PharmaSUG2010 - Paper CD14

Accessing the metadata from the define.xml using XSLT transformations

Lex Jansen, Octagon Research Solutions, Wayne, PA

ABSTRACT

When submitting clinical study data in electronic format to the FDA, not only information from trials has to 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 data sets. When submitting data in the CDISC SDTM format it is required to submit the data definition file in the Case Report Tabulation Data Definition Specification (define.xml) format as prepared by the CDISC define.xml team.

This paper illustrates how XML technology (XSLT) can be used to turn the metadata from the define.xml file into executable SAS code.

We will create the following examples of executable SAS code from the define.xml file:

- SAS formats from the define.xml code lists
- Data set templates from the define.xml itemgroups and items

INTRODUCTION

The FDA issued the Final Guidance on Electronic Submissions using the eCTD specifications in April 2006. The latest revision of this guidance was published in June 2008 [1]

Technical specifications associated with this guidance are provided as stand-alone documents. Among these are Study Data Specifications [2] that provide further guidance for submitting animal and human study data in electronic format when providing electronic submissions to the FDA. Study data includes information from trials submitted to the agency for evaluation and information to understand the data (data definition). The study data includes both raw and derived data.

As of January 1, 2008, sponsors submitting data electronically to the FDA are required to follow the new eCTD guidance. The previous guidance, originally issued in 1999, has been withdrawn as of the same date.

The new guidance differs from the 1999 guidance in one significant aspect: The application table of contents is no longer submitted as a PDF file, but is submitted as XML (eXtensible Markup Language). This means that the electronic submissions will now be XML based.

The current version of the Study Data Specifications contains specifications for the Data Tabulation data sets of human drug product clinical studies and provides a reference to the Study Data Tabulation Model (SDTM) [3][4][5][6] developed by the Submission Data Standard (SDS) working group of the Clinical Data Interchange Standard Consortium (CDISC).

Further, the Study Data Specifications document gives a reference to the Case Report Tabulation Data Definition Specification (CRT-DDS or define.xml) developed by the CDISC define.xml Team [7].

DATA DEFINITION TABLES: define.xml

Released for implementation in February 2005, the Case Report Tabulation Data Definition Specification (CRT-DDS or define.xml) Version 1.0 [7] specifies the standard for providing Case Report Tabulations Data Definitions in an XML format for submission to regulatory authorities (e.g., FDA). The XML schema used to define the expected structure for these XML files is based on an extension to the CDISC Operational Data Model (ODM). The current version of the CRT-DDS (version 1.0) is based on version 1.2.1 of the CDISC ODM [8], which is both semantically and syntactically identical to Version 1.2.0 of the CDISC ODM.

The Data Definition Document provides a list of the data sets included in the submission along with a detailed description of the contents of each data set. To increase the level of automation and improve the efficiency of the Regulatory Review process, the define.xml file can be used to provide the Data Definition Document in a machine-readable format.

USING THE METADATA IN THE DEFINE.XML

The define.xml file contains metadata that describes the SAS data sets that will be submitted to the FDA. Often, the define.xml file will be created after these SAS data sets have been created. However, it would also be possible to use the define.xml as a *specification* for the SAS data sets. The define.xml 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 we can create 0-observation data sets, that can act as data set specifications.

This paper will show how to generate SAS code from the define.xml that can be used to create an empty data set with the appropriate attributes. The SQL code that we will generate looks like the following example:

```
  CREATE TABLE DM(LABEL="Demographics") (
    STUDYID CHAR(7) LABEL="Study Identifier",
    DOMAIN CHAR(2) LABEL="Domain Abbreviation",
    USUBJID CHAR(14) LABEL="Unique Subject Identifier",
    SUBJID CHAR(6) LABEL="Subject Identifier for the Study",
    RFSTDTC CHAR(10) LABEL="Subject Reference Start Date/Time",
    RFENDTC CHAR(10) LABEL="Subject Reference End Date/Time",
    SITEID CHAR(3) LABEL="Study Site Identifier",
    BRTHDTC CHAR(10) LABEL="Date/Time of Birth",
    AGE NUMERIC LABEL="Age in AGEU at RFSTDTC",
    AGEU CHAR(5) LABEL="Age Units",
...
    );
```

Another example of metadata that is stored in the define.xml are codelists. A codelist defines a discrete set of values (controlled terminology) that is expected for a variable. In a submission, controlled terminology or decoded text should be used instead of arbitrary number codes in order to reduce ambiguity for submission reviewers. For example, for concomitant medications, the verbatim term and/or dictionary term should be presented, rather than numeric codes.

The define.xml file may contain codelists with pairs of code/decode values that can be used to map coded values to their decoded equivalents. By turning the codelists from the define.xml into SAS formats, the process of mapping coded values into decoded values can be supported.

Before this paper discusses the process of transforming XML into SAS code, it is important that the reader has a basic understanding of several XML standards.

XML 101

In this section we present a short introduction to XML.

BASIC SYNTAX

The Extensible Markup Language (XML) [9] is a general-purpose markup language. It is classified as an extensible language because it allows users to define their own elements. Its primary purpose is to facilitate the sharing of structured data across different information systems. XML is a language that is hierarchical, text-based and describes data in terms of markup tags. A good introductory guide to XML can be found in the reference section [10].

Every XML document starts with the **XML declaration**. This is a small collection of details that prepares an XML processor for working with the document.

syntax	XML declaration	example
`<?xml param1 param2 … ?>`		`<?xml version="1.0" encoding="ISO-8859-1" ?>`

Elements are the basic building blocks of XML, dividing a document into a hierarchy of regions. Some elements are containers, holding text or (child) elements. Others are empty and serve as place markers. Every XML file should have exactly 1 root element that contains all other elements.

syntax	Container Element	example
< name *attribute1 attribute2 …* > *content* </ name >		<def:leaf ID="**Location.DM**" xlink:href="**dm.xpt**"> <def:title>**dm.xpt**</def:title> </def:leaf>

An **empty element** is similar, but contains no content or end tag.

syntax	Empty Element	example
< name *attribute1 attribute2 …* />		<ItemRef ItemOID="**STUDYID**" OrderNumber="**1**" Mandatory="**Yes**" Role="**IDENTIFIER**" RoleCodeListOID="**RoleCodeList**" />

In the element starting tag there can be information about the element in the form of an **attribute**. Attributes define properties of elements. They associate a name with a value, which is a string of character data enclosed in quotes. There is no limit to how many attributes an element can have, as long as no two attributes have the same name.

syntax	Attributes	example
name = " value "		ItemOID="**STUDYID**" OrderNumber="**1**" Mandatory="**Yes**" Role="**IDENTIFIER**" RoleCodeListOID="**RoleCodeList**"

