	CA
MyStudy_HV	LB	1001	30	CREAT
MyStudy_HV	LB	1001	31	GGT
MyStudy_HV	LB	1001	32	LDH
MyStudy_HV	LB	1001	33	PHOS

One sees that the "sequence numbers" LBSEQ are automatically created and that they restart at "1" for each new subject, as required by the SDTM Implementation Guide:

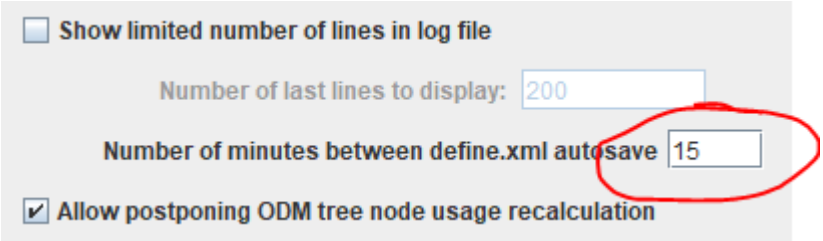
MyStudy_HV	LB	1001	77	URATE
MyStudy_HV	LB	1001	78	TODO
MyStudy_HV	LB	1002	1	ALT
MyStudy_HV	LB	1002	2	ALT
MyStudy_HV	LB	1002	3	ALB

If you haven't done before yet, it is now a good idea to save your work using the menu "File - Save define.xml".

Remark: every N minutes (default: 15 minutes) your work is automatically saved as a define.xml in the directory "define_autosave". For example:



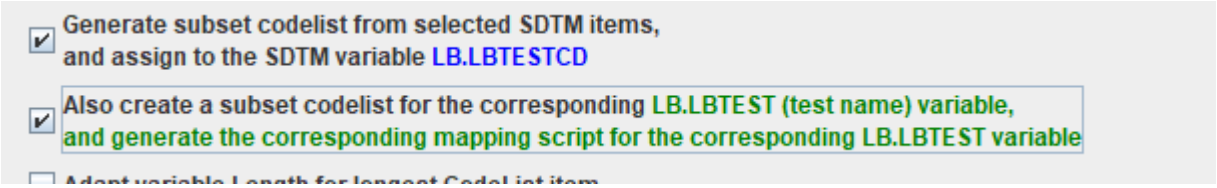
You can change the interval time for "autosaving" using the menu "Options - Properties", and then changing the value in the field "":



or set them in the file "properties.dat" which is always read first when starting the software.

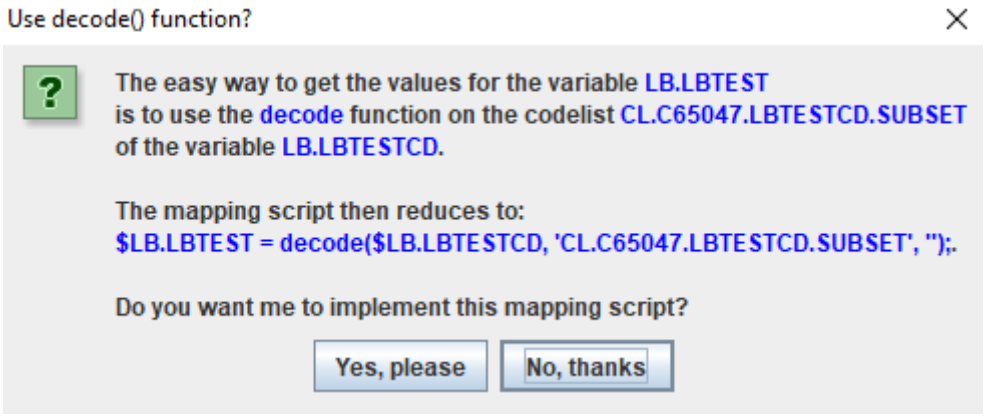
Generating the mapping for LBTEST

If, during the CodeList-CodeList mapping one has also checked the checkbox "Also create a subset codelist for the corresponding LBTEST ...":

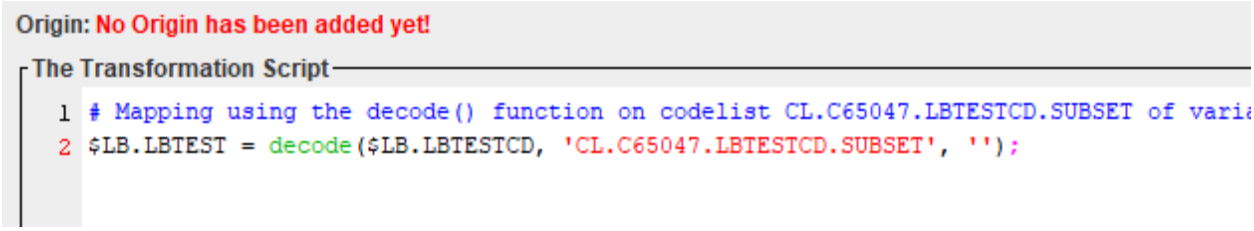


then a mapping script will also be generated for LBTEST, which can then be used and even edited when necessary. If one could map all ODM codelist terms to SDTM codelist terms, there is however an easier way to generate the LBTEST values that correspond to the LBTESTCD values (as there is a 1:1 relationship).

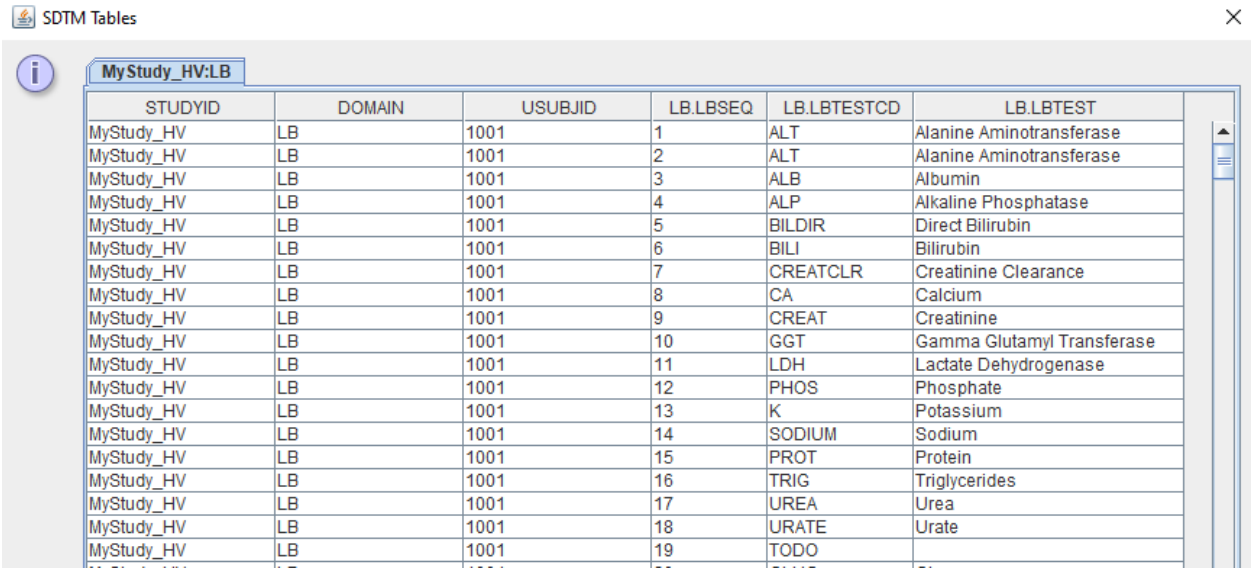
Just double-click "LB.LBTEST", and the following dialog is displayed:



and when clicking "Yes, please", the mapping for LBTEST just reduces to:



