PharmaSUG 2018 - Paper SS-11

Accessing the Metadata from Define-XML

Lex Jansen, SAS Institute Inc., Cary, NC, USA

ABSTRACT

Many papers have been written about creating Define-XML documents. This paper is about what we can do when we receive a Define-XML Document. We can view the Define-XML document in a browser by using a stylesheet, but how do we access the metadata in the Define-XML document? Once we have access to the metadata we can use it to drive automation. This paper will discuss various technologies to extract the metadata from a Define-XML document, like SAS XML Mapper, SAS PROC GROOVY, SAS PROC XSLT and the SAS Clinical Standards Toolkit.

This paper assumes that the reader is familiar with some basic XML concepts, and Define-XML version 2. An earlier paper by the author (see references Jansen (2012) [1]) contains both a brief overview of the XML needed to understand this paper, and an overview of the structure of a define.xml file based on Define-XML version 1.0.0. A detailed overview of differences between CRT-DDS version 1.0.0 and Define-XML version 2 can be found in "Define-XML v2 - What's New" (Jansen (2013) [2]).

Keywords: CDISC, Define-XML, define.xml, metadata, SAS XML Mapper, Groovy, XSLT, SAS Clinical Standards Toolkit

The code in this paper was developed with SAS 9.4TS Level 1M4 on Windows 7 Pro (x64).
After PharmaSUG 2018 the complete code will be available for download at:
https://www.lexjansen.com/pharmasug/2018/SS/SS11.zip

INTRODUCTION

In March 2013 the final version of the Define-XML 2.0.0 standard [3], formerly known as CRT-DDS (Case Report Tabulation Data Definition Specification) or "define.xml", as most people called it, was released by the CDISC XML Technologies team. Define-XML 2.0.0 is a major revision of the Define-XML standard for transmission of SDTM, SEND and ADaM metadata. Version 1.0.0 was released for implementation in February 2005 [4]. Define-XML has been a useful mechanism and critical component for providing Case Report Tabulations Data Definitions in an XML format for CDISC based electronic submissions to a regulatory authority such as the U.S. Food and Drug Administration (FDA). In August 2013 the FDA started accepting Define-XML version 2.0.0 [5]. In the latest version of the Study Data Technical Conformance Guide (version 4.0, October 2017) the FDA states that Define-XML version 2.0.0 is the preferred version of the Define-XML format [6]. In March 2016 the FDA had already announced the Support end date for CRT-DDS version 1.0.0 [7].

USING DEFINE-XML METADATA

Often, the Define-XML document will be created after the SAS data sets have been created. However, it would also be possible to use Define-XML as a specification for the SAS data sets. The Define-XML document will hold all the metadata that describes the data sets (name, label) and variables in the data sets (name, label, datatype, length, controlled terminology). By using the metadata from the Define-XML document we can create 0-observation data sets, which can act as data set specifications.

Another use case is that we can import the study metadata from a Define-XML document and use that metadata as a starting point for the metadata in a similar study.

This author wrote 2 earlier – award winning - papers about accessing the metadata in Define-XML. One paper dealt with using the metadata from Define-XML to create a PDF representation of Define-XML (Jansen (2008), [8]). The other paper was about accessing and using the metadata from a Define-XML document using XSLT transformations (Jansen (2010) [9]). Both were about Define-XML v1.

METADATA IN A DEFINE-XML DOCUMENT

When submitting clinical study data in electronic format to the FDA, or other regulatory agencies, not only information from trials must be submitted, but also information to help understand the data. Part of this information is a data definition file, which is the metadata describing the format and content of the submitted electronic data sets. When submitting data in CDISC format it is required to submit the data definition file in the Define-XML specification format (define.xml) as prepared by the CDISC define.xml team.

DEFINE-XML METADATA CONTENT

A Define-XML file is a structured data definition specification in a machine-readable XML format that provides various kinds of metadata for:

- Study:
    - Study name, study description, protocol name
- Data sets:
    - Name, domain, label, class, structure, purpose, keys, data location, comments, documentation
- Variables:
    - Name, label, type, length, controlled terminology, origin, significant digit, display format, derivations, comments, documentation
- Variables under a condition:
    - Value level metadata or parameter value level metadata with the same kind of metadata as for 'regular' variables
- Controlled Terminology:
    - Standard or sponsor defined, name, type, valid values, decodes, reference to NCI code, external terminologies
- Derivations or Algorithms
- Comments
- Links to submission files:
    - Annotated CRF, Reviewers' Guides, source code files

For analysis data sets the Define-XML standard provides Analysis Results Metadata, which is the metadata needed for traceability from a result used in a statistical display to the data in the analysis data sets:

- Identifiers and titles for the analysis displays (tables, figures) in the clinical study report
- Description of the specific analysis result within a display
- Purpose and reasons for performing the analysis
- The analysis parameter that is the focus of the analysis result
- Variables subject to analysis
- Data sets used to generate the analysis result
- Selection criteria for the records subject to analysis
- Corresponding description in the statistical analysis plan, analysis program name, and summary

of the analytical methods

- Extract of the analysis program corresponding to the analysis method

TEMPLATES FOR THE SAS TARGET DATA SETS

In this paper we will illustrate the import of metadata from a Define-XML document for data set and variable metadata. We are using templates from the SAS Clinical Standards Toolkit to describe the target data sets that we will extract from the Define-XML document.

```
* Build source table metadata data set template. *;
create table csttmp.studytablemetadata(label='Source Table Metadata')
  (
   sasref char(8) label='SASreferences sourcedata libref',
   table char(32) label='Table Name',
   label char(200) label='Table Label',
   order num label='Table order',
   repeating char(3)
     label="Can itemgroup occur repeatedly within the containing form?",
   isreferencedata char(3)
     label="Can itemgroup occur only within a ReferenceData element?",
   domain char(32) label='Domain',
   domaindescription char(256) label='Domain description',
   class char(40) label='Observation Class within Standard',
   xmlpath char(200) label='(Relative) path to xpt file',
   xmltitle char(200) label='Title for xpt file',
   structure char(200) label='Table Structure',
   purpose char(10) label='Purpose',
   keys char(200) label='Table Keys',
   state char(20) label='Data Set State (Final, Draft)',
   date char(20) label='Release Date',
   comment char(1000) label='Comment',
   studyversion char(128) label='Unique study version identifier',
   standard char(20) label='Name of Standard',
   standardversion char(20) label='Version of Standard'
  );

* Build source column metadata data set template. *;
create table csttmp.studycolumnmetadata(label='Source Column Metadata')
  (
   sasref char(8) label='SASreferences sourcedata libref',
   table char(32) label='Table Name',
   column char(32) label='Column Name',
   label char(200) label='Column Description',
   order num label='Column Order',
   type char(1) label='Column Type',
   length num label='Column Length',
   displayformat char(200) label='Display Format',
   significantdigits num label='Significant Digits',
   xmldatatype char(18) label='XML Data Type',
   xmlcodelist char(128) label='SAS Format/XML Codelist',
   core char(10) label='Column Required or Optional',
   origin char(40) label='Column Origin',
   origindescription char(1000) label='Column Origin Description',
   role char(200) label='Column Role',
```

```
  algorithm char(1000) label='Computational Algorithm or Method',
  algorithmtype char(11) label='Type of Algorithm',
  formalexpression char(1000) label='Formal Expression for Algorithm',
  formalexpressioncontext char(1000)
    label='Context to be used when evaluating the FormalExpression content',
  comment char(1000) label='Comment',
  studyversion char(128) label='Unique study version identifier',
  standard char(20) label='Name of Standard',
  standardversion char(20) label='Version of Standard'
 );
```

These metadata templates contain a few CST specific variables that are not created from the Define-XML document: sasref, state and core.

We will see some other interesting use cases for these templates:

- Definition of LENGTH and INPUT statements when reading CSV files
- Automatic creation of an XMLMap.

DEFINE-XML EXAMPLE

Below is an example of the Define-XML content that we will import.

```
<ItemGroupDef OID="IG.DM" Domain="DM" Name="DM" Repeating="No" IsReferenceData="No"
  SASDatasetName="DM" Purpose="Tabulation" def:Structure="One record per subject"
  def:Class="SPECIAL PURPOSE" def:CommentOID="COM.DOMAIN.DM"
  def:ArchiveLocationID="LF.DM">
  <Description>
    <TranslatedText xml:lang="en">Demographics</TranslatedText>
  </Description>
  <ItemRef ItemOID="IT.STUDYID" OrderNumber="1" Mandatory="Yes" KeySequence="1"/>
  <ItemRef ItemOID="IT.DM.DOMAIN" OrderNumber="2" Mandatory="Yes"/>
  <ItemRef ItemOID="IT.USUBJID" OrderNumber="3" Mandatory="Yes" KeySequence="2"
    MethodOID="MT.USUBJID"/>
  <ItemRef ItemOID="IT.DM.SUBJID" OrderNumber="4" Mandatory="Yes"/>
  <ItemRef ItemOID="IT.DM.RFSTDTC" OrderNumber="5" Mandatory="No" MethodOID="MT.RFSTDTC"/>
  <ItemRef ItemOID="IT.DM.RFENDTC" OrderNumber="6" Mandatory="No" MethodOID="MT.RFENDTC"/>
  <ItemRef ItemOID="IT.DM.SITEID" OrderNumber="7" Mandatory="Yes"/>
  <ItemRef ItemOID="IT.DM.BRTHDTC" OrderNumber="8" Mandatory="No"/>
  <ItemRef ItemOID="IT.DM.AGE" OrderNumber="9" Mandatory="Yes" MethodOID="MT.AGE"/>
  <ItemRef ItemOID="IT.DM.AGEU" OrderNumber="10" Mandatory="No"/>
  <ItemRef ItemOID="IT.DM.SEX" OrderNumber="11" Mandatory="Yes"/>
  <ItemRef ItemOID="IT.DM.RACE" OrderNumber="12" Mandatory="No"/>
  <ItemRef ItemOID="IT.DM.ETHNIC" OrderNumber="13" Mandatory="Yes"/>
  <ItemRef ItemOID="IT.DM.ARMCD" OrderNumber="14" Mandatory="Yes"/>
  <ItemRef ItemOID="IT.DM.ARM" OrderNumber="15" Mandatory="Yes"/>
  <ItemRef ItemOID="IT.DM.COUNTRY" OrderNumber="16" Mandatory="Yes"/>
  <def:leaf ID="LF.DM" xlink:href="dm.xpt">
    <def:title>dm.xpt</def:title>
  </def:leaf>
</ItemGroupDef>

<ItemDef OID="IT.USUBJID" Name="USUBJID" DataType="text" Length="14"
  SASFieldName="USUBJID">
  <Description>
    <TranslatedText xml:lang="en">Unique Subject Identifier</TranslatedText>
```

```
        </Description>
        <def:Origin Type="Derived"/>
      </ItemDef>

      <ItemDef OID="IT.DM.ARMCD" Name="ARMCD" DataType="text" Length="8"
        SASFieldName="ARMCD" def:CommentOID="COM.ARMCD">
        <Description>
          <TranslatedText xml:lang="en">Planned Arm Code</TranslatedText>
        </Description>
        <CodeListRef CodeListOID="CL.ARMCD"/>
        <def:Origin Type="Assigned"/>
      </ItemDef>

      <MethodDef OID="MT.USUBJID" Name="Algorithm to derive USUBJID" Type="Computation">
        <Description>
          <TranslatedText xml:lang="en">Concatenation of STUDYID and SUBJID</TranslatedText>
        </Description>
        <FormalExpression Context="SAS 9.0 or later, as part of a data step assignment or proc
sql select and update statements.">
          catx(".",STUDYID,SUBJID)
        </FormalExpression>
      </MethodDef>

      <def:CommentDef OID="COM.ARMCD">
        <Description>
          <TranslatedText xml:lang="en">Assigned based on Randomization Number. See Note
2.1</TranslatedText>
        </Description>
      </def:CommentDef>
```

TECHNOLOGIES FOR ACCESSING THE METADATA IN A DEFINE-XML DOCUMENT

THE SAS XML LIBNAME ENGINE AND THE SAS XML MAPPER

A good overview of the capabilities of SAS to work with XML files can be found in Schacherer (2014) [10] and Cox (2012) [11].

SAS can read generic XML files with the XML LIBNAME engine, which have the following characteristics:

- The enclosing **root element** is comparable to a SAS **library**
- A second-level element is translated to a **dataset** name
- Other elements within that second level become SAS **variables**

A Define-XML document is clearly too hierarchical to be able to use the XML LIBNAME engine without a so-called XMLMap. To be able to read the Define-XML document with SAS and create SAS data sets we need to create an XMLMap. The XMLMap tells the SAS XML engine how to map the content in the hierarchical Define-XML document to rows and columns in the rectangular SAS tables (SAS, 2013 [12]).