Namespaces are a mechanism by which element and attribute names can be assigned to groups. They are most often used when combining different vocabularies in the same document. If each vocabulary is given a namespace then the ambiguity between identically named elements or attributes can be resolved.

syntax	Namespaces	example
xmlns: *name = " URI "*		xmlns="**http://www.cdisc.org/ns/odm/v1.2**" xmlns:xsi="**http://www.w3.org/2001/XMLSchema-instance**" xmlns:xlink="**http://www.w3.org/1999/xlink**" xmlns:def="**http://www.cdisc.org/ns/def/v1.0**"

Comments in an XML document are not interpreted by the XML processor:

syntax	Comments	example
<!-- *comment text* -->		<!-- File: define.xml --> <!-- Date: 02/02/2005 --> <!-- Author: Clinical Data Interchange Standards --> <!-- Consortium (CDISC) define.xml team -->

A **CDATA** (character data) section tells the XML parser that this section of the document contains no markup and should be treated as regular text. CDATA sections can be useful for large regions of text that contain a lot of 'forbidden' XML characters. They should be used with care though, since it may be hard to use any elements or attributes inside the marked region.

syntax	CDATA	example
`<![CDATA unparsed character data ]]>`	<TranslatedText xml:lang="en"><![CDATA[Classifies subjects based on Hy's Law - 2 variations - For each variation, subject is classified as normal or abnormal at baseline and as normal or abnormal based on most extreme value during treatment, Safety population]]></TranslatedText>	

Processing instructions are meant to provide information to a specific XML processor, but may not be relevant to others. It is a container for data that is targeted toward a specific XML processor. The processing instruction looks like the XML declaration, but is targeted at a specific XML processor. The XML declaration can be viewed as a processing instruction for all XML processors.

syntax	Processing Instructions	example
`<? target data ?>`	<?xml-stylesheet type="text/xsl" href="define1-0-0.xsl" ?>	

Now that we have explained the most important building blocks of XML, we can look at a more complete example in Figure 1.

Figure 1: *Excerpt of an XML file (define.xml)*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet type="text/xsl" href="define1-0-0.xsl"?>
<ODM xmlns="http://www.cdisc.org/ns/odm/v1.2"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:def="http://www.cdisc.org/ns/def/v1.0"
     xsi:schemaLocation="http://www.cdisc.org/ns/odm/v1.2 define1-0-0.xsd"
     FileOID="Studycdisc01"
     ODMVersion="1.2" FileType="Snapshot"
     CreationDateTime="2007-04-09T12:24:09">
     - <Study OID="cdisc01">
       - <GlobalVariables>
            <StudyName>CDISC01</StudyName>
            <StudyDescription>CDISC01 Test Study.</StudyDescription>
            <ProtocolName>CDISC01</ProtocolName>
       </GlobalVariables>
       <MetaDataVersion  OID="CDISC.SDTM.3.1.1"
            Name="Study CDISC01, Data Definitions"
            Description="Study CDISC01, Data Definitions"
            def:DefineVersion="1.0.0"
            def:StandardName="CDISC SDTM"
            def:StandardVersion="3.1.1">
  ...
       <ItemGroupDef OID="MH"
            Name="MH" Repeating="Yes" IsReferenceData="No"
            Purpose="Tabulation" def:Label="Medical History"
            def:Structure="One record per medical history event per subject"
            def:DomainKeys="STUDYID, USUBJID, MHCAT, MHTERM, MHSTDTC"
            def:Class="EVENTS"
            def:ArchiveLocationID="Location.AE">
         <ItemRef ItemOID="STUDYID" OrderNumber="1" Mandatory="Yes"
            Role="IDENTIFIER" RoleCodeListOID="RoleCodeList" />
         ...
           <def:leaf ID="Location.MH" xlink:href="mh.xpt">
                <def:title>mh.xpt</def:title>
           </def:leaf>
       </ItemGroupDef>
```

The first line is the XML declaration.

The second line contains a processing instruction to include the XSL style sheet to display the XML file in a human readable form in a browser.

<ODM> is the root element.

<Study>, <GlobalVariables>, <MetaDataVersion>, <StudyName>, <ItemGroupDef>, etc are elements.

<Study> is a child element of the <ODM> element.

FileOID, ODMVersion, etc are attributes of the <ODM> element.

The 4 lines starting with "xmlns:" are namespace declarations.
This excerpt contains no comments or CDATA sections.

To conclude our short introduction to XML the reader is referred to Appendix 1 where an abbreviated version can be found of "XML in 10 points", a short essay by Bert Bos [11]

WELL-FORMED AND VALIDATED

An XML file is said to be *well-formed* if it conforms to the rules of XML syntax. At a very basic level this means:

- A single element, called the root element, contains all other elements in the document (in the define.xml the root element is <ODM>)
- Elements should be properly opened and closed
- Elements do not overlap, e.g. are properly nested

- Attributes are properly quoted
- The document does not contain illegal characters.
  For example, the "<" character has special meaning because it opens a tag. So, if this character is part of the content, it should be should substituted as "<"

A conforming XML parser is not allowed to process an XML document that is not well-formed.

An **XML schema** is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type. This description goes above and beyond the basic syntax constraints imposed by well-formedness of an XML document [12].

The schema defines the allowed elements and attributes, order of the elements, overall structure, etc...

A schema might also describe that the content of a certain element is only valid if it conforms to the ISO 8601 date and time specification. An XML document is *valid*, if it conforms to a specific XML schema.

The following example illustrates the difference between ***well-formed*** and ***validated***. The XML document in Figure 2 is a well-formed XML document, but is obviously not valid with respect to the schema that defines a valid define.xml file.

Figure 2: *A well-formed XML document, but not a valid define.xml document*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<favorites>
    <single genre="doowop">
        <artist>The Sheppards</artist>
        <title>Tragic</title>
        <producer>Bunky Sheppard</producer>
        <writer>Kermit Chandler and O.C. Perkins</writer>
        <label>Apex 7762</label>
        <year>1961</year>
        <position>Did not make pop charts</position>
    </single>
    <single genre="R&amp;amp;B">
        <artist>Joyce Harris</artist>
        <title>No Way Out</title>
        <producer>not credited</producer>
        <writer>Joyce Harris</writer>
        <label>Domino 905</label>
        <year>1961</year>
        <position> Did not make pop charts</position>
    </single>
</favorites>
```

XPATH AND XSL

In order to be able to understand the XML structure of a define.xml file, we need to briefly discuss a few more XML related standards and concepts.