Essentially stating: for LBTEST, take the "decode" values of the codelist that was assigned to LBTESTCD. When then executing the mappings, this leads to:



STUDYID	DOMAIN	USUBJID	LB.LBSEQ	LB.LBTESTCD	LB.LBTEST
MyStudy_HV	LB	1001	1	ALT	Alanine Aminotransferase
MyStudy_HV	LB	1001	2	ALT	Alanine Aminotransferase
MyStudy_HV	LB	1001	3	ALB	Albumin
MyStudy_HV	LB	1001	4	ALP	Alkaline Phosphatase
MyStudy_HV	LB	1001	5	BILDIR	Direct Bilirubin
MyStudy_HV	LB	1001	6	BILI	Bilirubin
MyStudy_HV	LB	1001	7	CREATCLR	Creatinine Clearance
MyStudy_HV	LB	1001	8	CA	Calcium
MyStudy_HV	LB	1001	9	CREAT	Creatinine
MyStudy_HV	LB	1001	10	GGT	Gamma Glutamyl Transferase
MyStudy_HV	LB	1001	11	LDH	Lactate Dehydrogenase
MyStudy_HV	LB	1001	12	PHOS	Phosphate
MyStudy_HV	LB	1001	13	K	Potassium
MyStudy_HV	LB	1001	14	SODIUM	Sodium
MyStudy_HV	LB	1001	15	PROT	Protein
MyStudy_HV	LB	1001	16	TRIG	Triglycerides
MyStudy_HV	LB	1001	17	UREA	Urea
MyStudy_HV	LB	1001	18	URATE	Urate
MyStudy_HV	LB	1001	19	TODO	

In the case that no "decode" can be found, the value for LBTEST will just remain blank, as one can e.g. see in row 19, where we still have "TODO" for LBTESTCD.

Getting the collected values - Using a relative path as an alternative

In a "normal" ODM, the "Value" attribute of an item will contain the measured value, e.g.:

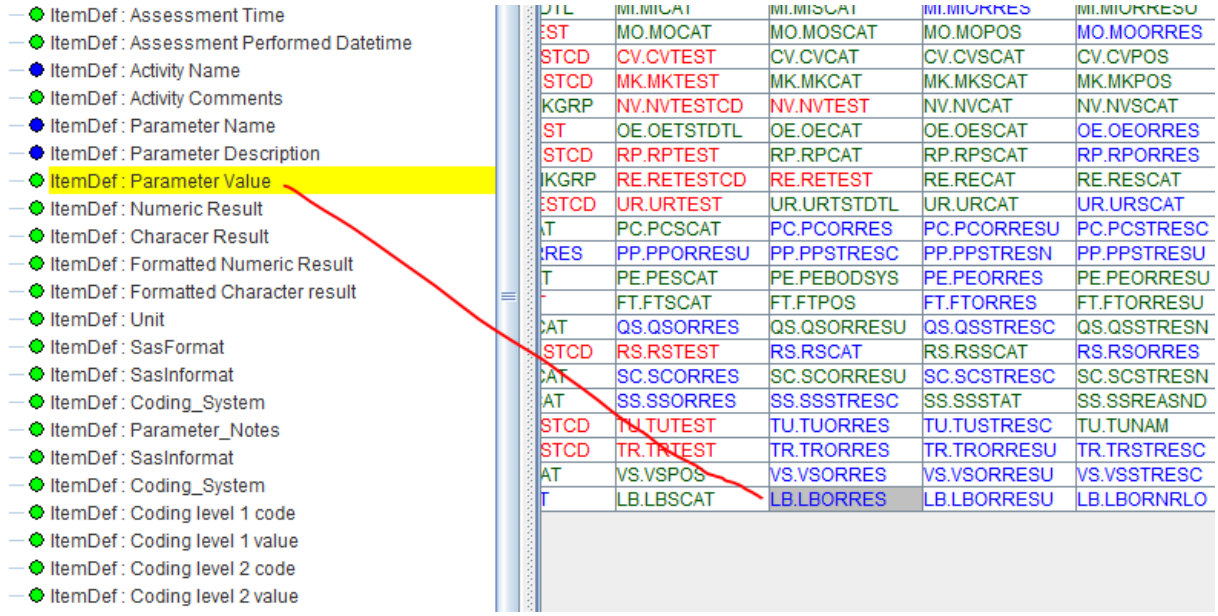
<ItemData ItemOID="IT.SYSBP" Value="110"/>

In a "hypervertical" structure, one Item will contain the name (parameter) of the test, and another, the value. For example:

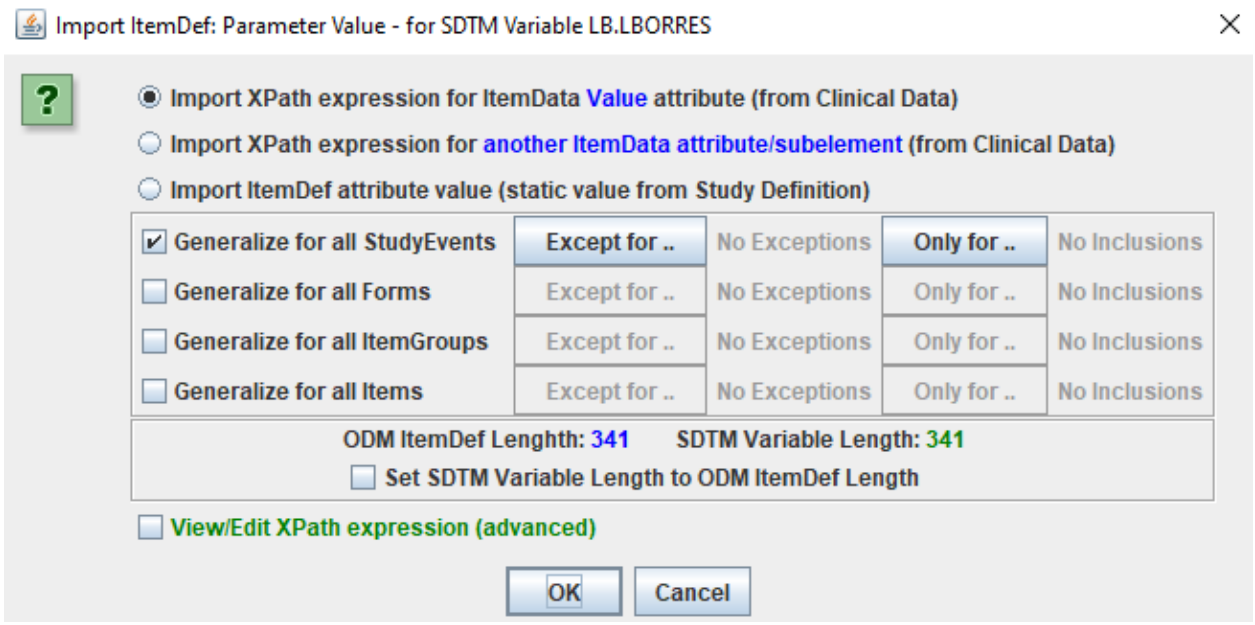
```
<ItemGroupData ItemGroupOID="IG.DEFAULT" ItemGroupRepeatKey="93">
    <ItemData ItemOID="IT.StudyID" Value="MyStudyHV"/>
    <ItemData ItemOID="IT.SubjectNr" Value="3"/>
    <ItemData ItemOID="IT.EventNumber" Value="0"/>
    <ItemData ItemOID="IT.ExpDeltaTime" Value="22"/>
    <ItemData ItemOID="IT.ActDeltaTime" Value="U"/>
    <ItemData ItemOID="IT.AssessmDateTime" Value="09NOV22:16:02:00"/>
    <ItemData ItemOID="IT.AssessmDate" Value="09NOV2022"/>
    <ItemData ItemOID="IT.AssessmTime" Value="16:02:00"/>
    <ItemData ItemOID="IT.AssessmPerfDatetime" Value="09NOV2022:16:02:00"/>
    <ItemData ItemOID="IT.ActivityName" Value="BsChem_A"/>
    <ItemData ItemOID="IT.ParameterName" Value="Albumin"/>
    <ItemData ItemOID="IT.ParameterDescription" Value="Albumin"/>
    <ItemData ItemOID="IT.ParameterValue" Value="45"/>
    <ItemData ItemOID="IT.ParameterValueNumeric" Value="45"/>
    <ItemData ItemOID="IT.ParameterValueNumericFormatted" Value="45"/>
    <ItemData ItemOID="IT.Unit" Value="g/L"/>
    <ItemData ItemOID="IT.SasFormat" Value="4.0"/>
    <ItemData ItemOID="IT.SasInformat" Value="BEST10."/>
</ItemGroupData>
```

where the value of the measurement is in another ItemData, with OID "IT.ParameterValue". So what to do?

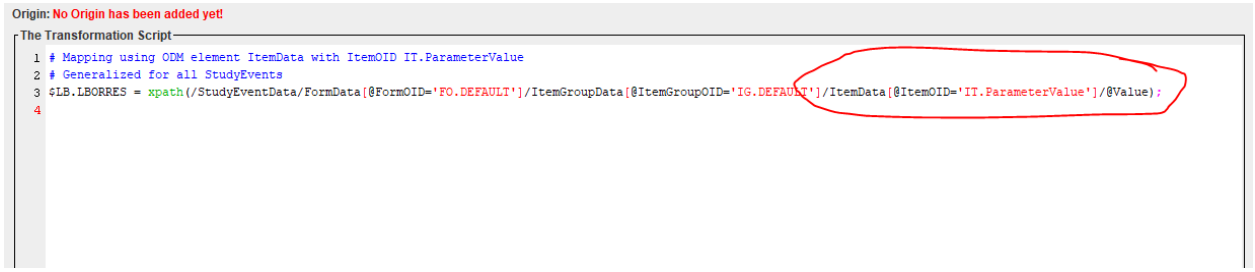
The first way is to use the classic way to just drag-and-drop, this time from "Parameter Value" to LBORRES:



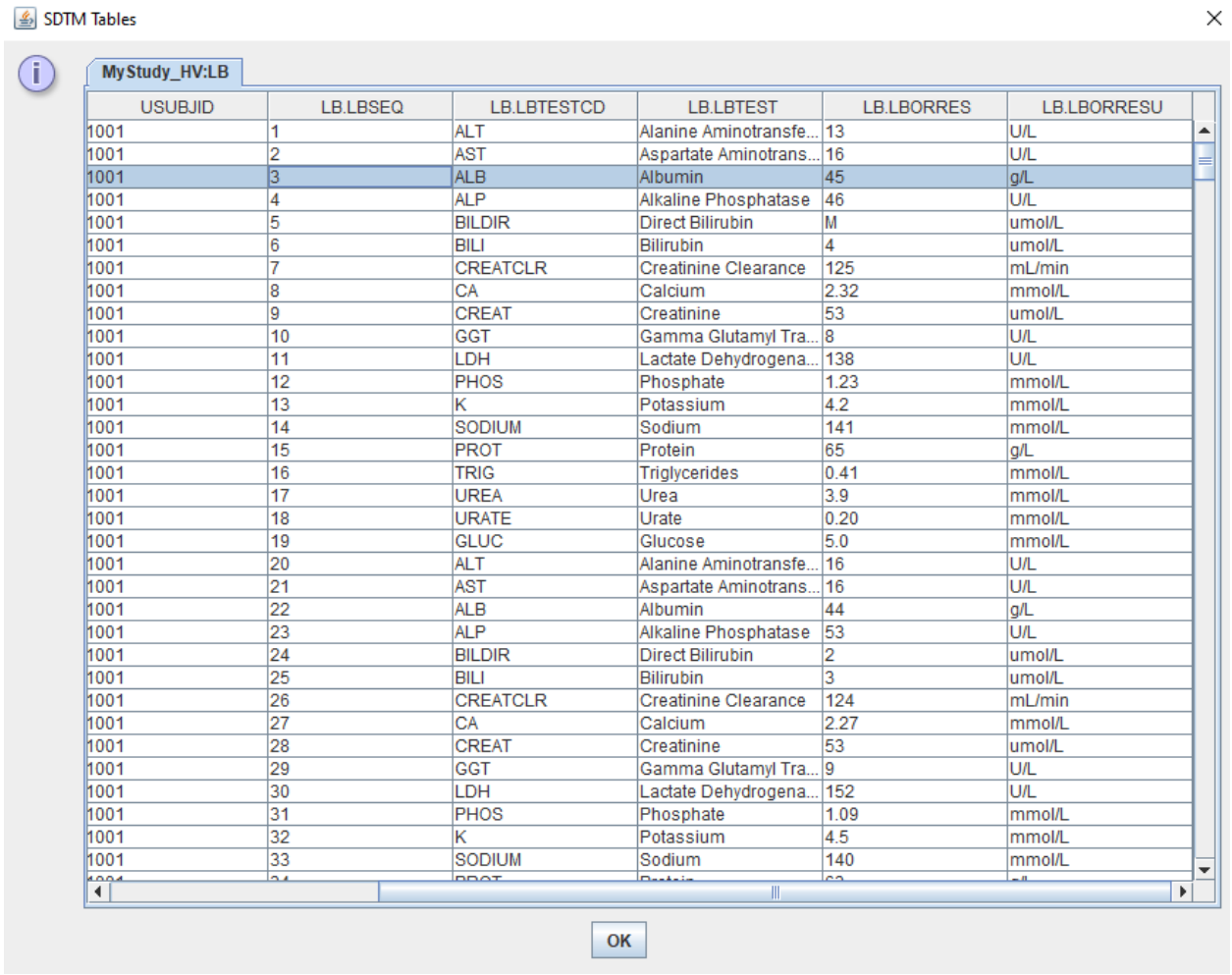
One can now just use "Import XPath expression for ItemData Value attribute ...":