The SAS XMLMap supports a subset of the XPath specification to define the locations of different SAS data set components within an XML document. Below is an example.

```
<?xml version="1.0" encoding="UTF-8"?>
<SXLEMAP name="AUTO_GEN" version="2.1">

 <NAMESPACES count="4">
   <NS id="1" prefix="">http://www.cdisc.org/ns/odm/v1.3</NS>
   <NS id="2" prefix="xlink">http://www.w3.org/1999/xlink</NS>
```

```
   <NS id="3" prefix="def">http://www.cdisc.org/ns/def/v2.0</NS>
   <NS id="4" prefix="xml">http://www.w3.org/XML/1998/namespace</NS>
 </NAMESPACES>

 <TABLE description="ItemGroupDef" name="ItemGroupDef">
   <TABLE-PATH syntax="XPathENR">/{1}ODM/{1}Study/{1}MetaDataVersion/{1}ItemGroupDef</TABLE-PATH>

   <COLUMN class="ORDINAL" name="MetaDataVersion_ORDINAL">
     <INCREMENT-PATH
       beginend="BEGIN" syntax="XPathENR">/{1}ODM/{1}Study/{1}MetaDataVersion</INCREMENT-PATH>
     <TYPE>numeric</TYPE>
     <DATATYPE>integer</DATATYPE>
   </COLUMN>

   <COLUMN class="ORDINAL" name="ItemGroupDef_ORDINAL">
     <INCREMENT-PATH beginend="BEGIN"
       syntax="XPathENR">/{1}ODM/{1}Study/{1}MetaDataVersion/{1}ItemGroupDef</INCREMENT-PATH>
     <TYPE>numeric</TYPE>
     <DATATYPE>integer</DATATYPE>
   </COLUMN>

   <COLUMN name="OID">
     <PATH syntax="XPathENR">/{1}ODM/{1}Study/{1}MetaDataVersion/{1}ItemGroupDef/@OID</PATH>
     <TYPE>character</TYPE>
     <DATATYPE>string</DATATYPE>
     <LENGTH>11</LENGTH>
   </COLUMN>

   <COLUMN name="Name">
     <PATH syntax="XPathENR">/{1}ODM/{1}Study/{1}MetaDataVersion/{1}ItemGroupDef/@Name</PATH>
     <TYPE>character</TYPE>
     <DATATYPE>string</DATATYPE>
     <LENGTH>8</LENGTH>
   </COLUMN>

   <COLUMN name="Class">
     <PATH syntax="XPathENR">/{1}ODM/{1}Study/{1}MetaDataVersion/{1}ItemGroupDef/@{3}Class</PATH>
     <TYPE>character</TYPE>
     <DATATYPE>string</DATATYPE>
     <LENGTH>15</LENGTH>
   </COLUMN>
   ...

 </TABLE>
...
</SXLEMAP>
```

This XMLMap in specifies that the SAS data set **ItemGroupDef** has a character variable named **Class** with length 15, whose content is defined by the XPath:

/{1}ODM/{1}Study/{1}MetaDataVersion/{1}ItemGroupDef/@{3}Class

Notice how the namespace identifier (id=”3”) is embedded in the XPath syntax to locate the Class attribute in the http://www.cdisc.org/ns/def/v2.0 namespace.

Once we have created a SAS XMLMap we can create SAS data sets as follows:

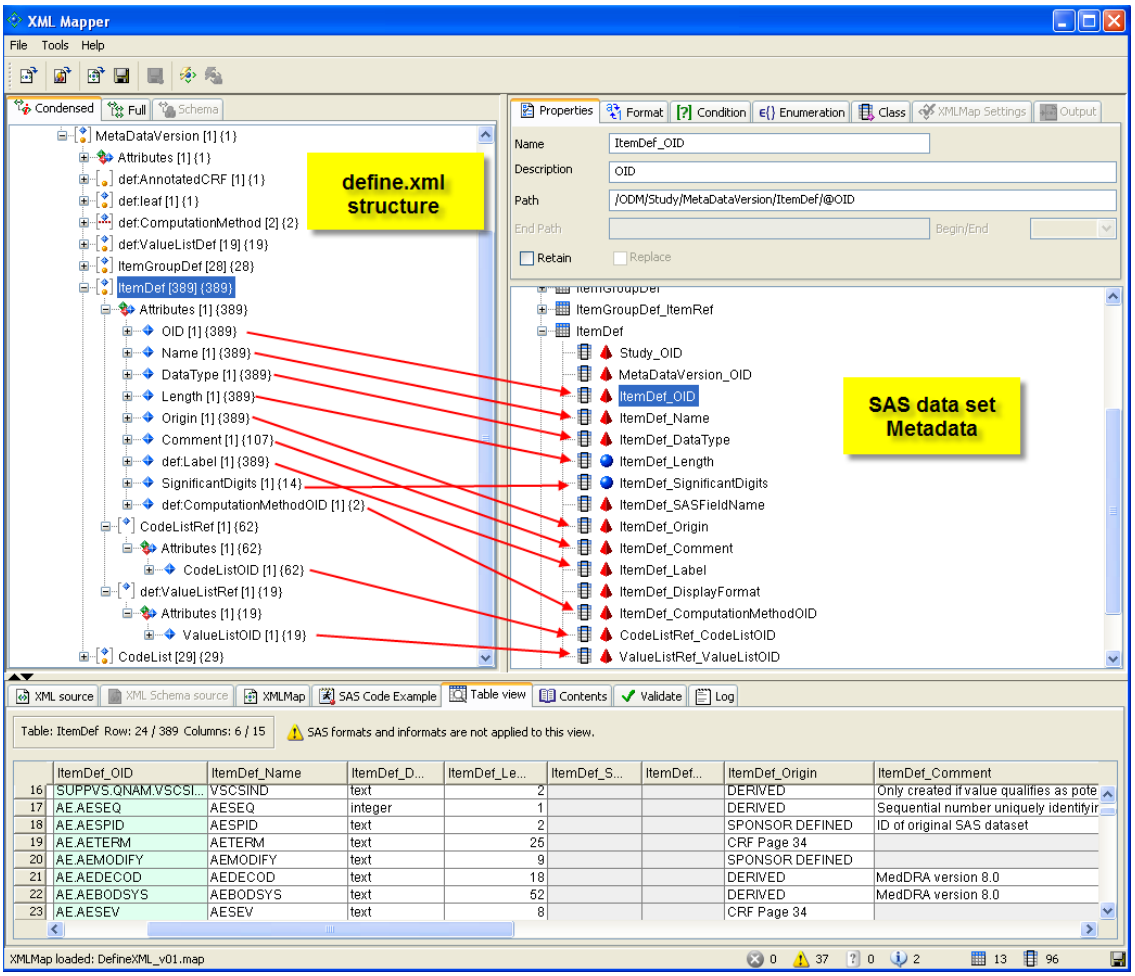
```
%let Root=C:/_Data/Presentations/PharmaSUG_2018/Accessing_DefineXML;

libname out "&Root/xmlmap/out";

filename define "&Root/xml/define2-0-0-example-sdtm.xml";
filename map "&Root/xmlmap/definexml_auto.map";
libname define xmlv2 xmlmap=map;

proc copy in=define out=out;
run;
```

There are different strategies to build an XMLMap file. SAS XML Mapper is an XMLMap support tool for the XML engine. Based on Java, SAS XML Mapper is a stand-alone application that removes the tedious work of creating and modifying an XMLMap.

SAS XML Mapper provides a graphical interface that you can use to generate the appropriate XML elements. SAS XML Mapper analyzes the structure of an XML document or an XML schema and generates basic XML syntax for the XMLMap.

The interface consists of windows, a menu bar, and a toolbar. Using SAS XML Mapper, you can display an XML document or an XML schema, create and modify an XMLMap, and generate example SAS programs.

SAS XML Mapper interface



Starting in SAS 9.1.3 SP3 the automapping functionality in SAS XML Mapper was introduced. XML Mapper is very good at making sure the hierarchical relationships identified in the data are maintained in a way that makes it simple to join data in tables that are subordinate to each other. This is done through the generation of surrogate primary and foreign key values (i.e., the "_ORDINAL" variables). Therefore, even though XML Mapper creates a number of separate tables in order to render all of the data in two dimensions, the hierarchical relationships between those tables are maintained. As Cox (2012) [11] points out, "the automapping algorithm creates a representation of the XML that is as relational as possible from the available context. It does this, along with the creation of surrogate keys, so that the tables can be rejoined using PROC SQL.".

The automapping functionality in SAS XML Mapper was never intended to fully replace the drag-and-drop creation of XMLMaps.

In the SAS XML Mapper application a map can be created based on the XML schema that describes the Define-XML documents or based on a specific instance of the Define-XML document. Also, the map can be created with drag-and-drop or by automapping

Following are a few challenges to keep in mind when creating a SAS XMLMap using the SAS XML Mapper application.

- When using an instance of a Define-XML document to create an XMLMap, one needs to make sure that this instance contains all elements and attributes that can be expected in a Define-XML document.
- When using the automapper, it may be necessary to tweak the lengths of the variables to be created, since the SAS XML Mapper application has no knowledge - based on an XML instance – what the maximum length is that can be expected.
- When using an instance of a Define-XML document with the automapper, the map may need to be tweaked to correct the type of the variable to be created. A good example is the DisplayFormat attribute in Define-XML. When the XML instance only has values like “5.1” and “4.2”, the automapper will create a numeric variable, even though we know that it should be a character variable.
  XML data that does not have any accompanying metadata (XML Schema) is character data, and must not be inferred to be otherwise at the core processing level by default. In contrast, the XML libname engine by default assumes types other than character.
- When using the automapper with an XML schema as input, there is a bug in the XML Mapper. Namespaces are only added to elements and not to attributes:

  /{1}ODM/{1}Study/{1}MetaDataVersion/{1}ItemGroupDef/@Class

  instead of:

  /{1}ODM/{1}Study/{1}MetaDataVersion/{1}ItemGroupDef/@**{3}**Class

  The namespaces can be added manually, but this quickly becomes extremely tedious and error prone.
  See: https://support.sas.com/kb/61/963.html

This paper used the automapper on an instance of a Define-XML document that contained all elements and attributes to be expected. No efforts were made to tweak the lengths of the variables to be created. The final variable lengths were set by using the metadata templates that were mentioned earlier in this paper. The automapping process created 51 data sets.

We can let the automapper create an SAS XMLMap and extract the SAS data sets as follows:

```
%let Root=C:/_Data/Presentations/PharmaSUG_2018/Accessing_DefineXML;

libname out "&Root/xmlmap/out";

filename define "&Root/xml/define2-0-0-example-sdtm.xml";
filename map "&Root/xmlmap/definexml_auto.map";
libname define xmlv2 automap=reuse xmlmap=map prefixattributes=no;

proc copy in=define out=out;
run;
```

automap=reuse does not overwrite an existing XMLMap file. If an XMLMap file exists at the specified physical location, the existing XMLMap file is used. If an XMLMap file does not exist at the specified physical location, the generated XMLMap file is written to the specified pathname and filename.

prefixattributes=no specifies that the element name is not concatenated to the attribute name when generating each XMLMap COLUMN element, which defines the SAS variable name.

The full code for importing Define-XML documents using SAS XMLMaps can be found in Appendix 1.

Although the process certainly works, maintaining and tweaking the resulting large, monolithic XMLMap is a tedious work. One needs a good understanding of the data sets that are created to be able to merge them.

SLURPING XML WITH GROOVY

Groovy is a dynamic language that runs on the Java Virtual Machine (JVM). PROC GROOVY enables SAS code to execute Groovy code on the JVM. PROC GROOVY was introduced in SAS 9.3.

Groovy takes Java and adds syntax from other languages like Python, Ruby and smalltalk. The rest of the Java universe is available to Groovy!

PROC GROOVY can run Groovy statements that are written as part of your SAS code, and it can run statements that are in files that you specify with PROC GROOVY commands. It can parse Groovy statements into Groovy Class objects, and run these objects or make them available to other PROC GROOVY statements or Java DATA Step Objects. You can also use PROC GROOVY to update your CLASSPATH environment variable with additional CLASSPATH strings or filerefs to JAR files.

Note: Groovy code that is submitted with PROC GROOVY runs as the process owner, and has the same access to resources (file system, network, and so on) as any process owner. Groovy code access to resources can cause problems when SAS code is running inside multiuser servers like the Stored Process Server. To give administrators some control over this functionality, PROC GROOVY runs only if the NOXCMD option is turned off. All SAS servers are shipped with the NOXCMD option turned on. PROC GROOVY will also not run within SAS University Edition.

PROC GROOVY SYNTAX

PROC GROOVY executes Groovy code between SUBMIT and ENDSUBMIT statements. A RUN statement is not necessary because the code will execute following the ENDSUBMIT statement.

```
proc groovy;
  submit;
    println("hello world!")
  endsubmit;

1    proc
1  !      groovy;
2      submit;
```

```
3        println("hello world!")
4      endsubmit;
hello world!
NOTE: The SUBMIT command completed.
```

There are actually 3 ways to run Groovy code in SAS:

EVALUATE	Parse the Groovy statement in the quoted string and call the Run method on the Script
EXECUTE	Read the contents of the file that is specified as either a quoted string path or as a fileref and call the Run method on the Script
SUBMIT	Parse the Groovy statements between the SUBMIT and ENDSUBMIT commands and call the Run method on the Script

Groovy code is run by default. If your script does not have any executable code an error will be thrown. If the LOAD, PARSEONLY, or NORUN option is present on these 3 statements, then the EVALUATE, EXECUTE or SUBMIT statement parses the Groovy statement into a Class object but does not run it. Any classes that are defined by the Groovy code are then available for use by PROC GROOVY statements or by Java DATA Step Objects.

Prior to the EVALUATE, EXECUTE or SUBMIT statement, the CLASSPATH value can be updated to include any necessary JAR files. To clear a CLASSPATH, use the CLEAR option after the ENDSUBMIT (or SUBMIT) statement.

Data can be exchanged between Groovy and SAS in different ways.

Through a CSV file:

```
proc groovy;
  submit;
    def filename = "file.csv"
    new File(filename).withWriter { out ->
      out.writeLine("one,two,three");
      out.writeLine("four,five,six");
    }
    exports.infile=filename;
  endsubmit;
quit;

data dataset;
  infile "&infile" delimiter=',';
  input col1 $ col2 $ col3 $;
run;
```

Through Java Data Step Objects:

```
proc groovy;
  submit parseonly;
    class CreateData {
      Integer i = 0;
      String nextString() {
        ++i;
        return "Value ${i}";
      }
    }
  endsubmit;
quit;
```

```
data Test (drop=n);
  dcl javaobj j("CreateData");
  length str $ 40;
  n = 0;
  do while( n < 30 );
    n = n + 1;
    j.callStringMethod("nextString", str);
    output;
  end;
run;
```

To use the latest stable jar files from Groovy go to http://groovy-lang.org/download.html and download the latest binary zip file. At the writing of this paper, the latest Groovy distribution is 2.4.14.