The XML Path Language (**XPath**) gives XML developers a tool for navigating the structure of an XML document. We can simply demonstrate this by 2 examples from Figure 2.

- The XPath location path `/favorites/single/artist` returns all artist elements: "The Sheppards" and "Joyce Harris".
- The XPath location path `/favorites/single/@genre` returns the "doowop" and "R&B" attributes.

The **XPath** standard is part of the **XSL** family of standards.

XSL is the Extensible Style sheet Language [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, XSL, to describe the format of a document, ready for use in a display application (computer screen, cell phone screen, paper).

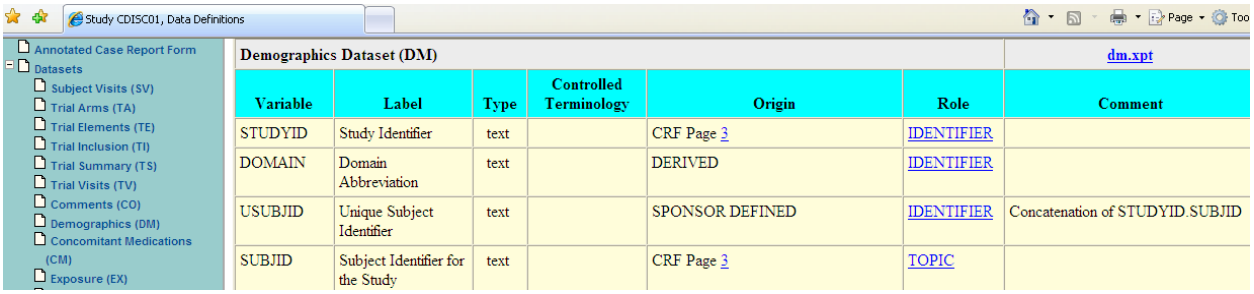
XSL is actually a family of transformation languages which allows one to describe how files encoded in the XML standard are to be formatted or transformed.

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 the define.xml to HTML in such a way that it can be viewed in a browser.

Figure 3: *Define.xml displayed in a browser using an XSL style sheet*



Before we return to XSLT and XPath in more detail, we will first get more familiar with the structure of the define.xml structure.

THE STRUCTURE OF THE DEFINE.XML

The previous section explained the building blocks of XML. This section will specifically describe the structure of a define.xml file that conforms to the Case Report Tabulation Data Definition Specification (CRT-DDS or define.xml) version 1.0 as developed by the CDISC define.xml Team [7].

SCHEMA STRUCTURE

The CRT-DDS (define.xml) standard is based on the CDISC operational model (ODM). The ODM is defined by an XML schema that allows extension [8]. This extension mechanism has been implemented by expressing the ODM schema using two files:

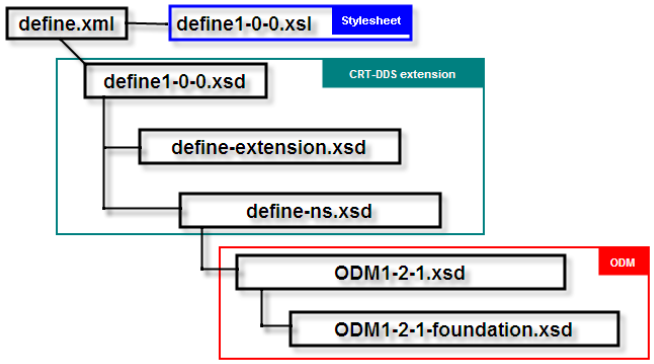
- a *foundation* XML Schema file (ODM1-2-1-foundation.xsd), which defines the elements, attributes and structure of the base ODM schema.
- an *application* XML Schema file (ODM1-2-1.xsd.) which imports the foundation XML Schema and other schema definitions needed by ODM, such as the core W3C XML schema (xml.xsd) and the XML schema that defines the W3C XML Signature standard (xmldsig-core-schema.xsd).

To create the define.xml extension three files have been provided:

- A *namespace* XML schema (define-ns.xsd) that defines the extension namespace and any new elements and attributes.
- An *extension* XML Schema file (define-extension.xsd) defines the location of the extensions within the ODM.
- An *application extension* XML Schema file (define1-0-0.xsd) that will import the extension XML Schema file and, in turn, any files imported in the ODM root schema.

In November 2009 CDISC published a XML Schema Validation for Define.xml white paper [14]. This white paper, created by the CDISC XML Technology team (the former ODM team), provides guidance on validating define.xml documents against the define.xml XML schemas and proposes practices and tools to improve define.xml schema validation.

Figure 4: *The define.xml (CRT-DDS) and the associated style sheet and schemas*



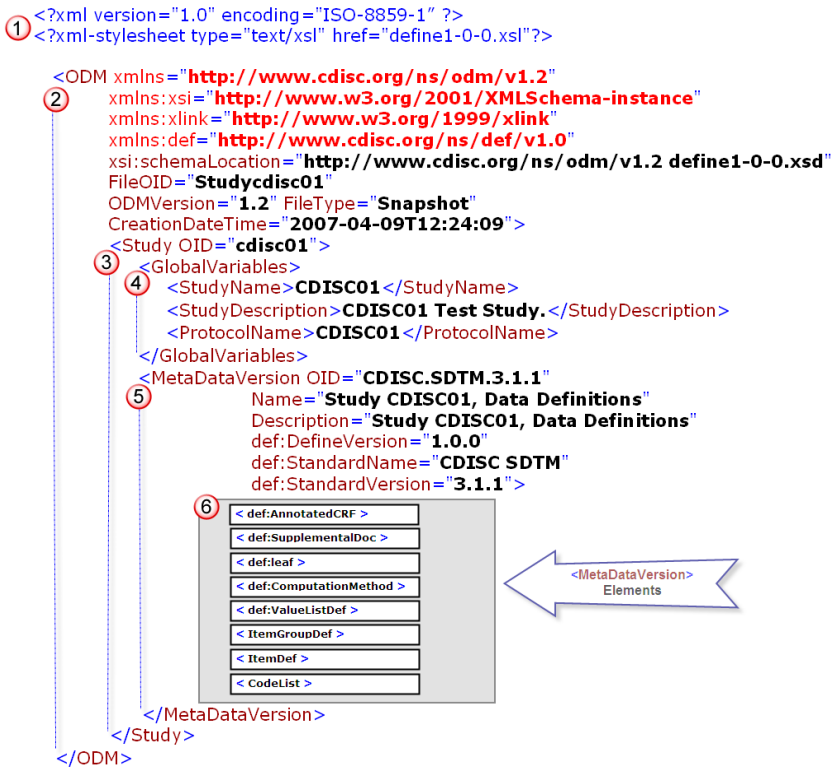
DEFINE.XML BUILDING BLOCKS

In this paragraph we will show the building blocks of the define.xml. As mentioned before, the CRT-DDS standard (define.xml) is an extension of ODM. The ODM model defines many elements and attributes that are optional.