leading to a mapping script:

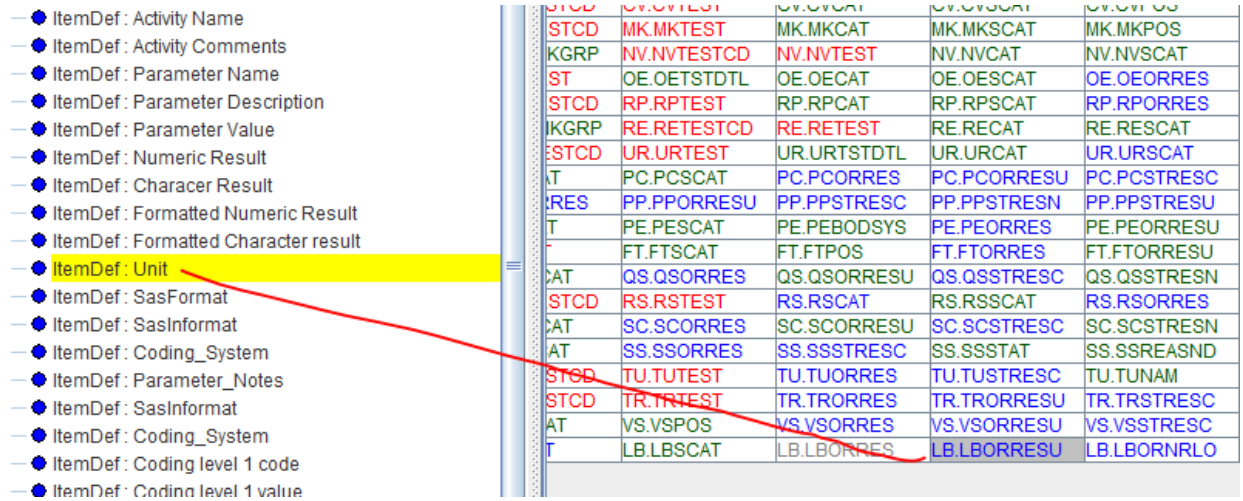


and to the result when executing the mappings:

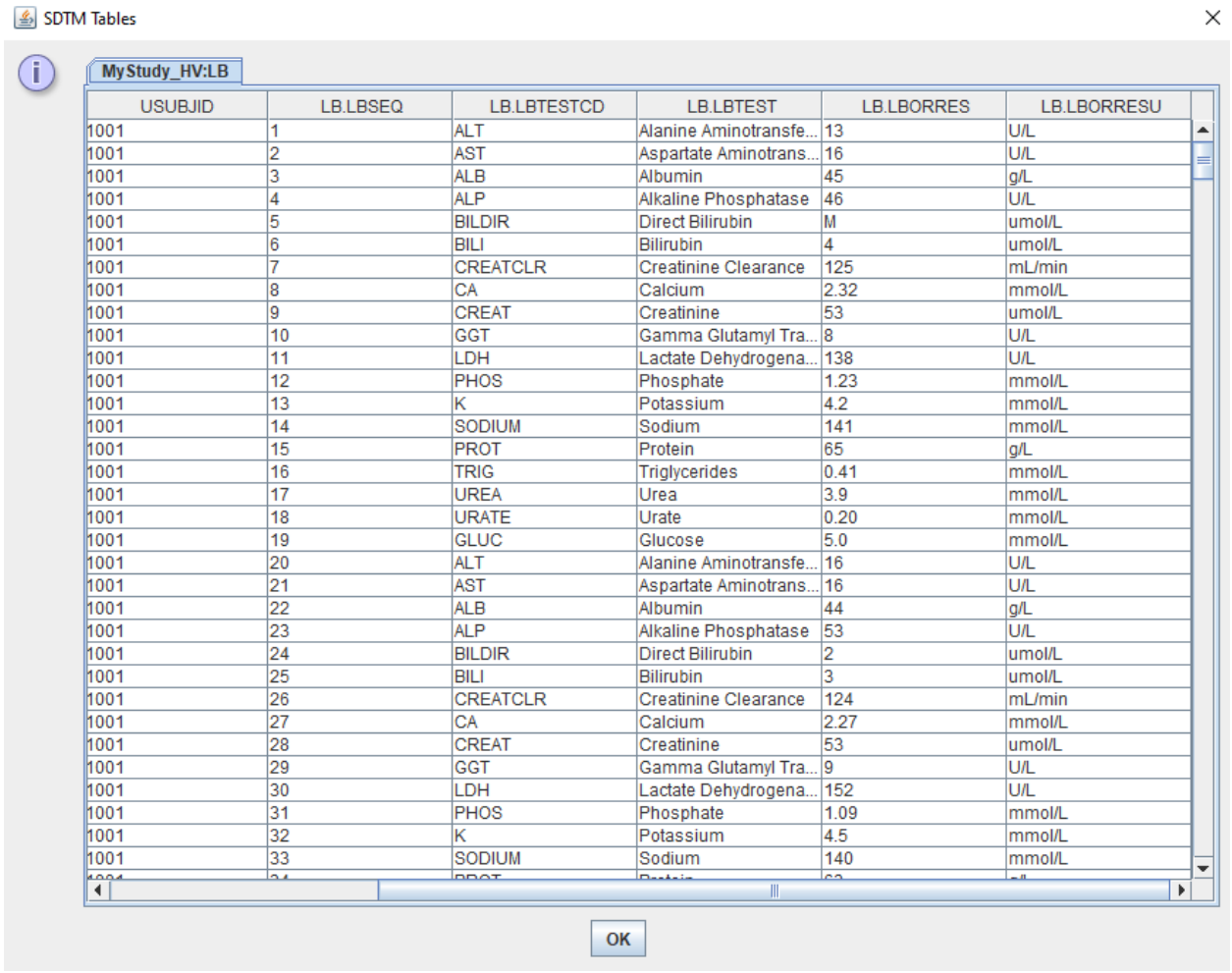


USUBJID	LB.LBSEQ	LB.LBTESTCD	LB.LBTEST	LB.LBORRES	LB.LBORRESU
1001	1	ALT	Alanine Aminotransfe...	13	U/L
1001	2	AST	Aspartate Aminotrans...	16	U/L
1001	3	ALB	Albumin	45	g/L
1001	4	ALP	Alkaline Phosphatase	46	U/L
1001	5	BILDIR	Direct Bilirubin	M	umol/L
1001	6	BILI	Bilirubin	4	umol/L
1001	7	CREATCLR	Creatinine Clearance	125	mL/min
1001	8	CA	Calcium	2.32	mmol/L
1001	9	CREAT	Creatinine	53	umol/L
1001	10	GGT	Gamma Glutamyl Tra...	8	U/L
1001	11	LDH	Lactate Dehydrogena...	138	U/L
1001	12	PHOS	Phosphate	1.23	mmol/L
1001	13	K	Potassium	4.2	mmol/L
1001	14	SODIUM	Sodium	141	mmol/L
1001	15	PROT	Protein	65	g/L
1001	16	TRIG	Triglycerides	0.41	mmol/L
1001	17	UREA	Urea	3.9	mmol/L
1001	18	URATE	Urate	0.20	mmol/L
1001	19	GLUC	Glucose	5.0	mmol/L
1001	20	ALT	Alanine Aminotransfe...	16	U/L
1001	21	AST	Aspartate Aminotrans...	16	U/L
1001	22	ALB	Albumin	44	g/L
1001	23	ALP	Alkaline Phosphatase	53	U/L
1001	24	BILDIR	Direct Bilirubin	2	umol/L
1001	25	BILI	Bilirubin	3	umol/L
1001	26	CREATCLR	Creatinine Clearance	124	mL/min
1001	27	CA	Calcium	2.27	mmol/L
1001	28	CREAT	Creatinine	53	umol/L
1001	29	GGT	Gamma Glutamyl Tra...	9	U/L
1001	30	LDH	Lactate Dehydrogena...	152	U/L
1001	31	PHOS	Phosphate	1.09	mmol/L
1001	32	K	Potassium	4.5	mmol/L
1001	33	SODIUM	Sodium	140	mmol/L

and similar for the unit, by drag-and-drop from "Unit" to LBORRESU:



leading to the result:



USUBJID	LB.LBSEQ	LB.LBTESTCD	LB.LBTEST	LB.LBORRES	LB.LBORRESU
1001	1	ALT	Alanine Aminotransfe...	13	U/L
1001	2	AST	Aspartate Aminotrans...	16	U/L
1001	3	ALB	Albumin	45	g/L
1001	4	ALP	Alkaline Phosphatase	46	U/L
1001	5	BILDIR	Direct Bilirubin	M	umol/L
1001	6	BILI	Bilirubin	4	umol/L
1001	7	CREATCLR	Creatinine Clearance	125	mL/min
1001	8	CA	Calcium	2.32	mmol/L
1001	9	CREAT	Creatinine	53	umol/L
1001	10	GGT	Gamma Glutamyl Tra...	8	U/L
1001	11	LDH	Lactate Dehydrogena...	138	U/L
1001	12	PHOS	Phosphate	1.23	mmol/L
1001	13	K	Potassium	4.2	mmol/L
1001	14	SODIUM	Sodium	141	mmol/L
1001	15	PROT	Protein	65	g/L
1001	16	TRIG	Triglycerides	0.41	mmol/L
1001	17	UREA	Urea	3.9	mmol/L
1001	18	URATE	Urate	0.20	mmol/L
1001	19	GLUC	Glucose	5.0	mmol/L
1001	20	ALT	Alanine Aminotransfe...	16	U/L
1001	21	AST	Aspartate Aminotrans...	16	U/L
1001	22	ALB	Albumin	44	g/L
1001	23	ALP	Alkaline Phosphatase	53	U/L
1001	24	BILDIR	Direct Bilirubin	2	umol/L
1001	25	BILI	Bilirubin	3	umol/L
1001	26	CREATCLR	Creatinine Clearance	124	mL/min
1001	27	CA	Calcium	2.27	mmol/L
1001	28	CREAT	Creatinine	53	umol/L
1001	29	GGT	Gamma Glutamyl Tra...	9	U/L
1001	30	LDH	Lactate Dehydrogena...	152	U/L
1001	31	PHOS	Phosphate	1.09	mmol/L
1001	32	K	Potassium	4.5	mmol/L
1001	33	SODIUM	Sodium	140	mmol/L

Reason is that the generated transformation (in XSLT) will look for the shortest path between the items one is iterating over (and which correspond to LBTESTCD) and the "ItemData" for the "Parameter Value" and "Unit" respectively.

However, this sometimes may go wrong, especially when one has changed the XPath expression for LBTESTCD. In such cases, there is however an easy method to still get everything right.

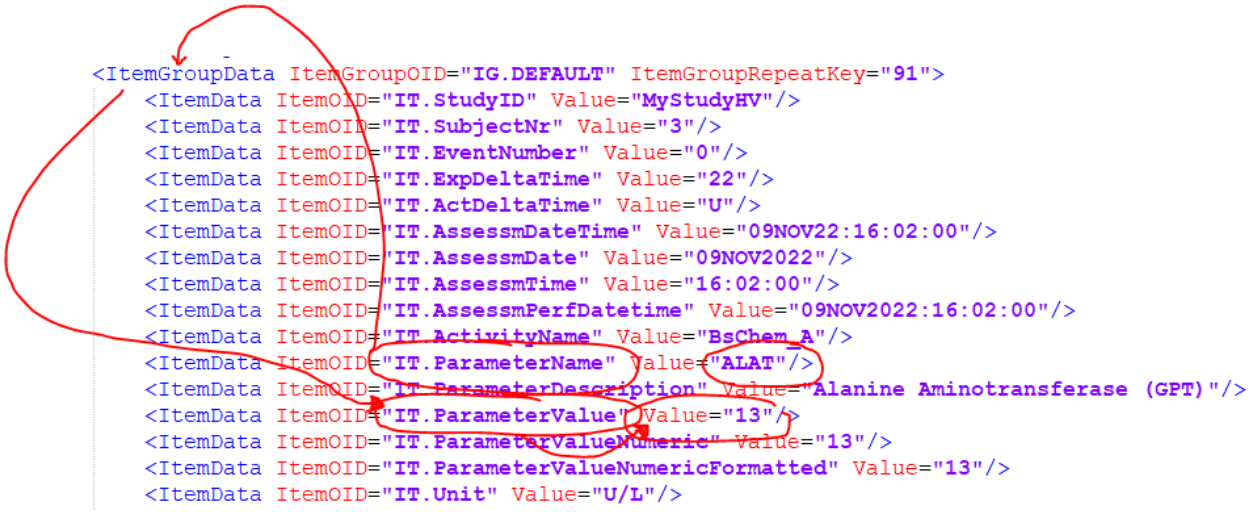
Essentially, the LBTESTCD is e.g. retrieved from the "ItemData" with ItemOID="IT.ParameterName" with value "ALAT":

```
<ItemGroupData ItemGroupOID="IG.DEFAULT" ItemGroupRepeatKey="91">
    <ItemData ItemOID="IT.StudyID" Value="MyStudyHV"/>
    <ItemData ItemOID="IT.SubjectNr" Value="3"/>
    <ItemData ItemOID="IT.EventNumber" Value="0"/>
    <ItemData ItemOID="IT.ExpDeltaTime" Value="22"/>
    <ItemData ItemOID="IT.ActDeltaTime" Value="U"/>
    <ItemData ItemOID="IT.AssessmDateTime" Value="09NOV22:16:02:00"/>
    <ItemData ItemOID="IT.AssessmDate" Value="09NOV2022"/>
    <ItemData ItemOID="IT.AssessmTime" Value="16:02:00"/>
    <ItemData ItemOID="IT.AssessmPerfDatetime" Value="09NOV2022:16:02:00"/>
    <ItemData ItemOID="IT.ActivityName" Value="BsChem_A"/>
    <ItemData ItemOID="IT.ParameterName" Value="ALAT"/>
    <ItemData ItemOID="IT.ParameterDescription" Value="Alanine Aminotransferase (GPT)"/>
    <ItemData ItemOID="IT.ParameterValue" Value="13"/>
    <ItemData ItemOID="IT.ParameterValueNumeric" Value="13"/>
    <ItemData ItemOID="IT.ParameterValueNumericFormatted" Value="13"/>
    <ItemData ItemOID="IT.Unit" Value="U/L"/>
```