Depending on your SAS version, you may need to reference the file groovy-all-2.4.14.jar in the proc groovy statement.

```
filename groovy "&Root/groovy/groovy-all-2.4.14.jar";
proc groovy;
  add classpath=groovy;
  ...
run;
```

We use PROC GROOVY in order to parse a Define-XML document. Once parsed we create a CSV file using the CSVWriter class from opencsv (http://opencsv.sourceforge.net/).

opencsv is an easy-to-use CSV (comma-separated values) parser library for Java. We could have used the built-in File.withWriter() method, but the advantage of opencsv is that it supports:

- Arbitrary numbers of values per line.
- Ignoring commas in quoted elements.
- Handling quoted entries with embedded carriage returns (i.e. entries that span multiple lines).
- Configurable separator and quote characters (or use sensible defaults).
- Reading and writing from an array of strings
- Read or write all the entries at once, or use an Iterator-style model

Download the latest opencsv jar file from https://sourceforge.net/projects/opencsv/files/opencsv/ and store it somewhere in your filesystem.

Both Groovy and opencsv jar files can be referenced as follows:

```
filename opencsv "&Root/groovy/opencsv-4.1.jar";
filename groovy "&Root/groovy/groovy-all-2.4.14.jar";
proc groovy;
  add classpath=groovy;
  add classpath=opencsv;
  execute parseonly "&Root/groovy/import_DefineXML_metadata.groovy";
run;
```

It is also possible to dynamically load the opencsv jar file.

```
%let Root=C:/_Data/Presentations/PharmaSUG_2018/Accessing_DefineXML;

filename opencsv "%sysfunc(pathname(work))/opencsv-4.1.jar";
```

```
proc http
   method = "get"
   url = "https://sourceforge.net/projects/opencsv/files/latest/download?source=files"
   out = opencsv;
run;

filename groovy "&Root/groovy/groovy-all-2.4.14.jar";
proc groovy;
  add classpath=groovy;
  add classpath=opencsv;
  ...
run;
```

PROC GROOVY and XML

Groovy provides the **XmlSlurper** class (`groovy.util.XmlSlurper`) to process XML. The XmlSlurper class makes parsing XML very easy. There are other options, but the XmlSlurper is usually considered to be the more efficient in terms of speed and flexibility. XmlSlurper can also be used to transform the XML while parsing it. XmlSlurper allows to parse an XML document and returns an **GPathResult** object. You can use **GPath** expressions to access nodes in the XML tree.

GPath is a path expression language integrated into Groovy which allows parts of nested structured data to be identified. In this sense, it has similar aims and scope as **XPath** does for XML.

As an example, you can specify a path to an object or element of interest:

MetaDataVersion.ItemGroupDef.ItemRef → for XML, yields all the <ItemRef> elements inside <ItemGroupDef> inside <MetaDataVersion>

For XML, you can also specify attributes, e.g.:

ItemDef["@DataType"] → the DataType attribute of all the ItemDef elements

ItemDef.'@DataType' → an alternative way of expressing this

ItemDef.@DataType → an alternative way of expressing this when using XmlSlurper

itemDef.@'def:DisplayFormat' → with namespaces

Let's look at an example to see how easy we can parse XML with the XmlSlurper class.

```
PROC GROOVY;
SUBMIT;

  String xmldocument = '''
    <singles>
      <entry rank="67" year="1954">
        <artist>Nolan Strong and the Diablos</artist>
        <title>The wind</title>
        <writer>Nolan Strong and The Diablos</writer>
        <label>Fortune Records</label>
        <year>1954</year>
      </entry>
      <entry rank="265" year="1962">
        <artist>Nathaniel Mayer</artist>
        <title>Village of love</title>
        <writer>Nathaniel Mayer &amp; Devora Brown</writer>
        <label>Fortune Records</label>
      </entry>
```

```
      <entry rank="938" year="1963">
        <artist>Gino Washington</artist>
        <title>Gino is a coward</title>
        <writer>Ronald Davis</writer>
        <label>Ric Tic Records</label>
      </entry>
    </singles>

'''

def singles = new XmlSlurper().parseText(xmldocument)

for (e in singles.entry) {
  println "[${e.@rank}] ${e.artist} sang \"${e.title}\" in ${e.@year}, " +
          "\n  written by ${e.writer}, for ${e.label}."
  }

singles.entry.each() {
  println "[${it.@rank}] ${it.artist} sang \"${it.title}\" in ${it.@year}, " +
          "\n  written by ${it.writer}, for ${it.label}."
  }

def Fortune = "Fortune Records"
def numberofFortune = singles.entry.findAll{it.label == Fortune}.size()
println "\n$numberofFortune singles were recorded for $Fortune:"
singles.entry.findAll {it.label == Fortune}.each() {
  println "  ${it.artist} - \"${it.title}\" in ${it.@year}"
}

ENDSUBMIT;
```

The above code will get us the following output:

```
[67] Nolan Strong and the Diablos sang "The wind" (1954),
  written by Nolan Strong and The Diablos, for Fortune Records.
[265] Nathaniel Mayer sang "Village of love" in 1962,
  written by Nathaniel Mayer & Devora Brown, for Fortune Records.
[938] Gino Washington sang "Gino is a coward" in 1963,
  written by Ronald Davis, for Ric Tic Records.

[67] Nolan Strong and the Diablos sang "The wind" (1954),
  written by Nolan Strong and The Diablos, for Fortune Records.
[265] Nathaniel Mayer sang "Village of love" (1962),
  written by Nathaniel Mayer & Devora Brown, for Fortune Records.
[938] Gino Washington sang "Gino is a coward" (1963),
  written by Ronald Davis, for Ric Tic Records.

2 singles were recorded for Fortune Records:
  Nolan Strong and the Diablos - "The wind" in 1954
  Nathaniel Mayer - "Village of love" in 1962
```

Notice how easy it is to filter elements that have label = “Fortune Records” with the **`findAll`** `method.`

The full code for importing Define-XML documents using PROC GROOVY can be found in Appendix 2.

One thing to notice about this code is how we use the SAS target data set templates to define SAS LENGTH and INPUT statements for reading the CSV files in the datastep:

```
proc format;
  value $typ 'char'='$' num=' ';
run;

proc sql noprint;
  select catx(' ', name, cats(put(type, $typ.)))
    into: tableinput separated by ' '
  from dictionary.columns
  where (upcase(libname)='TMPLTS' and upcase(memname)='STUDYTABLEMETADATA')
  order by varnum
  ;
  select catx(' ', name, cats(put(type, $typ.)), length)
    into: tablelength separated by ' '
  from dictionary.columns
  where (upcase(libname)='TMPLTS' and upcase(memname)='STUDYTABLEMETADATA')
  order by varnum
  ;
quit;
```

This creates 2 macro variables:

```
TABLEINPUT=sasref $ table $ label $ order repeating $ isreferencedata $ domain $
domaindescription $ class $ xmlpath $ xmltitle $ structure $ purpose $ keys $ state $
date $ comment $ studyversion $ standard $ standardversion $

TABLELENGTH=sasref $ 8 table $ 32 label $ 200 order 8 repeating $ 3 isreferencedata $ 3
domain $ 32 domaindescription $ 256 class $ 40 xmlpath $ 200 xmltitle $ 200 structure $
200 purpose $ 10 keys $ 200 state $ 20 date $ 20 comment $ 1000 studyversion $ 128
standard $ 20 standardversion $ 20
```

We use these macro variables when reading the CSV file:

```
data work.table_metadata;
  infile "&csvOutputFolder/tablemetadata.csv"
    delimiter='09'x missover dsd lrecl=32767 firstobs=2 ;
  length &tablelength;
  input &tableinput;
run;

 data work.table_metadata;
  set work.table_metadata;
  comment=tranwrd(comment, "\n", '0A'x);
run;
```

Several elements in the Define-XML file may have embedded carriage returns or linefeeds. This is a problem, since we are creating CSV files to be consumed in a datastep. This is solved by converting linefeeds to the string “\n” in the Groovy code, and then converting these “\n” strings back to linefeeds after reading the CSV files in the datastep. This typically may occur in MethodDef, FormalExpression, def:CommentDef, arm:AnalysisResult/arm:Documentation or arm:AnalysisResult/arm:ProgrammingCode elements in a Define-XML document.

In PROC GROOVY:

```
commentDef.Description.TranslatedText.text().trim().replace("\n", "\\n")
```

In the datastep:

```
comment=tranwrd(comment, "\n", '0A'x);
```

TRANSFORMING XML WITH XSLT

What is XSLT

XSLT stands for Extensible Style sheet Language Transformations [13], one of the most complicated – and most useful – parts of XML. While XML itself is intended to define the structure of a document, it does not contain any information on how it is to be displayed. In order to do this we need a language, XSLT, to describe the format of a document, ready for use in a display application (computer screen, cell phone screen, paper).

XSLT is actually a family of transformation languages which allows one to describe how files encoded in the XML standard are to be formatted or transformed.

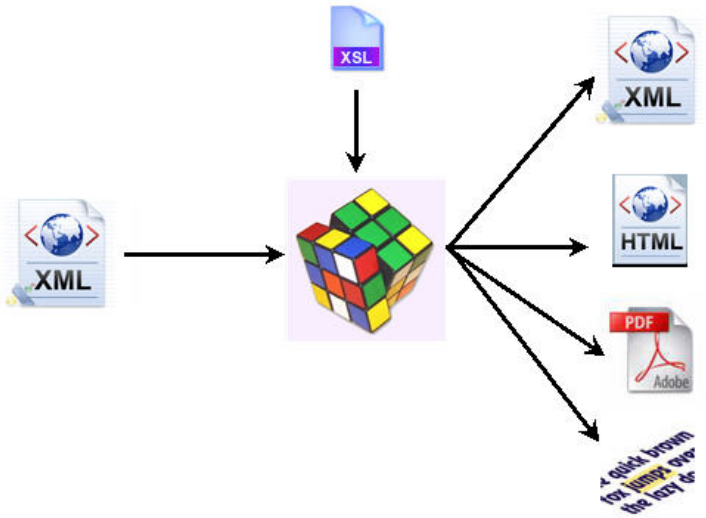
The following three languages can be distinguished:

- XSL Transformations (XSLT): an XML language for transforming XML documents.
- XSL Formatting Objects (XSL-FO): an XML language for specifying the visual formatting of an XML document
- The XML Path Language (XPath): a non-XML language used by XSLT, and also available for use in non-XSLT contexts, for addressing the parts of an XML document.

An XSL stylesheet can be used to transform a Define-XML document to HTML in such a way that it can be viewed in a browser.

XSLT lets you convert XML documents into other XML documents, into HTML documents, or into any other text based documents (like a SAS program), or even a PDF file.

XSLT is a language "*for transforming the structure and content of an XML document*" [14]. In [14]. Michael Kay compares XSLT with the famous Rubik's cube. XSLT can convert attributes to elements, or elements to attributes.



An XSLT stylesheet has a collection of template rules. Each template has a pattern that identifies the source tree nodes to which the pattern applies and a template that is added to the result tree when the XSLT processor applies that template rule to a matched node.

Flattening Define-XML

Using an XSL transformation (see Appendix 3) we can “flatten” the Define-XML. As an example take the following Define-XML fragment that describes the table metadata for the DM domain.

```
<ItemGroupDef OID="IG.DM" Domain="DM" Name="DM" Repeating="No" IsReferenceData="No"
  SASDatasetName="DM" Purpose="Tabulation" def:Structure="One record per subject"
  def:Class="SPECIAL PURPOSE" def:CommentOID="COM.DOMAIN.DM"
  def:ArchiveLocationID="LF.DM">
  <Description>
    <TranslatedText xml:lang="en">Demographics</TranslatedText>
  </Description>
...
</ItemGroupDef>


<def:CommentDef OID="COM.DOMAIN.DM">
  <Description>
    <TranslatedText xml:lang="en">See Reviewer's Guide, Section 2.1 Demographics
</TranslatedText>
  </Description>
</def:CommentDef>
```

After transforming with an XSL stylesheet this XML gets transformed to:

```
<?xml version="1.0" encoding="UTF-8"?>
<LIBRARY>
  <ItemGroupDef>
    <table>DM</table>
    <label>Demographics</label>
    <order>6</order>
    <repeating>No</repeating>
    <isreferencedata>No</isreferencedata>
    <domain>DM</domain>
    <domaindescription/>
    <class>SPECIAL PURPOSE</class>
    <xmlpath>dm.xpt</xmlpath>
    <xmltitle>dm.xpt</xmltitle>
    <structure>One record per subject</structure>
    <purpose>Tabulation</purpose>
    <keys>STUDYID USUBJID</keys>
    <date>2013-03-03</date>
    <comment>See Reviewer's Guide, Section 2.1 Demographics</comment>
    <studyversion>MDV.CDISC01.SDTMIG.3.1.2.SDTM.1.2</studyversion>
    <standard>SDTM-IG</standard>
    <standardversion>3.1.2</standardversion>
  </ItemGroupDef>
```