We will only mention required elements and attributes, or elements and attributes that were found in the example define.xml file.

Figure 5 shows a high-level overview of a define.xml file.

Figure 5: *The building blocks of the define.xml (CRT-DDS)*



We will now describe the different parts of the define.xml file.

1 The **header section** of the define.xml is important for the XML processor.

The first line identifies the file as an XML document and specifies the XML version (“1.0”) and the encoding of the document (“ISO-8859-1”). The second line includes a reference to the style sheet (“define1-0-0.xsl”) that can be used by an XSL processor to render the document. In this case the style sheet can be used by the XSL processor in a web browser to render the XML file as HTML for display. As mentioned before, the HTML that gets created by the browser depends on the particular browser application.

2 Following the header section is the **ODM** root element. All other elements in the define.xml file will be contained within the ODM element. The ODM element contains attributes that define the namespaces and the location of the schema that specifies the XML document.

Other required ODM attributes are displayed in Table 1.

Table 1: *ODM required attributes*

Attribute	Description
FileType	Type of transmission. For define.xml the only valid value is “Snapshot”. This means that the document contains only the current state of the data and metadata it describes, and no transactional history.
FileOID	A unique identifier for this file. FileOIDs should be universally unique if at all possible.
CreationDateTime	Date and Time when the define.xml file was last modified (ISO 8601 datetime).
ODMVersion	The version of the ODM standard used. According to the ODM schema, this is an optional attribute. However, a missing ODMVersion should be interpreted as "1.1". Documents based on ODM 1.2 should have ODMVersion="1.2".

3 **Study** is the first element contained in the ODM root element. The Study element collects static structural information about an individual study. It has one attribute “OID”, which is the unique identifier of the Study.

The Study element has two child-elements in the define.xml:

- GlobalVariables - General summary information about the Study.
- MetaDataVersion - Domain and variable definitions within the submission.

4 **GlobalVariables** is a required child element of the **Study** element and contains three required child elements:

- StudyName - Short external name for the study.
- StudyDescription - Free-text description of the study.
- ProtocolName - The sponsor’s internal name for the protocol.

5 The **MetaDataVersion** is a child element of the **Study** element and contains the domain and variable definitions included within a submission. Table 2 lists the MetaDataVersion attributes that are part of the define.xml file. The attributes with a prefix of “def:” are extensions to the ODM schema.

Table 2: *MetaDataVersion required attributes*

Attribute	Description
OID	The unique identifier of the MetaDataVersion
Name	Name of the MetaDataVersion
Description	Further description of the data definition document
def:DefineVersion	The schema version used for the define.xml
def:StandardName	Short name of the MetaDataVersion (e.g. CDISC SDTM)
def:StandardVersion	The version of an external standard to which the data conforms (e.g. 3.1.1)

6 Table 3 lists the MetaDataVersion child elements. The ItemGroupDef and ItemDef elements are required.

Table 3: *MetaDataVersion child elements*

Element	Description
def:AnnotatedCRF	This element can be used to reference an Annotated Case Report Form (CRF), a PDF file that maps the data collection fields used to collect subject data to the corresponding variables or discrete variable values contained within the data sets
def:SupplementalDoc	This element can be used to reference a PDF file with supplemental data definition information. A reason for this document can be the need for further explanations or descriptions of variables contained within the data sets.
def:leaf	This element has to be present if either the def:AnnotatedCRF and/or the def:SupplementalDoc are provided. The def:leaf element will then contain the actual location of the PDF file relative to the define.xml
def:ComputationMethod	This element is available for every unique computational algorithm that is referenced by variable metadata or variable value-level metadata. It contains the method name and the computation rule in a kind of pseudo code.
def:ValueListDef	This element provides additional value-level metadata for certain variables that are part of a normalized data structure. For example, the Vital Signs domain has a measurement parameter (VSTESTCD) that stores the name of a measurement (height, weight, systolic blood pressure, …). A corresponding value list will then describe each unique value of the measurement (“HEIGHT”, “WEIGHT”, “SYSBP”, …)
ItemGroupDef	For every data set in the submission there is an ItemGroupDef element describing data set metadata (label, structure, keys, variables, location of transport file, …)
ItemDef	For every variable referenced in either def:ValueListDef or ItemGroupDef there will be an ItemDef element. This element will describe properties like name, label, length, computation method, range or code list restrictions, and several other properties.
CodeList	The CodeList element defines a discrete set of permitted values for an item. The definition can be an explicit list of values or a reference to an externally defined code list.

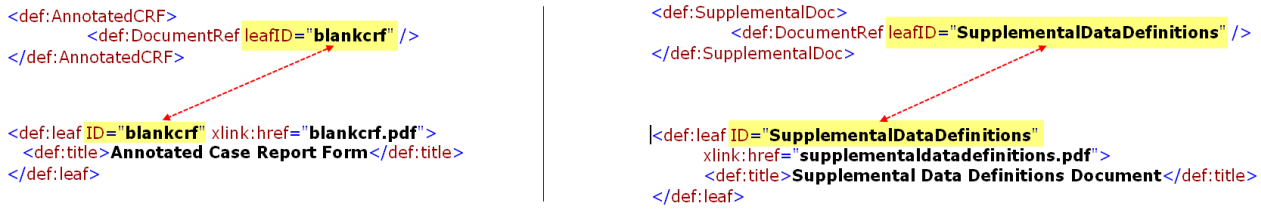
METADATAVERSION ELEMENT DETAILS

In this paragraph we will dive deeper into the sub-elements of the **MetaDataVersion** element. We will see how the different elements relate to each other.

def:AnnotatedCRF, def:SupplementalDoc and def:leaf

As mentioned before, these elements can be used to reference PDF files. Figure 6 shows the relation between these elements. The **def:DocumentRef/@leafID** must match the corresponding **def:leaf/@ID**.

Figure 6: *Referencing external PDF files*



ItemGroupDef

For every data set in the submission there will be an ItemGroupDef element with domain-level metadata. Table 4 lists the ItemGroupDef attributes.