and the relative path to the captured value is simply:

../ItemData[@ItemOID='IT.ParameterValue']/@Value

meaning: go one level up ("**..**"), then go down to the ItemData with the "ItemOID" "IT.ParameterValue" and then take the value of the "Value" attribute. Schematically:



So, the captured values can also be retrieved by the very simple script:

$LB.LBORRES = xpath(../ItemData[@ItemOID='IT.ParameterValue']/@Value);

Origin: No Origin has been added yet!

The Transformation Script

```
1 # Mapping using ODM element ItemData with ItemOID IT.ParameterValue
2 # Generalized for all StudyEvents
3 $LB.LBORRES = xpath(../ItemData[@ItemOID='IT.ParameterValue']/@Value);
4
```

Especially somewhat advanced users with some XPath knowledge prefer this method, writing their own XPath expressions, as it leads to clearer code and is more "bomb proof" than using the "drag-and-drop" method.

Similarly for LBORRESU:

Origin: No Origin has been added yet!

The Transformation Script

```
1 # Mapping using ODM element ItemData with ItemOID IT.Unit
2 # Generalized for all StudyEvents
3 $LB.LBORRESU = xpath(../ItemData[@ItemOID='IT.Unit']/@Value);
```

and leading to exactly the same result as above.

So, if one encounters problems with retrieving values for --ORRES and --ORRESU variables (but

also for other variables like timing variables) in ODMs with hypervertical structures, the "xpath" method is always a good alternative.

Developing mappings and assigning values for LBSPEC, LBSTRESC, LBSTRESN, LBSTRESU

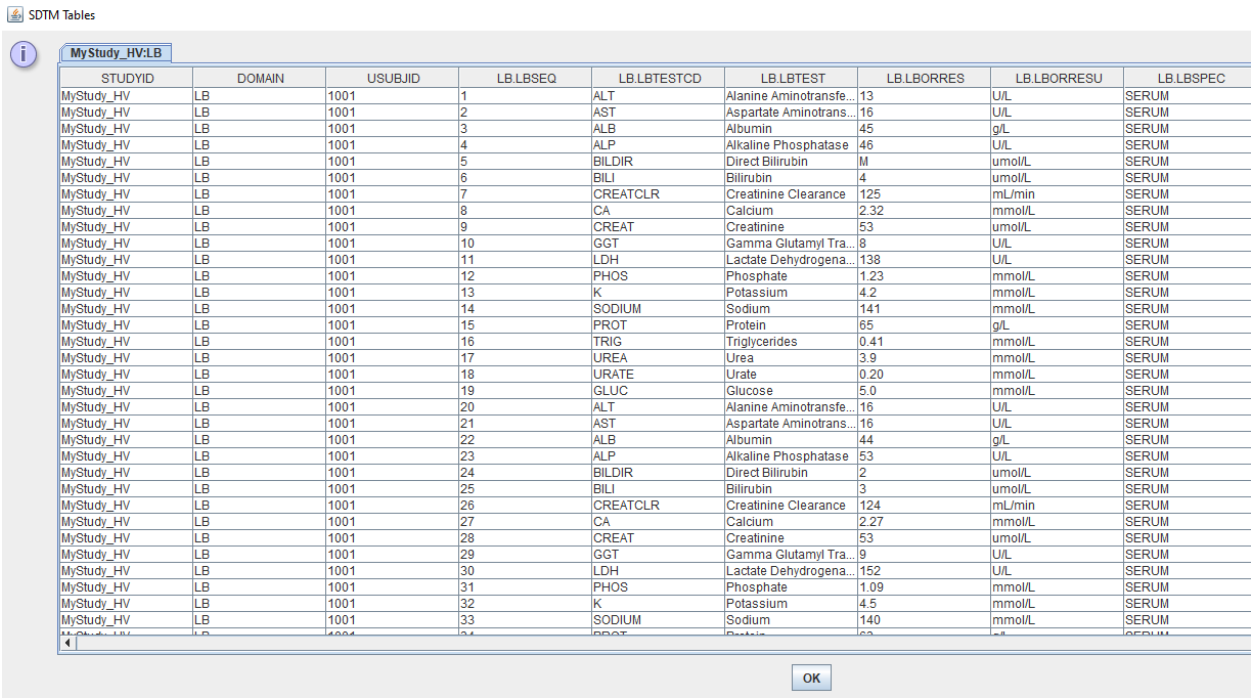
We will not go into much detail about doing unit conversions to "standardize" in LBSTRESN, LBSTRESC (usually a copy of LBSTRESN in case of numeric values) and LBSTRESU ("standardized unit"). This is well explained in other tutorials, such as "Performing Unit Conversions in SDTM-ETL" and "Using RESTful Web Services".
Especially when the LOINC code is available of the test, unit conversion from "US conventional" to "SI" units and the other way around, is very easy, and can be fully automated without the need for "conversion tables. Also, the regulatory requirements for the use of units differs between regulatory authorities, and even sometimes between reviewers within the same authority.