Similar for the following Define-XML fragment that describe the column metadata for the USUBJID variable:

```
<ItemRef ItemOID="IT.USUBJID" OrderNumber="3" Mandatory="Yes" KeySequence="2"
    MethodOID="MT.USUBJID"/>

<ItemDef OID="IT.USUBJID" Name="USUBJID" DataType="text" Length="14"
  SASFieldName="USUBJID">
  <Description>
    <TranslatedText xml:lang="en">Unique Subject Identifier</TranslatedText>
  </Description>
```

```
        <def:Origin Type="Derived"/>
      </ItemDef>

      <MethodDef OID="MT.USUBJID" Name="Algorithm to derive USUBJID" Type="Computation">
        <Description>
          <TranslatedText xml:lang="en">Concatenation of STUDYID and SUBJID</TranslatedText>
        </Description>
        <FormalExpression Context="SAS 9.0 or later, as part of a data step assignment or proc
sql select and update statements.">
          catx(".",STUDYID,SUBJID)
        </FormalExpression>
      </MethodDef>
```

After the transformation it looks like this:

```
   <ItemRefItemDef>
      <table>DM</table>
      <column>USUBJID</column>
      <label>Unique Subject Identifier</label>
      <order>3</order>
      <length>14</length>
      <displayformat/>
      <significantdigits/>
      <xmldatatype>text</xmldatatype>
      <xmlcodelist/>
      <origin>Derived</origin>
      <origindescription/>
      <role/>
      <algorithm>Concatenation of STUDYID and SUBJID</algorithm>
      <algorithmtype>Computation</algorithmtype>
      <formalexpression>
          catx(".",STUDYID,SUBJID)
        </formalexpression>
      <formalexpressioncontext>SAS 9.0 or later, as part of a data step assignment or proc sql
select and update statements.</formalexpressioncontext>
      <comment/>
      <studyversion>MDV.CDISC01.SDTMIG.3.1.2.SDTM.1.2</studyversion>
      <standard>SDTM-IG</standard>
      <standardversion>3.1.2</standardversion>
   </ItemRefItemDef>
```

We could read this flattened XML file with a simple XMLv2 LIBNAME statement:

```
libname out "&Root/xslt/out";
filename flatxml "&Root/xslt/out/DefineXML_flat.xml";
libname define xmlv2 xmlfileref=flatxml;

proc copy in=define out=out;
run;
```

The problem is that we still have no control over the datatype that the SAS XMLv2 LIBNAME engine creates. When the XML instance only has values like “5.1” and “4.2” for the DisplayFormat attribute, the SAS XMLv2 LIBNAME engine will create a numeric variable, even though we know that it should be a character variable.
We can solve this by using the metadata from the templates that describe the target data sets. These templates contain the desired column attributes: length, datatype and label. With a simple macro we can automatically generate an XMLMap from the metadata templates that we then can use with the SAS XMLv2 LIBNAME engine:

```
<?xml version="1.0" encoding="UTF-8"?>
<SXLEMAP name="define" version="2.1">
  <TABLE name="column_metadata">
    <TABLE-PATH syntax="XPath">/LIBRARY/ItemRefItemDef</TABLE-PATH>
    <COLUMN name="length">
      <PATH syntax="XPath">/LIBRARY/ItemRefItemDef/length</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>numeric</DATATYPE>
      <DESCRIPTION>Column Length</DESCRIPTION>
      <LENGTH>8</LENGTH>
    </COLUMN>
    <COLUMN name="displayformat">
      <PATH syntax="XPath">/LIBRARY/ItemRefItemDef/displayformat</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>character</DATATYPE>
      <DESCRIPTION>Display Format</DESCRIPTION>
      <LENGTH>200</LENGTH>
    </COLUMN>
```

This map is gets created with the following macro call:

```
%CreateXMLMap(
  tablepath=LIBRARY/ItemRefItemDef,
  tablename=column_metadata,
  library=tmplts,
  metadatadataset=studycolumnmetadata,
  mapfileref=colmap
);
```

The complete macro definition can be found in Appendix 3.

After leaving the heavy XML work to XSL, and automatically generating and XMLMap from metadata, importing Define-XML table metadata becomes really simple now. Here is the complete SAS code:

```
%let Root=C:/_Data/Presentations/PharmaSUG_2018/Accessing_DefineXML;

%include "&Root/xslt/CreateXMLMap.sas";

libname tmplts "&Root/templates";
libname metadata "&Root/xslt/metadata";

filename xml "&Root/xml/define2-0-0-example-sdtm.xml";
filename xsl "&Root/xslt/stylesheets/DefineXML.xsl";
filename flatxml "&Root/xslt/out/DefineXML_flat.xml";

proc xsl in=xml xsl=xsl out=flatxml;
run;

filename tabmap "&Root/xslt/out/definexml_tables.map";

%CreateXMLMap(
  library=tmplts,
  metadatadataset=studytablemetadata,
  tablepath=LIBRARY/ItemGroupDef,
  tablename=table_metadata,
  mapfileref=tabmap
);
```

```
libname define xmlv2 xmlfileref=flatxml xmlmap=tabmap;

data metadata.table_metadata;
  set define.table_metadata;
  sasref = "SRCDATA";
  state = "Final";
run;
```

The methodology of using an XSLT stylesheet to flatten the Define-XML document, before applying the SAS XML engine is exactly what is used in the SAS Clinical Standards Toolkit.

SAS CLINICAL STANDARDS TOOLKIT

Introduction

The SAS Clinical Standards Toolkit focuses on standards defined by the Clinical Data Interchange Standards Consortium (CDISC). CDISC is a global, open, multidisciplinary, nonprofit organization that has established standards to support the acquisition, exchange, submission, and archival of clinical research data and metadata. The CDISC mission is to develop and support global, platform-independent data standards that enable information-system interoperability, which, in turn, improves medical research and related areas of health care. The SAS Clinical Standards Toolkit is not limited to supporting CDISC standards. The SAS Clinical Standards Toolkit framework is designed to support the specification and use of any user-defined standard.

The SAS Clinical Standards Toolit is a separately orderable component that is available at no additional charge to currently licensed SAS customers. Contact your SAS Account Representative to request the toolkit to be added to your Base SAS order.

The SAS Clinical Standards Toolkit 1.7 (see reference [15]) includes support for the following CDISC standards:

- SDTM 1.3.1, 3.1.2, 3.1.3 and 3.2
- An initial implementation of the CDISC SEND 3.0 standard, including definition of all domains and columns, and the creation of a tumor data set.
- ADaM 2.1 (ADSL, Basic Data Structure, ADAE) and Analysis Results Metadata templates, as well as new validation checks in support of ADAE and ADTTE.
- CRT-DDS 1.0 (define.xml), including define.pdf and Value Level metadata support.
- Dataset-XML 1.0:
    - Creating Dataset-XML files from SAS data sets
    - Creating SAS data sets from Dataset-XML files
    - Validating Dataset-XML files against an XML schema
    - Comparing original SAS data sets with SAS data sets created from Dataset-XML files
- Define-XML 2.0:
    - A complete definition of the metadata model for CDISC Define-XML 2.0 (including Analysis Results Metadata after applying a hot fix)
    - Creation of a complete Define-XML 2.0 file based on study metadata, with study metadata examples from SDTM 3.2 and ADaM 2.1
    - Validation of a Define-XML 2.0 file against the XML schema definition, as published by CDISC

  - Import of a Define-XML 2.0 file into the SAS representation of the Define-XML 2.0 metadata model
  - support of creating an initial version of the SAS source metadata data sets (source_study, source_tables, source_columns, source_codelists, source_values, source_documents, and source_analysisresults) that serve as input for creating a Define-XML v2.0 file

- ODM 1.3.0 and ODM 1.3.1, including support for the extraction of ODM Clinical data and ODM Reference data into SAS data sets.
- The implementation of CT 1.0.0, a tool to support the import of NCI CDISC Controlled Terminology in the ODM XML format into SAS data sets and SAS format catalogs.
- CDISC Controlled Terminology packages that includes terminology sets as posted to the NCI FTP site.
- Furthermore, a set of macro tools to validate the SAS Clinical Standards Toolkit metadata itself ("Internal validation").
- Metadata Management tools to add, update and delete Toolkit metadata

SAS Clinical Standards Toolkit 1.7 Second Edition (also known as SAS Clinical Standards Toolkit 1.7.1) is a hot fix that added support for the CDISC Analysis Results Metadata specification version 1.0 for Define-XML version 2.0, and also adds the December 2015 snapshot of CDISC NCI controlled terminology for ADaM. SAS Clinical Standards Toolkit 1.7.1 is supported with SAS 9.4 (TS1M3) or later on Windows x64 and Linux x64.

The latest hot fix for SAS Clinical Standards Toolkit was released in December 2017 and contains extra Define-XML source metadata and bug fixes (see: http://ftp.sas.com/techsup/download/hotfix/HF2/Z50.html#Z50002).

Each SAS Clinical Standards Toolkit standard provides a SAS representation of the published source guidelines or source specification. The SAS representation is designed to serve as a model or template of the source specification. Two key design requirements shaped the implementation of the SAS Clinical Standards Toolkit standards.

1) Each supported standard is represented in one or more SAS files. This facilitates these points:
   - It provides SAS users with an implementation of data models and standards that are based on SAS.
   - It enables you to use SAS routines to assess how well any user-defined set of data and metadata conforms to the standard.
   - It enables you to use SAS code to read and derive files in other formats (for example, XML).

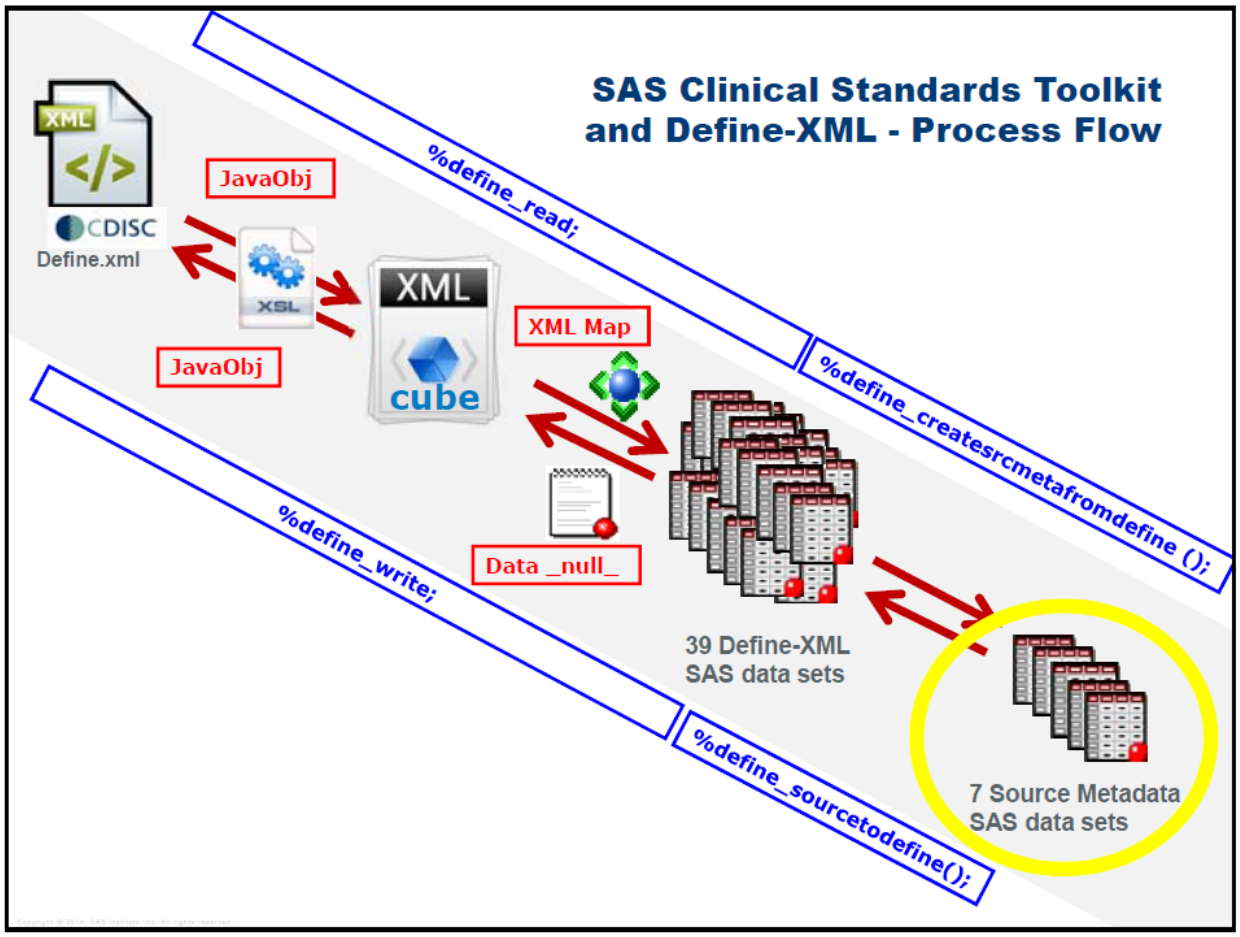
   Each SAS Clinical Standards Toolkit standard is an optimized reference standard from a SAS perspective.
2) You are able to define your own customized standards, or you are able to modify existing SAS standards.

Since a Define-XML file does not have a 2-dimensional data structure, it is not a trivial task to translate this hierarchical file to a number of SAS data set with rows and columns. SAS has defined a relational data model that represents a Define-XML file.

The highly structured nature of CDISC Define-XML data requires that any mapping to a relational format include a large number of data sets, with foreign key relationships to help preserve the intended non-relational object structure. In the SAS Clinical Standards Toolkit, foreign key relationships are enforced when validating the CDISC Define-XML data sets.

The figure below shows how SAS Clinical Toolkit converts between a Define-XML document, the SAS representation of Define-XML and the source metadata data sets.