Table 4: *ItemGroupDef attributes*

Attribute	Status	Description
OID	Required	The unique identifier of the domain.
Name	Required	The file name of the data set or data domain name (e.g., “DM” for Demographics)
Repeating	Required	“Yes” for domains with more than one record per subject whereas, “No” for domains with 1 record per subject.
IsReferenceData	Optional	“Yes” for domains that contain reference data only, no subject-level data. “No” indicates subject-level data. Absence of this attribute indicates subject-level data.
SASDatasetName[1]	Optional	Name of SAS data set.
Purpose	Optional	Purpose for the data set (e.g., “Tabulation”, “Analysis”).
def:Label	Required	Brief description of the data domain.
def:Structure	Optional	Data domain structure (e.g. “One record per subject per visit”).
def:DomainKeys	Optional	Comma-separated text string that contains the data set variables that uniquely identify a data record.
def:Class	Optional	General class of the domain as defined in the SDTM model (e.g., “Events”, “Interventions”, “Findings”, “Special Purpose”, “Trial Design”, ...)
def:ArchiveLocationID	Required	Reference to the def:leaf element that contains a link to the location of the data set.

ItemGroupDef elements have:

- One ItemRef child element per variable in the data set
- Exactly one def:leaf element that contains the XLink information that is referenced by the def:ArchiveLocationID attribute.

The relation between the **ItemGroupDef/@def:ArchiveLocationID** and the **def:leaf/@ID** is illustrated in Figure 7.

[1] SASDatasetName is an optional attribute that is part of the ODM foundation. This means that it is a valid attribute in the define.xml. This attribute is not mentioned in the define.xml specification.

Figure 7: *Referencing external data sets*

```
<ItemGroupDef OID="MH"
      Name="MH"
      Repeating="Yes"
      IsReferenceData="No"
      SASDatasetName="MH"
      Purpose="Tabulation"
      def:Label="Medical History"
      def:Structure="One record per medical history event per subject"
      def:DomainKeys="STUDYID, USUBJID, MHCAT, MHTERM, MHSTDTC"
      def:Class="EVENTS"
      def:ArchiveLocationID="Location.MH">

      <ItemRef ... />
       ...
      <ItemRef ... />

      <def:leaf ID="Location.MH" xlink:href="mh.xpt">
            <def:title>mh.xpt</def:title>
      </def:leaf>
</ItemGroupDef>
```

ItemRef and ItemDef

For every variable in a data set in the submission there will be an ItemRef and an ItemDef element with variable-level metadata. Table 5 lists the ItemRef attributes. Table 6 lists ItemDef attributes. ItemDef elements can have an optional associated CodeListRef or def:ValueListRef child element.

def:ComputationMethod

The def:ComputationMethod element has one attribute (OID) and contains the details about computational algorithms used to derive or impute variable values.

Figure 8: *Example of a ComputationMethod*

```
<def:ComputationMethod OID=" COMPMETHOD.QTCB">
   QTcB = QT interval / square root of (60 / heart rate)
</def:ComputationMethod>
```

Table 5: *ItemRef attributes*

Attribute	Status	Description
ItemOID	Required	The unique identifier of the variable.
OrderNumber	Optional	Provide an ordering on the ItemRefs (within the containing ItemGroupDef) for use whenever a list of ItemRefs is presented to a user.
Mandatory	Required	Indicates that the clinical data for an instance of the containing item group would be incomplete without an instance of this type of item. Variables that have an SDTM “Core” attribute given as "Required" should have Mandatory="Yes" in the define.xml. SDTM variables that have a “Core” attribute as "Expected" or "Permissible" should have Mandatory="No" in the define.xml.
Role	Optional	Variable classification (e.g., “Identifier”, “Topic”, .”Timing Variable”, ...). Values are interpreted as codes from CodeList as specified by @RoleCodeListOID
RoleCodeListOID	Optional	The identifier of the corresponding Role Code List, which defines the full set roles from which the Role attribute values are to be taken.

Table 6: *ItemDef attributes*

Attribute	Status	Description
OID	Required	The unique identifier of the variable.
Name	Required	Variable name.
DataType	Required	Type of the data (e.g., text, integer, float, ...)
Length	Conditional	The variable length.
SignificantDigits	Conditional	The number of decimal digits.
SASFieldName[2]	Optional	
Origin	Optional	Indicator of the origin of the variable (e.g., CRF page number, derived, or a reference to other variable(s).
Comment	Required	Further information regarding variable definition, usage, etc.
Def:Label	Required	Variable label.
Def:DisplayFormat	Optional	Display format for numeric variables (e.g., 8.2)
Def:ComputationMethodOID	Optional	Unique identifier of the corresponding computation method.

CodeList and def:ValueListDef

Many variables in a submission have a discrete list of valid values or controlled terms associated with them. The CodeList element defines the controlled terminology. For variables whose values are restricted to a list of values, the corresponding ItemDef element has to include a CodeListRef element that references a CodeList element. The CodeListOID attribute of the CodeListRef element should match the OID attribute of the CodeList element.

The def:ValueListDef element provides additional value-level metadata for certain variables that are part of a normalized data structure. For example, the Vital Signs domain has a measurement parameter (VSTESTCD) that stores the name of a measurement (height, weight, systolic blood pressure, …). A corresponding value list will then describe each unique value of the measurement (“HEIGHT”, “WEIGHT”, “SYSBP”, …). For these variables, the corresponding ItemDef element has to include a def:ValueListRef element that references a def:ValueListDef element. The **def:ValueListOID** attribute of the **def:ValueListRef** element should match the **OID** attribute of the **def:ValueListDef** element.