For LBSPEC, the case is simple here. As we limited our initial selection to blood serum tests, we can simply hard code as:

$LB.LBSPEC = 'SERUM';

In case of different specimen for different tests, one will either be able to use the "CodeList-CodeList Mapping" wizard (as also LBSPEC is under controlled terminology by CDISC), or using a relative simple "if-elsif-else" structure (see the base tutorials).

This then leads to:


SDTM Tables

MyStudy_HV:LB

STUDYID	DOMAIN	USUBJID	LB.LBSEQ	LB.LBTESTCD	LB.LBTEST	LB.LBORRES	LB.LBORRESU	LB.LBSPEC
MyStudy_HV	LB	1001	1	ALT	Alanine Aminotransfe...	13	U/L	SERUM
MyStudy_HV	LB	1001	2	AST	Aspartate Aminotrans...	16	U/L	SERUM
MyStudy_HV	LB	1001	3	ALB	Albumin	45	g/L	SERUM
MyStudy_HV	LB	1001	4	ALP	Alkaline Phosphatase	46	U/L	SERUM
MyStudy_HV	LB	1001	5	BILDIR	Direct Bilirubin	M	umol/L	SERUM
MyStudy_HV	LB	1001	6	BILI	Bilirubin	4	umol/L	SERUM
MyStudy_HV	LB	1001	7	CREATCLR	Creatinine Clearance	125	mL/min	SERUM
MyStudy_HV	LB	1001	8	CA	Calcium	2.32	mmol/L	SERUM
MyStudy_HV	LB	1001	9	CREAT	Creatinine	53	umol/L	SERUM
MyStudy_HV	LB	1001	10	GGT	Gamma Glutamyl Tra...	8	U/L	SERUM
MyStudy_HV	LB	1001	11	LDH	Lactate Dehydrogena...	138	U/L	SERUM
MyStudy_HV	LB	1001	12	PHOS	Phosphate	1.23	mmol/L	SERUM
MyStudy_HV	LB	1001	13	K	Potassium	4.2	mmol/L	SERUM
MyStudy_HV	LB	1001	14	SODIUM	Sodium	141	mmol/L	SERUM
MyStudy_HV	LB	1001	15	PROT	Protein	65	g/L	SERUM
MyStudy_HV	LB	1001	16	TRIG	Triglycerides	0.41	mmol/L	SERUM
MyStudy_HV	LB	1001	17	UREA	Urea	3.9	mmol/L	SERUM
MyStudy_HV	LB	1001	18	URATE	Urate	0.20	mmol/L	SERUM
MyStudy_HV	LB	1001	19	GLUC	Glucose	5.0	mmol/L	SERUM
MyStudy_HV	LB	1001	20	ALT	Alanine Aminotransfe...	16	U/L	SERUM
MyStudy_HV	LB	1001	21	AST	Aspartate Aminotrans...	16	U/L	SERUM
MyStudy_HV	LB	1001	22	ALB	Albumin	44	g/L	SERUM
MyStudy_HV	LB	1001	23	ALP	Alkaline Phosphatase	53	U/L	SERUM
MyStudy_HV	LB	1001	24	BILDIR	Direct Bilirubin	2	umol/L	SERUM
MyStudy_HV	LB	1001	25	BILI	Bilirubin	3	umol/L	SERUM
MyStudy_HV	LB	1001	26	CREATCLR	Creatinine Clearance	124	mL/min	SERUM
MyStudy_HV	LB	1001	27	CA	Calcium	2.27	mmol/L	SERUM
MyStudy_HV	LB	1001	28	CREAT	Creatinine	53	umol/L	SERUM
MyStudy_HV	LB	1001	29	GGT	Gamma Glutamyl Tra...	9	U/L	SERUM
MyStudy_HV	LB	1001	30	LDH	Lactate Dehydrogena...	152	U/L	SERUM
MyStudy_HV	LB	1001	31	PHOS	Phosphate	1.09	mmol/L	SERUM
MyStudy_HV	LB	1001	32	K	Potassium	4.5	mmol/L	SERUM
MyStudy_HV	LB	1001	33	SODIUM	Sodium	140	mmol/L	SERUM

OK

Timing Variables and date and time formatting

Some may already have seen that the collection date is formatted in a somewhat unusual way:

```
<ItemData ItemOID="IT.AssessmDateTime" Value="09NOV22:16:02:00"/>
<ItemData ItemOID="IT.AssessmDate" Value="09NOV2022"/>
<ItemData ItemOID="IT.AssessmTime" Value="16:02:00"/>
<ItemData ItemOID="IT.AssessmPerfDatetime" Value="09NOV2022:16:02:00"/>
<ItemData ItemOID="IT.ActivityName" Value="BsChem_A"/>
```

Even more complicated is that in some cases the time part is missing in which the value for "AssessmTime" is "U", and that in some cases, also the date itself is missing, e.g. when the test was not done, but there is still an entry in the source data. In such a case, one will need to decide whether to include that data point, or exclude such "not done" data points without a date or time right from the start. The latter can lead to somewhat more complex XPath expressions for LBTESTCD.

However, the SDTM standard requires that dates and times need to be formatted in ISO-8601 format. This then requires a mapping script like:

```
$COLLECTIONDATE = xpath(../ItemData[@ItemOID='IT.AssessmDate']/@Value);
$COLLECTIONTIME = xpath(../ItemData[@ItemOID='IT.AssessmTime']/@Value);
$DAY = substring($COLLECTIONDATE,1,2);
$MONTHSTR = substring($COLLECTIONDATE,3,3);
$YEAR = substring($COLLECTIONDATE,6,4);
if($MONTHSTR='JAN') {
     $MONTH = '01';
} elsif($MONTHSTR='FEB') {
     $MONTH = '02';
} elsif($MONTHSTR='MAR') {
     $MONTH = '03';
} elsif($MONTHSTR='APR') {
     $MONTH = '04';
} elsif($MONTHSTR='MAY') {
     $MONTH = '05';
} elsif($MONTHSTR='JUN') {
     $MONTH = '06';
} elsif($MONTHSTR='JUL') {
     $MONTH = '07';
} elsif($MONTHSTR='AUG') {
     $MONTH = '08';
} elsif($MONTHSTR='SEP') {
     $MONTH = '09';
} elsif($MONTHSTR='OCT') {
     $MONTH = '10';
} elsif($MONTHSTR='NOV') {
     $MONTH = '11';
} elsif($MONTHSTR='DEC') {
     $MONTH = '12';
}  else {
     $MONTH = 'INVALID';
}
# Sometimes the hour part does not have a leading '0' for hours before mid day
if(string-length($COLLECTIONTIME) = 7) {
     $TIME = concat('0',$COLLECTIONTIME);
} elsif($COLLECTIONTIME = 'U') {
     $TIME = '';
} else {
     $TIME = $COLLECTIONTIME;
}
if($COLLECTIONDATE != '' and $COLLECTIONDATE != 'U') {
     if($TIME != '') {
          $LB.LBDTC = concat($YEAR,'-',$MONTH,'-',$DAY,'T',$TIME);
     } else {
          $LB.LBDTC = concat($YEAR,'-',$MONTH,'-',$DAY);
     }
} else {
     $LB.LBDTC = '';
}
```