SAS Clinical Standards Toolkit and Define-XML – Process Flow



Source Metadata SAS data sets in SAS Clinical Standards Toolkit

For Define-XML 2.0 the following source metadata SAS data sets are defined in SAS Clinical Standards Toolkit:

- source_study:
  - Metadata about the study, such as study name, study description and protocol name.
- source_tables:
  - Domain metadata, such as name, domain, description (label), class, structure, purpose, keys, data location, comments and documentation reference.
- source_columns
  - Column metadata, such as name, description (label), order number, datatype, length, codelist, origin, significant digits, display format, derivation (algorithm), comments and documentation reference.
- source_values

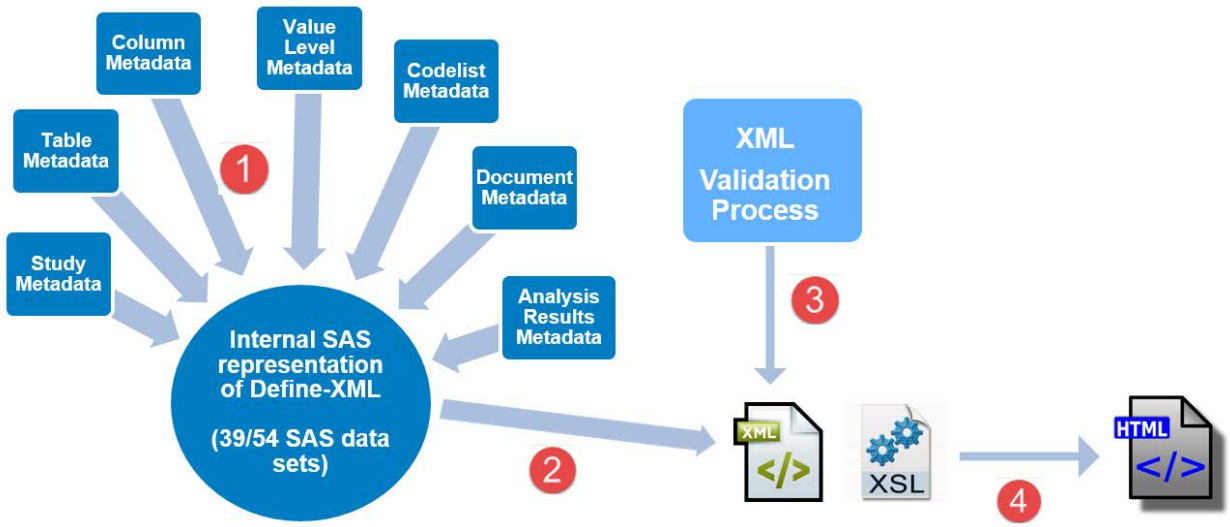
  - Value level metadata, where a condition is defined in the WHERECLAUSE column. Example WHERECLAUSE values are:
    - (LBTESTCD EQ "BILI") AND (LBCAT EQ "CHEMISTRY") AND (LBSPEC EQ "BLOOD")
    - VSTESTCD EQ “HEIGHT”
    - PARAMCD IN ("ACITM01","ACITM02","ACITM03")
    - PARAMCD NOTIN ("ACTOT")

    The column which the value level metadata is attached to, is defined by the TABLE and COLUMN columns. Apart from the WHERECLAUSE column, this data set contains the same kind of metadata as the source_columns data set.

- source_codelists:
  - Metadata related to Controlled Terminology, such as name, description, datatype, SAS formatname, valid values, decodes, rank, order number, reference to NCI code, external terminologies
- source_documents:
  - Metadata related to referenced documents, such as annotated CRF, reviewer guides or other supplemental documents. Records in this data set can be linked to source_tables, source_columns, source_values, or source_analysisresults data sets by the combination of the TABLE, COLUMN, WHERECLAUSE, DISPLAYIDENTIFIER and RESULTIDENTIFIER columns. Page numbers and named destinations in PDF files can be defined in this data set as well. Documents are attached to comments, methods or origins based on the value of the DOCTYPE column.
- Source_analysisresults:
  - Metadata related to analysis displays and results: display identifier, display name, display description, result identifier, result description, analysis purpose and reason, parameter column, analysis variables, analysis datasets, selection criteria (WhereClause), Selection criteria for the records subject to analysis, result programming code and context, result documentation.

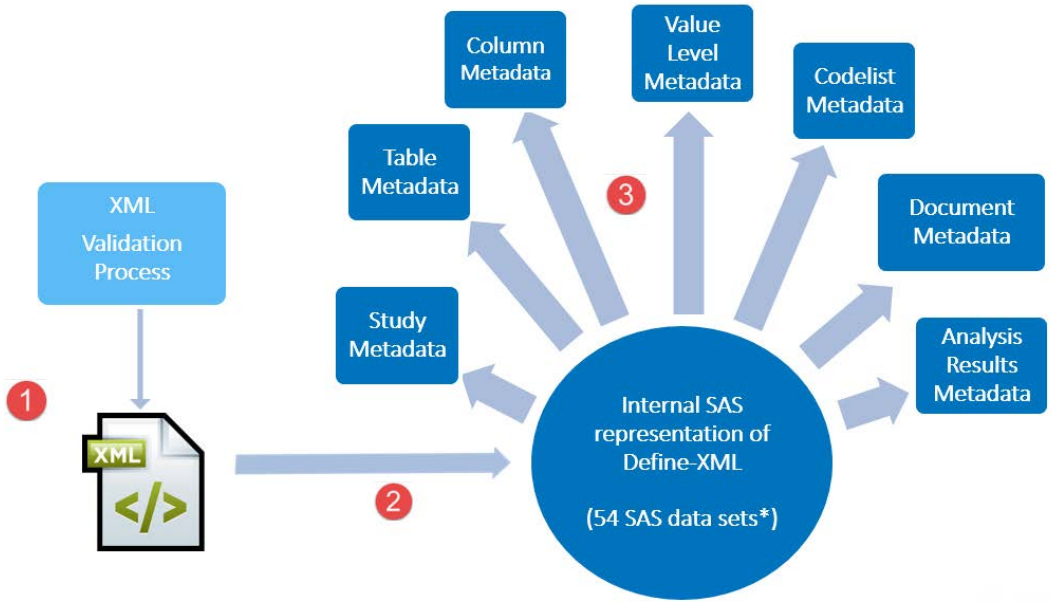
Below we see the complete process for creating a complete Define-XML version 2 in the SAS Clinical Standards Toolkit [16].

The SAS macro process to create a valid Define-XML 2.0 document from study source metadata



The SAS Clinical Standards Toolkit can also create the same source metadata data sets from a Define-XML 2.0 document.

The SAS macro process to create study source metadata from a Define-XML 2.0 document



There are three key macros that are provided with the SAS Clinical Standards Toolkit that support creation of source metadata from a CDISC Define-XML 2.0 document. The three macros are listed in the order in which they are executed:

1. The **cstutilxmlvalidate** macro validates that the XML file is syntactically correct according the XML schema that is associated with the CDISC Define-XML 2.0 standard.
2. The **define_read** macro creates the SAS representation of the CDISC Define-XML 2.0 files from a Define-XML 2.0 document.
3. The **define_createsrcmetafromdefine** macro creates the source metadata data sets from the SAS representation of the Define-XML 2.0.

These macros are called by driver programs that are responsible for properly setting up each SAS Clinical Standards Toolkit process to perform a specific SAS Clinical Standards Toolkit task. Several sample driver programs are provided with the SAS Clinical Standards Toolkit CDISC Define-XML 2.0 standard related to the import of the define.xml file.

Here is the purpose of each of these driver programs:

- The **create_sasdefine_fromxml.sas** driver program sets up the required metadata and SASReferences data set for the sample study. It runs the **define_read** and **cstutilxmlvalidate** macros. It creates the SAS representation of the CDISC Define-XML 2.0 data sets from the CDISC Define-XML 2.0 document. This driver program also validates the XML syntax for the Define-XML document.
- The **create_sourcemetadata_fromsasdefine** driver program creates the study source metadata data sets from the SAS representation of the CDISC Define-XML 2.0 data sets. It runs the **define_createsrcmetafromdefine** macro.

These driver programs are examples that are provided with the SAS Clinical Standards Toolkit. You can use these driver programs or create your own. The names of these driver programs are not important.

However, the content is important and demonstrates how the various SAS Clinical Standards Toolkit framework macros are used to generate the required metadata files.

The **define_createsrcmetafromdefine** macro creates SDTM, SEND or ADaM study source metadata data sets.

The parameters in table below must be set before submitting the **define_createsrcmetafromdefine** macro.

define_createsrcmetafromdefine macro parameters

Parameter	Required	Description
_cstDefineDataLib	Yes	The library where the SAS representation of a Define-XML V2.0.0 file is located.
_cstTrgStandard	Yes	The name of the study standard for which the macro creates source study metadata. Example: CDISC-SDTM
_cstTrgStandardVersion	Yes	The version of the study standard for which the macro creates source study metadata. Example: 3.1.2
_cstTrgStudyDS	Yes	The data set that contains the metadata for the studies to include in the Define-XML file.
_cstTrgTableDS	Yes	The data set that contains the metadata for the domains to include in the Define-XML file.
_cstTrgColumnDS	Yes	The data set that contains the metadata for the Domain columns to include in the Define-XML file.
_cstTrgCodeListDS	Yes	The data set that contains the metadata for the CodeLists to include in the Define-XML file.
_cstTrgValueDS	Yes	The data set that contains the metadata for the Value Level columns to include in the Define-XML file.
_cstTrgDocumentDS	Yes	The data set that contains the metadata for document references to include in the Define-XML file.
_cstTrgAnalysisResultDS	No.	The data set that contains the metadata for analysis results to include in the Define-XML file.
_cstLang	No	The ODM TranslatedText/@lang attribute.
_cstUseRefLib	Yes	Use reference and class metadata to impute missing information in target metadata (Y/N). Default: N.
_cstRefTableDS	Conditional	The data set that contains table reference metadata for the target data standard. Required in case _cstUseRefLib=Y.
_cstRefColumnDS	Conditional	The data set that contains column reference metadata for the target data standard. Required in case _cstUseRefLib=Y.
_cstClassTableDS	Conditional	The data set that contains table class metadata for the target data standard. Required in case _cstUseRefLib=Y.
_cstClassColumnDS	Conditional	The data set that contains column class reference metadata for the target data standard. Required in case _cstUseRefLib=Y.
_cstReturn	Yes	The macro variable that contains the return value as set by this macro. Default: _cst_rc
_cstReturnMsg	Yes	The macro variable that contains the return message as set by this macro. Default: _cst_rcmsg

Here is an example of a call to the define_createsrcmetafromdefine macro:

```
%define_createsrcmetafromdefine(
  _cstDefineDataLib=srcdata,
  _cstTrgStandard=CDISC-SDTM,
  _cstTrgStandardVersion=3.1.2,
  _cstTrgStudyDS=trgmeta.source_study,
  _cstTrgTableDS=trgmeta.source_tables,
  _cstTrgColumnDS=trgmeta.source_columns,
  _cstTrgCodeListDS=trgmeta.source_codelists,
  _cstTrgValueDS=trgmeta.source_values,
  _cstTrgDocumentDS=trgmeta.source_documents,
  _cstTrgAnalysisResultDS=trgmeta.source_analysisresults,
  _cstLang=en,
  _cstUseRefLib=Y,
  _cstRefTableDS=refmeta.reference_tables,
  _cstRefColumnDS=refmeta.reference_columns,
  _cstClassTableDS=refmeta.class_tables,
  _cstClassColumnDS=refmeta.class_columns,
  _cstReturn=_cst_rc,
  _cstReturnMsg=_cst_rcmsg
);
```

In the example, the define_createsrcmetafromdefine macro extracts data from the tables in the srcdata library that represent the SAS interpretation of the CDISC Define-XML 2.0 model and creates SDTM source metadata tables in the sampdata library. The macro will use standard class and reference metadata to impute missing metadata.

The full code to create source metadata data sets from a Define-XML 2.0 document with SAS Clinical Standards Toolkit can be found in Appendix 4.

CONCLUSION

Until now, most papers about Define-XML were about the process of creating a Define-XML document. This paper demonstrated various ways of extracting metadata from a Define-XML document. We used SAS XMLMaps, PROC GROOVY, PROC XSL and the SAS Clinical Standards Toolkit. We only showed examples for extracting table and column metadata, but the same methodologies can be used to extract the complete metadata from a Define-XML document.

Creating XMLMaps for complex XML documents like Define-XML can be tedious. PROC GROOVY and XSLT provide alternatives.

Importing Define-XML metadata with SAS Clinical Standards Toolkit is a matter of setting up some macro variables, defining libnames and filenames, and running a few macros to get the full metadata from a Define-XML v2 document.