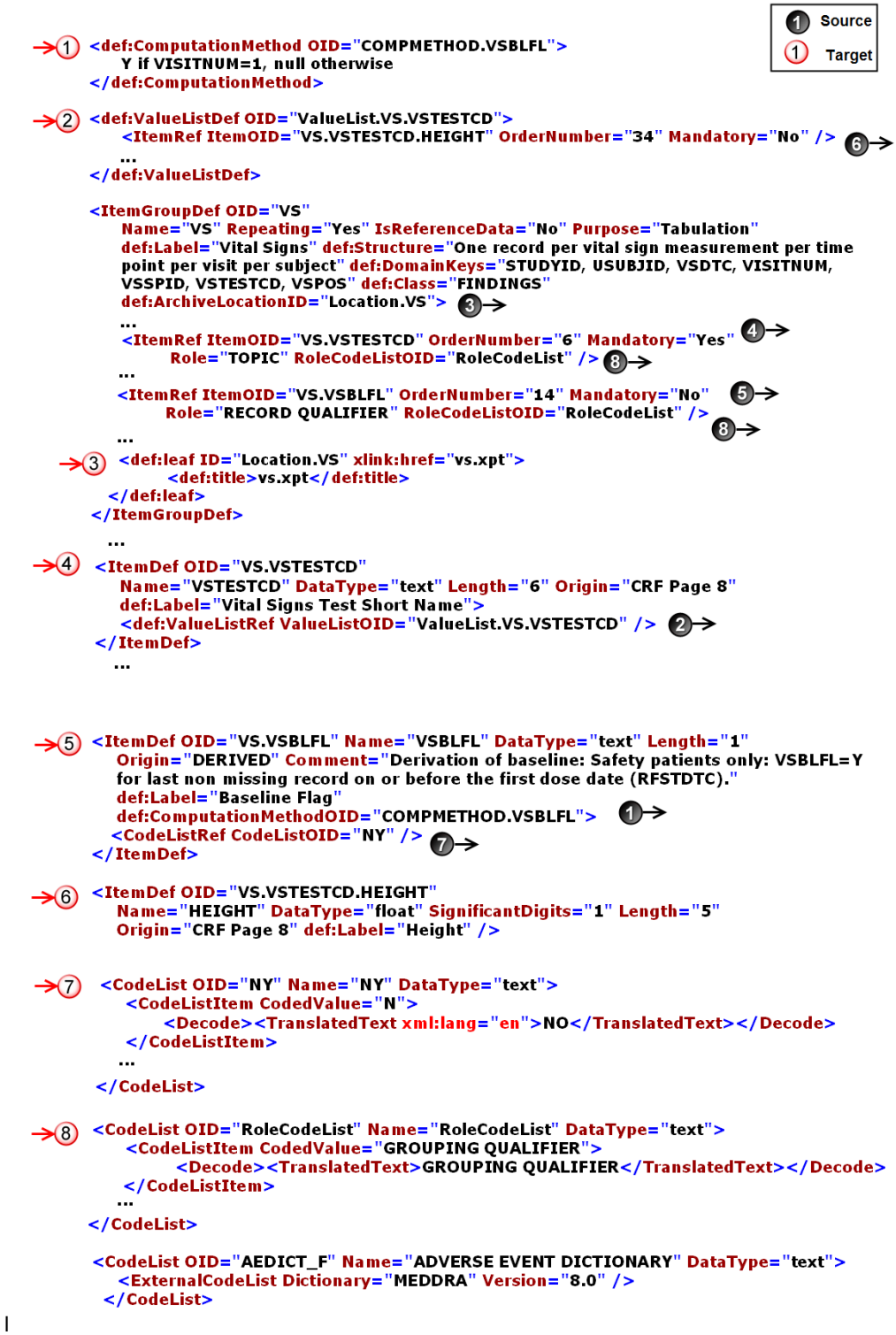
Figure 9 illustrates several concepts that are related to ItemRef, ItemDef, def:ComputationMethod, CodeList and def:ValueListDef elements.

The black numbers on the right side represent the source of a relation and a red numbers on the left side represent the target of a relation. In every relation there has to be a correspondence between the identifier of the source and the target.

For example: In relation 4: source identifier = **ItemRef/@ItemOID**, target identifier = **ItemDef/@OID** and the both have the value “VS.VSTESTCD”;

[2] SASFieldName is optional an attribute that is part of the ODM foundation. This means that it is a valid attribute in the define.xml. This attribute is not mentioned in the define.xml specification.

Figure 9: Relations within a define.xml



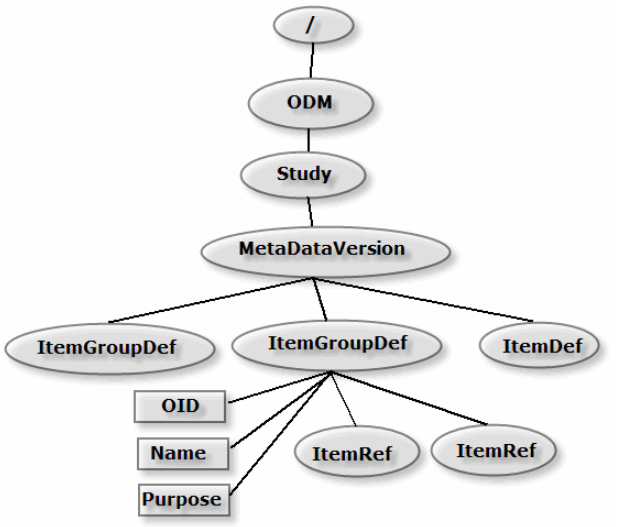
XPATH

XPath sees an XML document as a tree with branches called nodes. The nodes in this tree have family relationships: parent-child, ancestor-descendant, sibling, and so forth.

Conceptually, the XPath data model describes an XML document as having seven possible node types:

- Root
- Element
- Attribute
- Text
- Namespace
- Comment
- Processing instruction

Figure 10: define.xml as a tree



In Figure 10 we see part of a define.xml as a tree with nodes. We only show element and attribute nodes. **ODM** is the root element and is the *parent* of the **Study** element. The **ItemGroupDef** and **ItemDef** elements are *siblings*. The **OID**, **Name** and **Purpose** attributes and the **ItemRef** elements are *children* of The **ItemGroupDef** element. **Study**, **MetaDataVersion**, **ItemGroupDef**, **ItemDef** and **ItemRef** are all *descendants* of the ODM element. The **ODM** element is an *ancestor* of the **ItemRef** elements, or the **ItemDefs** elements.

XPath allows us to access any node from any other node simply by knowing the relationship between the two.

At the foundation of the XPath language is the ability to use *location paths* to refer to a node or nodeset. A node is a piece of the XML document (such as an element, an attribute, or text content).

A location path identifies a set of nodes in a document. This set may be empty, may contain a single node, or may contain several nodes. These can be element nodes, attribute nodes, namespace nodes, text nodes, comment nodes, processing-instruction nodes, root nodes, or any combination of these. A location path is built out of successive *location steps*. Each location step is evaluated relative to a particular node in the document called the *context node* [15].

There are two kinds of location paths: relative location paths and absolute location paths.

A *relative location* path consists of a sequence of location steps separated by a forward slash (/). Each step selects a node or nodeset relative to the current node. Then, each node in that set is used as the current node for the following step, and so on.

An *absolute location* path starts with a forward slash, optionally followed by a relative location path.

Examples:

- odm:Decode/odm:TranslatedText/text()
- /odm:ODM/odm:Study/odm:MetaDataVersion/ItemGroupDef

The SAS XML Mapper application also uses XPath location paths to tell the SAS XML engine where to find content from the XML elements and attribute content to create rows and columns in SAS data sets [17].

Figure 11 shows part of an XMLMAP. It specifies that the data set **ItemDef** has a variable named **ItemDef_Name**, whose content is defined by the XPath specification **/ODM/Study/MetaDataVersion/ItemDef/@Name**.