This script can be used over and over again for a lot of --DTC variables, just by copy-and-paste. However, if one usually obtains dates and times in this somewhat unusual format, it is well worth to invest some to and develop a "custom function" in XSLT. Such a custom function can e.g. be found in the file "functions.xsl" in the directory "stylesheets", as the "my_DateTimeToIso" function, which transforms dates and datetimes formatted as "ddMMMyyyyTaa:bb:cc" to ISO8601 date and datetime:

```
<xsl:function name="sdtm-etl:my_DateTimeToIso">
    <!-- takes a date formatted as ddMMMyyyyTaa:bb:cc (with the time parts being optional) and returns the ISO-8601 representation -->
    <xsl:param name="datetime"/>
    <xsl:variable name="BLANK"/>
    <xsl:variable name="MONTHPART" select="substring($datetime,3,3)"/>
    <xsl:variable name="DAYPART" select="substring($datetime,1,2)"/>
    <xsl:variable name="YEARPART" select="substring($datetime,6,4)"/>
    <xsl:variable name="TIMEPARTORIG" select="substring-after($datetime,'T')"/>
    <xsl:variable name="HOURSPARTORIG" select="tokenize($TIMEPARTORIG,':')[1]"/>
    <xsl:variable name="MINUTESPARTORIG" select="tokenize($TIMEPARTORIG,':')[2]"/>
    <xsl:variable name="SECONDSPARTORIG" select="tokenize($TIMEPARTORIG,':')[3]"/>
    <xsl:variable name="HOURSPART">
        <xsl:choose>
            <xsl:when test="string-length($HOURSPARTORIG) = 2"><xsl:value-of select="$HOURSPARTORIG"/></xsl:when>
            <xsl:when test="string-length($HOURSPARTORIG) = 1"><xsl:value-of select="concat('0',$HOURSPARTORIG)"/></xsl:when>
            <xsl:when test="string-length($HOURSPARTORIG) = 0"></xsl:when>
            <xsl:otherwise>INVALID_HOURS</xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
    <xsl:variable name="MINUTESPART">
        <xsl:choose>
            <xsl:when test="string-length($MINUTESPARTORIG) = 2"><xsl:value-of select="$MINUTESPARTORIG"/></xsl:when>
            <xsl:when test="string-length($MINUTESPARTORIG) = 1"><xsl:value-of select="concat('0',$MINUTESPARTORIG)"/></xsl:when>
            <xsl:when test="string-length($MINUTESPARTORIG) = 0"></xsl:when>
            <xsl:otherwise>INVALID_MINUTES</xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
    <xsl:variable name="SECONDSPART">
        <xsl:choose>
            <xsl:when test="string-length($SECONDSPARTORIG) = 2"><xsl:value-of select="$SECONDSPARTORIG"/></xsl:when>
            <xsl:when test="string-length($SECONDSPARTORIG) = 1"><xsl:value-of select="concat('0',$SECONDSPARTORIG)"/></xsl:when>
            <xsl:when test="string-length($SECONDSPARTORIG) = 0"></xsl:when>
            <xsl:otherwise>INVALID_SECONDS</xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
    <xsl:variable name="MONTH">
        <xsl:choose>
            <xsl:when test="$MONTHPART=''"></xsl:when><!-- empty (missing/incomplete)-->
            <xsl:when test="$MONTHPART='JAN'">01</xsl:when>
            <xsl:when test="$MONTHPART='FEB'">02</xsl:when>
            <xsl:when test="$MONTHPART='MAR'">03</xsl:when>
            <xsl:when test="$MONTHPART='APR'">04</xsl:when>
```

The date/time format we have in our case (see above) is only slightly different from the one used by the "my_DateTimeToIso" function, so that one can easily adapt it, or add a new one based on this example function. Such "custom functions" of course need a bit of time to develop and test, but this investment is very well worth spending. And of course you can always ask us to develop a "custom function" for you.

For the timing variables VISITNUM and VISIT, one can again use drag and drop, starting from the StudyEvent level, or use a relative path in the script. For example:

Origin: No Origin has been added yet!

The Transformation Script

```
1 $VISITID = xpath(../../../@StudyEventOID);
2 $VISITNUM = substring-after($VISITID,'.');
3 $LB.VISIT = concat('VISIT ',$VISITNUM);
```

as the StudyEvent-OID values are "SE.0", "SE.1, ... "SE.99".
The XPath expression essentially means: "go three levels up (to StudyEvent) and take the value of the StudyEventOID attribute".

For the -DY variables, one will probably want to first generate a "GLOBAL" domain, to store the reference date that can then be used in all mappings. Please see the tutorial "Creating and working with Subject Global Variables".

Conclusions

Working with ODM files representing "hypervertical" structures in SDTM-ETL is slightly different from working with "classic" ODM files where each Item ("ItemDef" - "ItemData" pair) represents a single test or question on a form or from a data transfer.

In the case of "hypervertical" structures, the most prominent differences are:

- the "Item" in an hypervertical structure no longer represents a single data point, but only one of the attributes of a test or question, which is represented by the parent "ItemGroup".

- thus, when doing drag-and-drop of an Item that represents the code or identifier for the test (e.g. "Parameter Name", to a "--TESTCD" variable in a Findings domain instance, one needs to select "Import XPath expression for ItemData Value attribute (from ClinicalData)" as the latter contains the identifier of the test or question. See section "Generating the mapping for LBTESTCD".

- In the next step, when the system proposes to use the "CodeList-CodeList Mapping Wizard", ensure that the checkbox "First make a pre-selection of the ODM coded values" is checked, allowing to select the codes for the parameters that are representing the tests for the specific domain (or subset domain).

- when doing drag-and-drop of an Item that represents the value for the test (or question answer), e.g. "Parameter Value", to --ORRES (or --TERM in the case of Events) variables one can (as in the case of "classic" ODM), use "Import XPath expression for ItemData Value attribute ..."

- for further "qualifier" SDTM variables such as --SPEC, one can also use drag-and-drop. When this leads to problems (when one has e.g. manually edited the XPath expression for the selection), one can also use a relative XPath expression, which usually reduces to something like:
$AA.AAAA = xpath(../ItemData[@ItemOID='XXX']/@Value);
where $AA.AAAA represents the SDTM variable and XXX represents the ItemOID of the item in the OID structure.

- When dates and date-times in the source are not in ISO-8601 format (required by SDTM) yet, one will need to generate a mapping script for date/time conversions to ISO-8601. In such a case, one may consider to generate a function for this in XSLT and add it to the file "functions.xsl" in the "stylesheets" folder. If there is no in-house XSLT knowledge, you can always ask us to develop such a function.