REFERENCES

1. Lex Jansen (2012). Using the SAS® Clinical Standards Toolkit for define.xml creation. Proceedings of the Pharmaceutical Industry SAS® Users Group (PharmaSUG 2012, San Francisco, CA) (http://www.lexjansen.com/pharmasug/2012/HW/PharmaSUG-2012-HW02-SAS.pdf)
2. Lex Jansen (2013). Define-XML v2 – What's New. Proceedings of the 9th Pharmaceutical Users Software Exchange (PhUSE 2013, Brussels, Belgium) (http://www.lexjansen.com/phuse/2013/cd/CD05.pdf)
3. CDISC Define-XML Specification, Version 2.0, March 5, 2013 (https://www.cdisc.org/standards/data-exchange/define-xml)
4. Case Report Tabulation Data Definition Specification (define.xml), Version 1.0, February 9, 2005 (https://www.cdisc.org/standards/data-exchange/define-xml)

5. FDA Study Data Standards Catalog (Version 4.5; Effective 2016-08-31). (http://www.fda.gov/ForIndustry/DataStandards/StudyDataStandards/default.htm)
6. U.S. Department of Health and Human Services Food and Drug Administration Center for Drug Evaluation and Research (CDER), Center for Biologics Evaluation and Research (CBER). Study Data Technical Conformance Guide, Version 4.0, October 2017 (http://www.fda.gov/ForIndustry/DataStandards/StudyDataStandards/default.htm)
7. Department Of Health And Human Services Food and Drug Administration [Docket No. FDA–2014–N–1840] Electronic Study Data Submission; Data Standards; Support End Date for Case Report Tabulation Data Definition Specification Version 1.0, March 17, 2016 (http://www.gpo.gov/fdsys/pkg/FR-2016-03-17/pdf/2016-05958.pdf)
8. Lex Jansen (2008). Using the SAS XML Mapper and ODS PDF to create a PDF representation of the define.xml (that can be printed). Proceedings of the 4th Pharmaceutical Users Software Exchange (PhUSE 2008, Manchester, United Kingdom) (http://www.lexjansen.com/phuse/2008/cd/CD04.pdf)
9. Lex Jansen (2010). Accessing the metadata from the define.xml using XSLT transformations. Proceedings of the Pharmaceutical Industry SAS® Users Group (PharmaSUG 2008, Orlando, FL) (http://www.lexjansen.com/pharmasug/2010/CD/CD14.pdf)
10. Chris Schacherer (2014). SAS® XML Programming Techniques. Proceedings of the SAS Global Forum 2014. Cary, NC: SAS Institute, Inc. (http://support.sas.com/resources/papers/proceedings14/1282-2014.pdf)
11. Thomas W. Cox (2012). Advanced XML Processing with SAS® 9.3. Proceedings of the SAS Global Forum 2012. Cary, NC: SAS Institute, Inc. (http://support.sas.com/resources/papers/proceedings12/220-2012.pdf)
12. SAS Institute Inc. (2013). SAS® 9.4 XML LIBNAME Engine: User's Guide. Cary, NC: SAS Institute, Inc.
13. Dough Tidwell, 2001, XSLT, *Mastering XML Transformations*. (O'Reilly and Associates)
14. Michael Kay, 2008, XSLT 2.0 and XPath 2.0, 4th Edition, *Programmer's Reference* (Wrox)
15. SAS Institute Inc. 2016. SAS® Clinical Standards Toolkit 1.7.1. Cary, NC: SAS Institute Inc. (http://support.sas.com/rnd/base/cdisc/cst/index.html)
16. Lex Jansen (2017). Creating Define-XML version 2 including Analysis Results Metadata with the SAS® Clinical Standards Toolkit. Proceedings of the Pharmaceutical Industry SAS® Users Group (PharmaSUG 2017, Baltimore, MD) (http://www.lexjansen.com/pharmasug/2017/SS/PharmaSUG-2017-SS08.pdf)

CONTACT INFORMATION

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

Lex Jansen
SAS Institute Inc.
Email: lex.jansen@sas.com

APPENDIX 1: IMPORTING DEFINE-XML USING SAS XMLMAPS

import_DefineXML_metadata_XMLMap.sas

```
%let Root=C:/_Data/Presentations/PharmaSUG_2018/Accessing_DefineXML;

libname tmplts "&Root/templates";
libname metadata "&Root/xmlmap/metadata";
libname out "&Root/xmlmap/out";

filename define "&Root/xml/define2-0-0-example-sdtm.xml";
filename map "&Root/xmlmap/definexml_auto.map";
libname define xmlv2 automap=reuse xmlmap=map prefixattributes=no;

proc copy in=define out=out;
run;

*******************************************************************************;
*** Table metadata                                                          ***;
*******************************************************************************;
proc sql;
  create table work.ItemGroupDefKeys
    as select
    igd.OID as ItemGroupDefOID,
    igd.Name as Table,
    ir1.KeySequence,
    id.Name as Column

    from out.Study std
      left join out.MetaDataVersion mdv
    on mdv.Study_ORDINAL = std.Study_ORDINAL
      left join out.ItemGroupDef igd
    on igd.MetaDataVersion_ORDINAL = mdv.MetaDataVersion_ORDINAL
      left join out.ItemRef1 ir1
    on ir1.ItemGroupDef_ORDINAL = igd.ItemGroupDef_ORDINAL
      left join out.ItemDef id
    on (id.MetaDataVersion_ORDINAL = mdv.MetaDataVersion_ORDINAL) and
       (ir1.ItemOID = id.OID)
    where not missing(keysequence)
    order by table, keysequence
    ;
quit;

data work.ItemGroupDefKeys;
  length keys $200;
  retain keys;
  set work.ItemGroupDefKeys;
  by table keysequence;
  if first.table then keys=Column;
                 else keys=catx(' ', keys, Column);
  if last.table;
run;

data work.ItemGroupDef;
 set out.ItemGroupDef;
```

```
 order=_n_;
run;

proc sql;
  create table work.table_metadata
    as select
    igd.Name as Table,
    tt.TranslatedText as Label,
    igd.Order,
    igd.Repeating,
    igd.isReferenceData,
    igd.Domain,
    al.Name as DomainDescription,
    igd.Class,
    lf.href as xmlpath,
    lf.title as xmltitle,
    igd.Structure,
    igd.Purpose,
    igdk.Keys,
    put(odm.CreationDateTime, E8601DT.) as Date,
    tt5.TranslatedText5 as Comment,
    mdv.OID as StudyVersion,
    mdv.StandardName as Standard,
    mdv.StandardVersion

    from out.ODM odm
      left join out.Study std
    on std.ODM_ORDINAL = odm.ODM_ORDINAL
      left join out.MetaDataVersion mdv
    on mdv.Study_ORDINAL = std.Study_ORDINAL
      left join work.ItemGroupDef igd
    on igd.MetaDataVersion_ORDINAL = mdv.MetaDataVersion_ORDINAL
      left join out.Description des
    on des.ItemGroupDef_ORDINAL = igd.ItemGroupDef_ORDINAL
      left join out.TranslatedText tt
    on tt.Description_ORDINAL = des.Description_ORDINAL
      left join out.Leaf lf
    on lf.ItemGroupDef_ORDINAL = igd.ItemGroupDef_ORDINAL
      left join out.Alias al
    on (al.ItemGroupDef_ORDINAL = igd.ItemGroupDef_ORDINAL) and
       (al.Context = "DomainDescription")
      left join out.CommentDef comd
    on (comd.MetaDataVersion_ORDINAL = mdv.MetaDataVersion_ORDINAL) and
       (igd.CommentOID = comd.OID)
      left join out.Description4 des4
    on des4.CommentDef_ORDINAL = comd.CommentDef_ORDINAL
      left join out.TranslatedText5 tt5
    on tt5.Description4_ORDINAL = des4.Description4_ORDINAL
      left join work.ItemGroupDefKeys igdk
    on igdk.ItemGroupDefOID = igd.OID
  ;
quit;

data metadata.table_metadata;
  set tmplts.studytablemetadata work.table_metadata;
  sasref = "SRCDATA";
  state = "Final";
```

```
run;

*************************************************************************;
*** Column metadata                                                   ***;
*************************************************************************;
proc sql;
  create table work.column_metadata
    as select
    igd.Name as Table,
    id.Name as Column,
    tt1.TranslatedText1 as Label,
    ir1.OrderNumber as Order,
    case when id.DataType in ("integer","float")
      then 'N'
      else 'C'
    end as type,
    id.Length,
    id.DisplayFormat,
    id.SignificantDigits,
    id.DataType as xmldatatype,
    clr.CodeListOID as xmlcodelist,
    itor.type as origin,
    tt2.TranslatedText2 as origindescription,
    ir1.Role,
    tt4.TranslatedText4 as algorithm,
    metd.Type as algorithmtype,
    formex.Context as formalexpressioncontext,
    formex.FormalExpression as formalexpression,
    tt5.TranslatedText5 as Comment,
    mdv.OID as StudyVersion,
    mdv.StandardName as Standard,
    mdv.StandardVersion

    from out.ODM odm
      left join out.Study std
    on std.ODM_ORDINAL = odm.ODM_ORDINAL
      left join out.MetaDataVersion mdv
    on mdv.Study_ORDINAL = std.Study_ORDINAL
      left join work.ItemGroupDef igd
    on igd.MetaDataVersion_ORDINAL = mdv.MetaDataVersion_ORDINAL
      left join out.ItemRef1 ir1
    on ir1.ItemGroupDef_ORDINAL = igd.ItemGroupDef_ORDINAL
      left join out.ItemDef id
    on (id.MetaDataVersion_ORDINAL = mdv.MetaDataVersion_ORDINAL) and
       (ir1.ItemOID = id.OID)
      left join out.Description1 des1
    on des1.ItemDef_ORDINAL = id.ItemDef_ORDINAL
      left join out.TranslatedText1 tt1
    on tt1.Description1_ORDINAL = des1.Description1_ORDINAL
       left join out.CodeListRef clr
    on clr.ItemDef_ORDINAL = id.ItemDef_ORDINAL
       left join out.origin itor
    on itor.ItemDef_ORDINAL = id.ItemDef_ORDINAL
      left join out.Description2 des2
    on des2.Origin_ORDINAL = itor.Origin_ORDINAL
      left join out.TranslatedText2 tt2
    on tt2.Description2_ORDINAL = des2.Description2_ORDINAL
```

```
      left join out.methoddef metd
    on (metd.MetaDataVersion_ORDINAL = mdv.MetaDataVersion_ORDINAL) and (ir1.MethodOID =
metd.OID)
      left join out.Description3 des3
    on des3.MethodDef_ORDINAL = metd.MethodDef_ORDINAL
      left join out.TranslatedText4 tt4
    on tt4.Description3_ORDINAL = des3.Description3_ORDINAL
      left join out.FormalExpression formex
    on formex.MethodDef_ORDINAL = metd.MethodDef_ORDINAL
      left join out.CommentDef comd
    on (comd.MetaDataVersion_ORDINAL = mdv.MetaDataVersion_ORDINAL) and
       (id.CommentOID = comd.OID)
      left join out.Description4 des4
    on des4.CommentDef_ORDINAL = comd.CommentDef_ORDINAL
      left join out.TranslatedText5 tt5
    on tt5.Description4_ORDINAL = des4.Description4_ORDINAL
  ;
quit;

data metadata.column_metadata;
  set tmplts.studycolumnmetadata work.column_metadata;
  sasref = "SRCDATA";
  /* Date/Time related items do not have a length in Define-XML */
  if missing(length) and xmldatatype in
     ('datetime' 'date' 'time' 'partialDate' 'partialTime'
      'partialDatetime' 'incompleteDatetime' 'durationDatetime')
     then length=64;
run;
```

APPENDIX 2: IMPORTING DEFINE-XML USING PROC GOOVY

import_DefineXML_metadata_Groovy.sas

```
%let Root=C:/_Data/Presentations/PharmaSUG_2018/Accessing_DefineXML;

libname groovy "&Root/groovy/metadata";
libname tmplts "&Root/templates";
libname metadata "&Root/Groovy/metadata";

proc format;
  value $typ 'char'='$' num=' ';
run;

filename groovy "&Root/groovy/groovy-all-2.4.14.jar";
filename opencsv "%sysfunc(pathname(work,l))/opencsv-4.1.jar";
proc http
   method = "get"
   url    = "https://sourceforge.net/projects/opencsv/files/latest/download?source=files"
   out    = opencsv;
run;

proc groovy;
  add classpath=groovy;
  add classpath=opencsv;
  execute parseonly "&Root/groovy/import_DefineXML_metadata.groovy";
run;

%let xmlFile=&Root/xml/define2-0-0-example-sdtm.xml;
%let xsdFile=&Root/schema-repository/cdisc-definexml-2.0.0/define2-0-0.xsd;
%let csvOutputFolder=%sysfunc(pathname(work));
%let tableMetadataCSV=&csvOutputFolder/tablemetadata.csv;
%let columnMetadataCSV=&csvOutputFolder/columnmetadata.csv;

data _null_;
   declare javaobj validatexml("ValidateXML");
   validatexml.exceptiondescribe(1);
   validatexml.callVoidMethod("validateXML", "&xmlFile", "&xsdFile");
   validatexml.delete();

   declare javaobj tables("TableMetadataSlurper");
   tables.exceptiondescribe(1);
   tables.callVoidMethod("setXmlFilename", "&xmlFile");
   tables.callVoidMethod("setCsvFilename", "&tableMetadataCSV");
   tables.callVoidMethod("createTableMetadata");
   tables.delete();

   declare javaobj columns("ColumnMetadataSlurper");
   columns.exceptiondescribe(1);
   columns.setStringField("xmlFilename", "&xmlFile");
   columns.setStringField("csvFilename", "&columnMetadataCSV");
   columns.callVoidMethod("createColumnMetadata");
   columns.delete();
run;
```

```
*******************************************************************************;
*** Table metadata                                                            ***;
*******************************************************************************;
proc sql noprint;
  select catx(' ', name, cats(put(type, $typ.))) into: tableinput separated by ' '
  from dictionary.columns
  where (upcase(libname)='TMPLTS' and upcase(memname)='STUDYTABLEMETADATA')
  order by varnum
  ;
  select catx(' ', name, cats(put(type, $typ.)), length) into: tablelength separated by ' '
  from dictionary.columns
  where (upcase(libname)='TMPLTS' and upcase(memname)='STUDYTABLEMETADATA')
  order by varnum
  ;
quit;

data work.table_metadata;
  infile "&csvOutputFolder/tablemetadata.csv" delimiter='09'x missover dsd lrecl=32767 firstobs=2
;
  length &tablelength;
  input &tableinput;
run;

data metadata.table_metadata;
  set tmplts.studytablemetadata work.table_metadata;
  comment=tranwrd(comment, "\n", '0A'x);
run;

*******************************************************************************;
*** Column metadata                                                           ***;
*******************************************************************************;
proc sql noprint;
  select catx(' ', name, cats(put(type, $typ.))) into: columninput separated by ' '
  from dictionary.columns
  where (upcase(libname)='TMPLTS' and upcase(memname)='STUDYCOLUMNMETADATA')
  order by varnum
  ;
  select catx(' ', name, cats(put(type, $typ.), length)) into: columnlength separated by ' '
  from dictionary.columns
  where (upcase(libname)='TMPLTS' and upcase(memname)='STUDYCOLUMNMETADATA')
  order by varnum
  ;
quit;

data work.column_metadata;
  infile "&csvOutputFolder/columnmetadata.csv" delimiter='09'x missover dsd lrecl=32767
firstobs=2 ;
  length &columnlength;
  input &columninput;
run;

data metadata.column_metadata;
  set tmplts.studycolumnmetadata work.column_metadata;
  comment=tranwrd(comment, "\n", '0A'x);
  algorithm=tranwrd(algorithm, "\n", '0A'x);
  formalexpression=strip(tranwrd(formalexpression, "\n", '0A'x));
  if missing(length) and xmldatatype in
```

```
     ('datetime' 'date' 'time' 'partialDate' 'partialTime'
      'partialDatetime' 'incompleteDatetime' 'durationDatetime')
     then length=64;
run;
```