Figure 11: *XMLMAP example*

```
- <TABLE name="ItemDef">
    <TABLE-DESCRIPTION>ItemDef</TABLE-DESCRIPTION>
    <TABLE-PATH syntax="XPath">/ODM/Study/MetaDataVersion/ItemDef</TABLE-PATH>
  + <COLUMN name="Study_OID" retain="YES">
  + <COLUMN name="MetaDataVersion_OID" retain="YES">
  + <COLUMN name="ItemDef_OID">
  - <COLUMN name="ItemDef_Name">
      <PATH syntax="XPath">/ODM/Study/MetaDataVersion/ItemDef/@Name</PATH>
      <DESCRIPTION>Name</DESCRIPTION>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>40</LENGTH>
    </COLUMN>
  ...
  + <COLUMN name="CodeListRef_CodeListOID">
  + <COLUMN name="ValueListRef_ValueListOID">
  </TABLE>
```

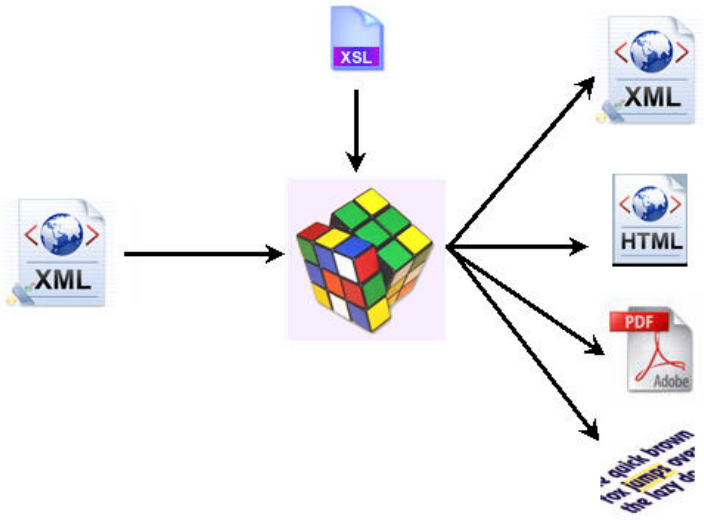
For a more indepth discussion of XPath, the reader is referred to [15] or [16].

XSLT

Extensible Stylesheet Language Transformations (XSLT) is a language that 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*" [16]. In [16]. Michael Kay compares XSLT with the famous Rubik's cube. XSLT can convert attributes to elements, or elements to attributes.

Figure 12: *Transforming with XSLT*



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.

A very basic (minimal) XSLT stylesheet is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

Surprisingly, when applied to an XML file, this stylesheet would just strip all markup and reproduce the content of the input document.

USING XSLT STYLESHEETS WITH SAS

SAS has a new procedure, PROC XSL, that is not documented yet. The following example shows how to use it:

```
FILENAME XMLfile "c:\data\xml\define.xml";
FILENAME XSLfile "c:\data\xsl\define_format.xsl";
FILENAME Outfile "c:data\ProcFormat.sas";

PROC XSL IN=XMLfile XSL=XSLfile OUT=Outfile;
RUN;
```

Another possibility to run XSLT stylesheets in SAS is to download the Java library from the Apache Xalan project (http://xml.apache.org/xalan-j/) and then run the transformation in SAS with an X-command:

```
x "java -jar xalan.jar -IN define.xml -XSL define_format.xsl -OUT ProcFormat.sas";
```

Using Xalan would also allow to pass parameters to the stylesheet, which is something that PROC XSL does not allow at the moment:

```
x "java -jar xalan.jar -IN define.xml -XSL define_format.xsl -OUT ProcFormat.sas -PARAM
library MyLib";
```

EXAMPLES OF XSLT STYLESHEETS

The appendix contains 2 examples of stylesheets that can be used with the define.xml together with the output that gets created by applying the stylesheets.

CONCLUSION

By using XSLT we can transform the define.xml into executable SAS code. This will enable the use of the define.xml not only to document SAS data sets, but also to use the define.xml as a data specification or to create executable SAS programs.

REFERENCES

1. 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)".
"Final Guidance for Industry: Providing Regulatory Submissions in Electronic Format--Human Pharmaceutical Applications and Related Submissions Using the eCTD Specifications". Revision 2, June 2008.
(http://www.fda.gov/downloads/Drugs/GuidanceComplianceRegulatoryInformation/Guidances/UCM072349.pdf)
2. U.S. Department of Health and Human Services Food and Drug Administration Center for Drug Evaluation and Research (CDER). Study Data Specifications, Version 1.5.1, January 2010
(http://www.fda.gov/downloads/Drugs/DevelopmentApprovalProcess/FormsSubmissionRequirements/ElectronicSubmissions/UCM199759.pdf)
3. CDISC Study Data Tabulation Model, Version 1.1, April 28, 2005
(http://www.cdisc.org/content1605)
4. CDISC Study Data Tabulation Model Implementation Guide: Human Clinical Trials, Version 3.1.1, August 26, 2005
(http://www.cdisc.org/content1605)
5. CDISC Study Data Tabulation Model, Version 1.2, November 12, 2008
(http://www.cdisc.org/sdtm)
6. CDISC Study Data Tabulation Model Implementation Guide: Human Clinical Trials, Version 3.1.2, November 12, 2008 (http://www.cdisc.org/sdtm)
7. Case Report Tabulation Data Definition Specification (define.xml), Version 1.0, February 9, 2005
(http://www.cdisc.org/define-xml)
8. CDISC Operational Data Model (ODM), Version 1.2.1, January, 2005
(http://www.cdisc.org/odm)
9. Extensible Markup Language (XML) 1.0, Fourth Edition, August 16, 2006
(http://www.w3.org/TR/2006/REC-xml-20060816)
10. Eric T. Ray, 2003, Learning XML, *Creating Self-Describing Data*. 2nd Edition, (O'Reilly and Associates)
11. Bert Bos, 2000, XML in 10 points.
(http://www.w3.org/XML/1999/XML-in-10-points)
12. Eric van der Vlist, 2002, XML Schema, *The W3C's Object-Oriented Descriptions for XML* (O'Reilly and Associates)
13. Dough Tidwell, 2001, XSLT, *Mastering XML Transformations*. (O'Reilly and Associates)
14. XML Schema Validation for Define.xml, Version 1.0, November 30, 2009
(http://www.cdisc.org/define-xml)
15. Eliotte Rusty Harold & W. Scott Means, 2004, XML in a Nutshell, *A Desktop Quick Reference* (O'Reilly and Associates)
16. Michael Kay, 2008, XSLT 2.0 and XPath 2.0, 4th Edition, *Programmer's Reference* (Wrox)
17. Lex Jansen (2008). Using the SAS XML Mapper and SAS ODS to create a PDF representation of the define.xml (that can be printed). *Proceedings of the 4th Pharmaceutical Users Software Exchange (PhUSE 2008, Manchester, UK)* (http://www.lexjansen.com/phuse)

CONTACT INFORMATION

Lex Jansen,
Senior Consultant, Clinical Data Strategies
Octagon Research Solutions, Inc.
585 East Swedesford Road, Suite 200
Wayne, PA 19087
Email: ljansen@octagonresearch.com

This paper can be found at http://www.lexjansen.com together with links to more than 10,000 other papers that were presented at SAS usergroups.