import_DefineXML_metadata.groovy

```
import com.opencsv.CSVWriter
import org.xml.sax.ErrorHandler
import org.xml.sax.SAXParseException
import static javax.xml.XMLConstants.W3C_XML_SCHEMA_NS_URI
import javax.xml.transform.stream.StreamSource
import javax.xml.validation.Schema
import javax.xml.validation.SchemaFactory
import javax.xml.validation.Validator

public class ValidateXML {

  public void validateXML(String xmlFile, String xsdFile) {
    try {
      SchemaFactory factory = SchemaFactory.newInstance( W3C_XML_SCHEMA_NS_URI )
      Schema schema = factory.newSchema( new StreamSource( xsdFile ) )
      Validator validator = schema.newValidator()

      List exceptions = []
      Closure<Void> handler = { exception -> exceptions << exception }
      validator.errorHandler = [ warning:    handler,
                                 fatalError: handler,
                                 error:      handler ] as ErrorHandler
      validator.validate( new StreamSource( xmlFile ) )

      exceptions.each {
        println "ERROR: [validateXML] $xmlFile is not valid."
        println "$it.lineNumber:$it.columnNumber: $it.message"
      }

    } catch (FileNotFoundException e) {
      println "ERROR: [validateXML] $xmlFile can not be found."
    } catch (SAXParseException e) {
      println "ERROR: [validateXML] XML schema validation issues with $xmlFile."
      println "ERROR: [validateXML] " + "${e.message}."
    } catch (Exception e) {
      println "ERROR: [validateXML] "+ "${e.message}."
    }
  }
}

private class DefineXMLParser {
  def parseFile(path) {
      def definexml = new File(path).text
      def define = new XmlSlurper().parseText(definexml)
      def ns = [
        "" : "http://www.cdisc.org/ns/odm/v1.3",
        "def" : "http://www.cdisc.org/ns/def/v2.0",
```

```
        "xlink" : "http://www.w3.org/1999/xlink",
        "arm" : "http://www.cdisc.org/ns/arm/v1.0"
        ]
      define.declareNamespace(ns)
      return define
  }
}

private class CSVFile {
  void createCSV(String [] header, List metadata, String path) {
    def csvout = new FileWriter(path)
    try {
      CSVWriter writer = new CSVWriter(csvout, '\t' as char, '\0' as char, '\0' as char);
      writer.writeNext(header)
      writer.writeAll(metadata)
      writer.close()
    } catch (FileNotFoundException e) {
      println "ERROR: [CSVFile] $path could not be found"
    } catch (Exception e) {
      println "ERROR: [CSVFile] " + e.toString()
    }
  }
}

public class TableMetadataSlurper {

  private String xmlFilename = null
  private String csvFilename = null

  public void setXmlFilename(String xmlFilename) {
      this.xmlFilename = xmlFilename
  }

  public void setCsvFilename(String csvFilename) {
      this.csvFilename = csvFilename
  }

  public void createTableMetadata() {

    DefineXMLParser myParser = new DefineXMLParser()
    try {
      def define = myParser.parseFile(xmlFilename)

      String [] header = ["sasref", "table", "label", "order", "repeating", "isreferencedata",
                          "domain", "domaindescription", "class", "xmlpath", "xmltitle",
                          "structure", "purpose", "keys", "state", "date", "comment",
                          "studyversion", "standard", "standardversion"
                         ]
      String creationDateTime = define.@CreationDateTime.toString().substring(0,10)
      String studyVersion = define.Study.MetaDataVersion.@OID.toString()
      String standard = define.Study.MetaDataVersion.@'def:StandardName'.toString()
      String standardVersion = define.Study.MetaDataVersion.@'def:StandardVersion'.toString()
      Integer order=0

      def itemGroups=[]
      def itemGroupDefs = define.Study.MetaDataVersion.ItemGroupDef
```

```
      itemGroupDefs.each {

        def itemRefs = it.ItemRef
        Map keyMap = [:]
        itemRefs.each {
            if (it.@KeySequence.toString() != "") {
                def itemOID = it.@ItemOID
                def itemDef = define.Study.MetaDataVersion.ItemDef.find {it.@OID == itemOID}
                keyMap[(it.@KeySequence).toInteger()]= itemDef.@Name
            }
        }
        String keys = keyMap.sort { a, b -> a.key <=> b.key }.values().toArray().join(' ')


        order+=1
        String commentOID = it.'@def:CommentOID'.toString()
        def commentDef = define.Study.MetaDataVersion.'def:CommentDef'.find {it.@OID ==
commentOID}
        String archiveLocationID = it.'@def:ArchiveLocationID'.toString()
        def leaf = it.'def:leaf'.find {it.@ID == archiveLocationID}
        def domainDescription = it.Alias.find {it.@Context == 'DomainDescription'}

        String[] itemGroup = new String[header.size()]
        itemGroup = [
          'SRCDATA',
          it.@Name.toString(),
          it.Description.TranslatedText.text(),
          order.toString(),
          it.@Repeating.toString(),
          it.@IsReferenceData.toString(),
          it.@Domain.toString(),
          domainDescription.@Name.toString(),
          it.'@def:Class'.toString(),
          leaf.'@xlink:href'.toString(),
          leaf.'def:title'.toString(),
          it.'@def:Structure'.toString(),
          it.@Purpose.toString(),
          keys,
          'Final',
          creationDateTime,
          commentDef.Description.TranslatedText.text().trim().replace("\n", "\\n"),
          studyVersion,
          standard,
          standardVersion
        ]
        itemGroups << itemGroup

      }

      CSVFile csvFile = new CSVFile()
      try {
        csvFile.createCSV(header, itemGroups, csvFilename)
      } catch (FileNotFoundException e) {
        println "ERROR: [createTableMetadata] $csvFilename can not be created."
      }

    } catch (FileNotFoundException e) {
```

```
      println "ERROR: [createTableMetadata] $xmlFilename can not be found."
    } catch (Exception e) {
      println "ERROR: [createTableMetadata] " + e.toString()
    }
  }
}

public class ColumnMetadataSlurper {

  private String xmlFilename = null
  private String csvFilename = null

  public void setXmlFilename(String xmlFilename) {
      this.xmlFilename = xmlFilename
  }

  public void setCsvFilename(String csvFilename) {
      this.csvFilename = csvFilename
  }

  public void createColumnMetadata() {

    DefineXMLParser myParser = new DefineXMLParser()
    try {
      def define = myParser.parseFile(xmlFilename)

      String [] header = ["sasref", "table", "column", "label", "order", "type", "length",
                          "displayformat", "significantdigits", "xmldatatype", "xmlcodelist",
                          "core", "origin", "origindescription", "Role", "algorithm",
                          "algorithmtype", "formalexpression", "formalexpressioncontext",
                          "comment", "studyversion", "standard", "standardversion"
                         ]
      String studyVersion = define.Study.MetaDataVersion.@OID.toString()
      String standard = define.Study.MetaDataVersion.@'def:StandardName'.toString()
      String standardVersion = define.Study.MetaDataVersion.@'def:StandardVersion'.toString()

      def items = []
      def itemGroupDefs = define.Study.MetaDataVersion.ItemGroupDef

      itemGroupDefs.each {

        def itemRefs = it.ItemRef
        itemRefs.each {

          String itemOID = it.@ItemOID.toString()
          def itemDef = define.Study.MetaDataVersion.ItemDef.find {it.@OID == itemOID}
          String commentOID = itemDef.'@def:CommentOID'.toString()
          def commentDef = define.Study.MetaDataVersion.'def:CommentDef'.find {it.@OID ==
commentOID}
          String methodOID = it.@MethodOID.toString()
          def methodDef = define.Study.MetaDataVersion.MethodDef.find{it.@OID == methodOID}

          String dataType = itemDef.@DataType.toString()
          String type = null
          if (dataType == 'float' | dataType == 'integer') {
              type = 'N'
            } else {
```

```
              type = 'C'
            }

            String[] item = new String[header.size()]
            item = [
              'SRCDATA',
              it.parent().@Name.toString(),
              itemDef.@Name.toString(),
              itemDef.Description.TranslatedText.text(),
              it.@OrderNumber.toString(),
              type,
              itemDef.@Length.toString(),
              itemDef.@'def:DisplayFormat'.toString(),
              itemDef.@SignificantDigits.toString(),
              itemDef.@DataType.toString(),
              itemDef.CodeListRef.@CodeListOID.toString(),
              '',
              itemDef.'def:Origin'.@Type.toString(),
              itemDef.'def:Origin'.Description.TranslatedText.toString(),
              it.@Role.toString(),
              methodDef.Description.TranslatedText.text().trim().replace("\n", "\\n"),
              methodDef.@Type.toString(),
              methodDef.FormalExpression.text().trim().replace("\n", "\\n"),
              methodDef.FormalExpression.@Context.toString(),
              commentDef.Description.TranslatedText.text().trim().replace("\n", "\\n"),
              studyVersion,
              standard,
              standardVersion
            ]
            items << item

          }
        }
        CSVFile csvFile = new CSVFile()
        try {
          csvFile.createCSV(header, items, csvFilename)
        } catch (FileNotFoundException e) {
          println "ERROR: [createColumnMetadata] $csvFilename can not be created."
        }

      } catch (FileNotFoundException e) {
        println "ERROR: [createColumnMetadata] $xmlFilename can not be found."
      } catch (Exception e) {
        println "ERROR: [createTableMetadata] " + e.toString()
      }
    }
}
```

APPENDIX 3: IMPORTING DEFINE-XML USING XSLT

DefineXML.xls

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:odm="http://www.cdisc.org/ns/odm/v1.3"
  xmlns:def="http://www.cdisc.org/ns/def/v2.0"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:arm="http://www.cdisc.org/ns/arm/v1.0">
  <xsl:output method="xml" encoding="UTF-8" indent="yes"/>

  <xsl:variable name="ODM" select="/odm:ODM"/>
  <xsl:variable name="Study" select="$ODM/odm:Study"/>
  <xsl:variable name="MetaDataVersion" select="$Study/odm:MetaDataVersion"/>
  <xsl:variable name="ItemGroupDefs" select="$MetaDataVersion/odm:ItemGroupDef"/>
  <xsl:variable name="ItemDefs" select="$MetaDataVersion/odm:ItemDef"/>
  <xsl:variable name="MethodDefs" select="$MetaDataVersion/odm:MethodDef"/>
  <xsl:variable name="CommentDefs" select="$MetaDataVersion/def:CommentDef"/>

  <xsl:template match="odm:ODM">
    <xsl:element name="LIBRARY">
      <xsl:call-template name="ItemGroupDef"/>
    </xsl:element>
  </xsl:template>

  <xsl:template name="ItemGroupDef">

    <xsl:for-each select="//odm:ItemGroupDef">

      <xsl:variable name="ArchiveLocationID" select="@def:ArchiveLocationID"/>
      <xsl:variable name="CommentOID" select="@def:CommentOID"/>

      <xsl:variable name="KeySequence" select="odm:ItemRef/@KeySequence"/>
      <xsl:variable name="n_keys" select="count($KeySequence)"/>
      <xsl:variable name="keys" >
        <xsl:for-each select="odm:ItemRef">
          <xsl:sort select="@KeySequence" data-type="number" order="ascending"/>
          <xsl:if test="@KeySequence[ .!='' ]">
            <xsl:variable name="ItemOID" select="@ItemOID"/>
            <xsl:variable name="Name" select="$ItemDefs[@OID=$ItemOID]"/>
            <xsl:value-of select="$Name/@Name"/>
            <xsl:if test="@KeySequence &lt; $n_keys"><xsl:text> </xsl:text></xsl:if>
          </xsl:if>
        </xsl:for-each>
      </xsl:variable>

      <xsl:element name="ItemGroupDef">
        <xsl:element name="table"><xsl:value-of select="@Name"/></xsl:element>
        <xsl:element name="label"><xsl:value-of
select="odm:Description/odm:TranslatedText/text()"/></xsl:element>
        <xsl:element name="order"><xsl:value-of select="position()"/></xsl:element>
        <xsl:element name="repeating"><xsl:value-of select="@Repeating"/></xsl:element>
```

```
        <xsl:element name="isreferencedata"><xsl:value-of
select="@IsReferenceData"/></xsl:element>
        <xsl:element name="domain"><xsl:value-of select="@Domain"/></xsl:element>
        <xsl:element name="domaindescription"><xsl:value-of
select="odm:Alias[@Context='DomainDescription']/@Name"/></xsl:element>
        <xsl:element name="class"><xsl:value-of select="@def:Class"/></xsl:element>
        <xsl:element name="xmlpath"><xsl:value-of select="def:leaf[@ID =
$ArchiveLocationID]/@xlink:href"/></xsl:element>
        <xsl:element name="xmltitle"><xsl:value-of select="def:leaf[@ID =
$ArchiveLocationID]/def:title/text()"/></xsl:element>
        <xsl:element name="structure"><xsl:value-of select="@def:Structure"/></xsl:element>
        <xsl:element name="purpose"><xsl:value-of select="@Purpose"/></xsl:element>
        <xsl:element name="keys"><xsl:value-of select="$keys"/></xsl:element>
        <xsl:element name="date"><xsl:value-of select="substring($ODM/@CreationDateTime, 1,
10)"/></xsl:element>
        <xsl:element name="comment"><xsl:value-of select="$CommentDefs[@OID =
$CommentOID]/odm:Description/odm:TranslatedText/text()"/></xsl:element>
        <xsl:element name="studyversion"><xsl:value-of
select="$MetaDataVersion/@OID"/></xsl:element>
        <xsl:element name="standard"><xsl:value-of
select="$MetaDataVersion/@def:StandardName"/></xsl:element>
        <xsl:element name="standardversion"><xsl:value-of
select="$MetaDataVersion/@def:StandardVersion"/></xsl:element>
      </xsl:element>

      <xsl:for-each select="odm:ItemRef">
        <xsl:call-template name="ItemRefItemDef"/>
      </xsl:for-each>

    </xsl:for-each>
  </xsl:template>

  <xsl:template name="ItemRefItemDef">
    <xsl:for-each select=".">

      <xsl:variable name="ItemOID" select="@ItemOID"/>
      <xsl:variable name="ItemDef" select="$ItemDefs[@OID = $ItemOID]"/>
      <xsl:variable name="MethodOID" select="@MethodOID"/>
      <xsl:variable name="CommentOID" select="$ItemDef/@def:CommentOID"/>

      <xsl:element name="ItemRefItemDef">
        <xsl:element name="table"><xsl:value-of select="../@Name"/></xsl:element>
        <xsl:element name="column"><xsl:value-of select="$ItemDef/@Name"/></xsl:element>
        <xsl:element name="label"><xsl:value-of
select="$ItemDef/odm:Description/odm:TranslatedText/text()"/></xsl:element>
        <xsl:element name="order"><xsl:value-of select="@OrderNumber"/></xsl:element>
        <xsl:element name="length"><xsl:value-of select="$ItemDef/@Length"/></xsl:element>
        <xsl:element name="displayformat"><xsl:value-of
select="$ItemDef/@def:DisplayFormat"/></xsl:element>
        <xsl:element name="significantdigits"><xsl:value-of
select="$ItemDef/@SignificantDigits"/></xsl:element>
        <xsl:element name="xmldatatype"><xsl:value-of select="$ItemDef/@DataType"/></xsl:element>
        <xsl:element name="xmlcodelist"><xsl:value-of
select="$ItemDef/odm:CodeListRef/@CodeListOID"/></xsl:element>
        <xsl:element name="origin"><xsl:value-of
select="$ItemDef/def:Origin/@Type"/></xsl:element>
        <xsl:element name="origindescription"><xsl:value-of
```

```
select="$ItemDef/def:Origin/odm:Description/odm:TranslatedText/text()"/></xsl:element>
        <xsl:element name="role"><xsl:value-of select="@Role"/></xsl:element>
        <xsl:element name="algorithm"><xsl:value-of select="$MethodDefs[@OID =
$MethodOID]/odm:Description/odm:TranslatedText/text()"/></xsl:element>
        <xsl:element name="algorithmtype"><xsl:value-of select="$MethodDefs[@OID =
$MethodOID]/@Type"/></xsl:element>
        <xsl:element name="formalexpression"><xsl:value-of select="$MethodDefs[@OID =
$MethodOID]/odm:FormalExpression/text()"/></xsl:element>
        <xsl:element name="formalexpressioncontext"><xsl:value-of select="$MethodDefs[@OID =
$MethodOID]/odm:FormalExpression/@Context"/></xsl:element>
        <xsl:element name="comment"><xsl:value-of select="$CommentDefs[@OID =
$CommentOID]/odm:Description/odm:TranslatedText/text()"/></xsl:element>
        <xsl:element name="studyversion"><xsl:value-of
select="$MetaDataVersion/@OID"/></xsl:element>
        <xsl:element name="standard"><xsl:value-of
select="$MetaDataVersion/@def:StandardName"/></xsl:element>
        <xsl:element name="standardversion"><xsl:value-of
select="$MetaDataVersion/@def:StandardVersion"/></xsl:element>
      </xsl:element>

    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

CreateXMLMap.sas

```
%macro CreateXMLMap(library=, metadatadataset=, tablepath=, tablename=, mapfileref=);

    proc sql noprint;
      create table work.__attributes as
      select memname as table, strip(name) as name, strip(label) as label, type, length
      from dictionary.columns
      where (upcase(libname)=%upcase("&library") and upcase(memname)=%upcase("&metadatadataset"))
      order by varnum
      ;
    quit;

    data _null_;
      length element $400 numval dtype ddtype $10;
      set work.__attributes END=eof;
      file &mapfileref;
      by table;
      if _n_ = 1 then do;
        put '<?xml version="1.0" encoding="UTF-8"?>';
        put '<SXLEMAP name="define" version="2.1">';
      end;

      dtype = ifc(upcase(type)="CHAR", "character", "numeric");
      ddtype = ifc(upcase(type)="CHAR", "string", "integer");
      numval = strip(input(length, best12.));

      if first.table then do;
        put '<TABLE name="' "&tablename" '">';
```

```
        put '<TABLE-PATH syntax="XPath">/' "&tablepath" '</TABLE-PATH>';
      end;

      element=catt('<COLUMN name="', name, '">');
      put element;
      element=catt('<PATH syntax="XPath">/', "&tablepath", "/", name, '</PATH>');
      put element;
      element=catt('<TYPE>',dtype,'</TYPE>');
      put element;
      element=catt('<DATATYPE>', ddtype, '</DATATYPE>');
      put element;
      element=catt('<DESCRIPTION>', label, '</DESCRIPTION>');
      put element;
      element=catt('<LENGTH>', numval, '</LENGTH>');
      put element;
      put '</COLUMN>';

      if last.table then put "</TABLE>";
      if eof then put "</SXLEMAP>";
    run;

%mend CreateXMLMap;
```

import_DefineXML_metadata_XSLT.sas

```
%let Root=C:/_Data/Presentations/PharmaSUG_2018/Accessing_DefineXML;

%include "&Root/xslt/CreateXMLMap.sas";

libname xslt "&Root/xslt/metadata";
libname tmplts "&Root/templates";
libname metadata "&Root/xslt/metadata";
libname out "&Root/xslt/out";

filename xml "&Root/xml/define2-0-0-example-sdtm.xml";
filename xsl "&Root/xslt/stylesheets/DefineXML.xsl";
filename flatxml "&Root/xslt/out/DefineXML_flat.xml";

proc xsl in=xml xsl=xsl out=flatxml;
run;

*************************************************************************;
*** Table metadata                                                    ***;
*************************************************************************;
filename tabmap "&Root/xslt/out/definexml_tables.map";

%CreateXMLMap(
  library=tmplts,
  metadatadataset=studytablemetadata,
  tablepath=LIBRARY/ItemGroupDef,
  tablename=table_metadata,
  mapfileref=tabmap
);

libname define xmlv2 xmlfileref=flatxml xmlmap=tabmap;
```

```
data metadata.table_metadata;
  set define.table_metadata;
  sasref = "SRCDATA";
  state = "Final";
run;


*************************************************************************;
*** Column metadata                                                   ***;
*************************************************************************;
filename colmap "&Root/xslt/out/definexml_columns.map";

%CreateXMLMap(
  library=tmplts,
  metadatadataset=studycolumnmetadata,
  tablepath=LIBRARY/ItemRefItemDef,
  tablename=column_metadata,
  mapfileref=colmap
);

libname define xmlv2 xmlfileref=flatxml xmlmap=colmap;

data metadata.column_metadata;
  set define.column_metadata;
  sasref = "SRCDATA";
  type = ifc(xmldatatype in ('integer' 'float'), 'N', 'C');
  if missing(length) and xmldatatype in
     ('datetime' 'date' 'time' 'partialDate' 'partialTime'
      'partialDatetime' 'incompleteDatetime' 'durationDatetime')
     then length=64;
run;
```

APPENDIX 4: IMPORTING DEFINE-XML USING SAS CLINICAL STANDARDS TOOLKIT

```
%let Root=C:/_Data/Presentations/PharmaSUG_2018/Accessing_DefineXML;

%let _cstStandard=CDISC-DEFINE-XML;
%let _cstStandardVersion=2.0.0;

%let _cstTrgStandard=CDISC-SDTM;
%let _cstTrgStandardVersion=3.1.2;
%let _cstDefineFile=define2-0-0-example-sdtm.xml;
%let workPath=%sysfunc(pathname(work));

*******************************************************************************;
* One strategy to defining the required library and file metadata for a CST process    *;
* is to optionally build SASReferences in the WORK library.  An example of how to do   *;
*  this follows.                                                                       *;
*                                                                                      *;
* The call to cstutil_processsetup below tells CST how SASReferences will be provided  *;
*  and referenced.  If SASReferences is built in work, the call to cstutil_processsetup*;
*  may, assuming all defaults, be as simple as:                                        *;
*        %cstutil_processsetup()                                                       *;
*******************************************************************************;

%let _cstSetupSrc=SASREFERENCES;

%cst_createdsfromtemplate(_cstStandard=CST-FRAMEWORK, _cstType=control,_cstSubType=reference,
_cstOutputDS=work.sasreferences);

proc sql;
  insert into work.sasreferences
  values("CST-FRAMEWORK"    "1.2"                  "messages"   ""           "messages"
         "libref" "input"  "dataset"   "N" "" ""          1  ""                        "")
  values("&_cstStandard"    "&_cstStandardVersion"  "messages"   ""           "crtmsg"
         "libref" "input"  "dataset"   "N" "" ""          2  ""                        "")
  values("&_cstStandard"    "&_cstStandardVersion"  "autocall"   ""           "crtcode"
         "fileref" "input"  "folder"   "N" "" ""          1  ""                        "")
  values("&_cstStandard"    "&_cstStandardVersion"  "properties" "initialize" "inprop"
         "fileref" "input"  "file"     "N" "" ""          1  ""                        "")
  values("&_cstStandard"    "&_cstStandardVersion"  "results"    "results"    "results"
         "libref"  "output" "dataset"  "Y" "" "&Root/cst/results" . "xml2source_results_sdtm" "")
  values("&_cstStandard"    "&_cstStandardVersion"  "externalxml"       "xml"        "crtxml"
         "fileref" "input"  "file"     "N" "" "&Root/xml"        . "&_cstDefineFile"      "")
  values("&_cstStandard"    "&_cstStandardVersion"  "referencexml"      "map"        "crtmap"
         "fileref" "input"  "file"     "N" "" "&Root/cst/referencexml" . "define.map"     "")
  values("&_cstStandard"    "&_cstStandardVersion"  "sourcedata"        ""           "srcdata"
         "libref"  "output" "folder"   "Y" "" "&workPath" . ""                          "")
  values("&_cstStandard"    "&_cstStandardVersion"  "studymetadata"     "study"      "trgmeta"
         "libref"  "output" "folder"   "Y" "" "&Root/cst/metadata"  . ""      "")
  values("&_cstTrgStandard" "&_cstTrgStandardVersion"  "referencemetadata" "table"   "refmeta"
         "libref"  "input"  "dataset"  "N" "" ""          . ""                        "")
  values("&_cstTrgStandard" "&_cstTrgStandardVersion"  "referencemetadata" "column"  "refmeta"
         "libref"  "input"  "dataset"  "N" "" ""          . ""                        "")
  values("&_cstTrgStandard" "&_cstTrgStandardVersion"  "classmetadata"     "column"  "refmeta"
         "libref"  "input"  "dataset"  "N" "" ""          . ""                        "")
  values("&_cstTrgStandard" "&_cstTrgStandardVersion"  "classmetadata"     "table"   "refmeta"
         "libref"  "input"  "dataset"  "N" "" ""          . ""                        "")
```

```
  ;
quit;

************************************************;
* Process SASReferences file.                  *;
************************************************;
%cstutil_processsetup();

*************************************************************************;
* Run the schema validation macro.                                      *;
*************************************************************************;
%cstutilxmlvalidate();

*************************************************************************;
* Run the standard-specific Define-XML macros.                          *;
*************************************************************************;
%define_read();

*************************************************************************;
* Run the standard-specific Define-XML macros.                          *;
*************************************************************************;
%define_createsrcmetafromdefine(
  _cstDefineDataLib=srcdata,
  _cstTrgStandard=&_cstTrgStandard,
  _cstTrgStandardVersion=&_cstTrgStandardVersion,
  _cstTrgStudyDS=trgmeta.source_study,
  _cstTrgTableDS=trgmeta.source_tables,
  _cstTrgColumnDS=trgmeta.source_columns,
  _cstTrgCodeListDS=trgmeta.source_codelists,
  _cstTrgValueDS=trgmeta.source_values,
  _cstTrgDocumentDS=trgmeta.source_documents,
  _cstTrgAnalysisResultDS=trgmeta.source_analysisresults,
  _cstLang=en,
  _cstUseRefLib=Y,
  _cstRefTableDS=refmeta.reference_tables,
  _cstRefColumnDS=refmeta.reference_columns,
  _cstClassTableDS=refmeta.class_tables,
  _cstClassColumnDS=refmeta.class_columns,
  _cstReturn=_cst_rc,
  _cstReturnMsg=_cst_rcmsg
  );
```