APPENDIX

Appendix 1 **XML in 10 Points, an essay by Bert Bos**
(full version at http://www.w3.org/XML/1999/XML-in-10-points)

XML is for structuring data

... XML is a set of rules (you may also think of them as guidelines or conventions) for designing text formats that let you structure your data ... XML makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous. ...

XML looks a bit like HTML

Like HTML, XML makes use of *tags* (words bracketed by '<' and '>') and *attributes* (of the form name="value"). ... While HTML specifies what each tag and attribute means, and often how the text between them will look in a browser, XML uses the tags only to delimit pieces of data ...

XML is text, but isn't meant to be read

... Like HTML, XML files are text files that people shouldn't have to read, but may when the need arises. Compared to HTML, the rules for XML files allow fewer variations. A forgotten tag, or an attribute without quotes makes an XML file unusable ...

XML is verbose by design

Since XML is a text format and it uses tags to delimit the data, XML files are nearly always larger than comparable binary formats. That was a conscious decision by the designers of
XML. ...

XML is a family of technologies

XML 1.0 is the specification that defines what "tags" and "attributes" are. Beyond XML 1.0, "the XML family" is a growing set of modules that offer useful services to accomplish important and frequently demanded tasks. ... (Xlink, Xpointer, CSS, XSL, XSLT, DOM, Schema, ...)

XML is new, but not that new

... The designers of XML simply took the best parts of SGML, guided by the experience with HTML, and produced something that is no less powerful than SGML, and vastly more regular and simple to use. ...

XML leads HTML to XHTML

There is an important XML application that is a document format: W3C's XHTML, the successor to HTML. XHTML has many of the same elements as HTML. The syntax has been changed slightly to conform to the rules of XML. ...

XML is modular

XML allows you to define a new document format by combining and reusing other formats. ... To eliminate name confusion when combining formats, XML provides a namespace mechanism. ...

XML is the basis for RDF and the Semantic Web

RDF is an XML text format that supports resource description and metadata applications, such as music playlists, photo collections, and bibliographies. ...

XML is license-free, platform-independent and well-supported

By choosing XML as the basis for a project, you gain access to a large and growing community of tools (one of which may already do what you need!) and engineers experienced in the technology. Opting for XML is a bit like choosing SQL for databases: you still have to build your own database and your own programs and procedures that manipulate it, but there are many tools available and many people who can help you. And since XML is license-free, you can build your own software around it without paying anybody anything. The large and growing support means that you are also not tied to a single vendor. *XML isn't always the best solution, but it is always worth considering.*

APPENDIX

Appendix 2-A An XSLT stylesheet to create a PROC FORMAT from the define.xml

```
<?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/v1.2"
                xmlns:def="http://www.cdisc.org/ns/def/v1.0"
                xmlns:xlink="http://www.w3c.org/1999/xlink">

<xsl:output method="text" />
<xsl:variable name="lf" select="'&#13;&#10;'"/>  <!—define linefeed -->

<xsl:template match="/">
  <xsl:text>PROC FORMAT LIBRARY=work
  <xsl:text>;</xsl:text>
  <xsl:value-of select="$lf"/>
     <xsl:call-template name="CodeLists"/>
   <xsl:text>RUN;</xsl:text>
</xsl:template>

<!--  Template: CodeLists
      Purpose:  Process all CodeLists within a define file. -->
<xsl:template name="CodeLists">

  <xsl:for-each select="/odm:ODM/odm:Study/odm:MetaDataVersion/odm:CodeList">
      <xsl:sort select="@Name"/>
      <xsl:variable name="DataType" select="@DataType"/>
      <xsl:if test="count(odm:CodeListItem) > 0">
        <xsl:text>  VALUE </xsl:text>
        <xsl:choose>
          <xsl:when test="@SASFormatName">
             <xsl:value-of select="@SASFormatName"/>
          </xsl:when>
          <xsl:otherwise>

              <xsl:if test="$DataType='text'">$</xsl:if>
              <xsl:value-of select="substring(translate(@Name, ' 0123456789',''),1,7)"/>

          </xsl:otherwise>
        </xsl:choose>

        <xsl:value-of select="$lf"/>

        <!-- Process all CodeListItems within a CodeList. -->

        <xsl:for-each select="./odm:CodeListItem">
          <xsl:sort select="concat(@def:Rank, ' ', @CodedValue)"/>
          <xsl:text>    </xsl:text>
          <xsl:if test="$DataType='text'">"</xsl:if>
          <xsl:value-of select="@CodedValue"/>
          <xsl:if test="$DataType='text'">"</xsl:if>
          <xsl:text> = "</xsl:text>
          <xsl:value-of select="normalize-space(odm:Decode/odm:TranslatedText/text())"/>
          <xsl:text>"</xsl:text>
          <xsl:value-of select="$lf"/>
        </xsl:for-each>

        <xsl:text>    ;</xsl:text>
        <xsl:value-of select="$lf"/>

      </xsl:if>
   </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

APPENDIX

Appendix 2-B Output from an XLST stylesheet to create a PROC FORMAT from the define.xml

```
PROC FORMAT LIBRARY=work;
  VALUE $FRAME
    "S" = "Small"
    "M" = "Medium"
    "L" = "Large"
    "XL" = "Extra large"
    ;
  VALUE $SEX
    "F" = "FEMALE"
    "M" = "MALE"
    "U" = "UNKNOWN"
    ;
  VALUE SMKCLAS
    1 = "NEVER SMOKED"
    2 = "SMOKER"
    3 = "EX SMOKER"
    ;
  VALUE $YESNO
    "N" = "NO"
    "U" = "NOT APPLICABLE"
    "Y" = "YES"
    ;
  VALUE $YESNOUN
    "N" = "NO"
    "NA" = "NOT APPLICABLE"
    "U" = "UNKNOWN"
    "Y" = "YES"
    ;
RUN;
```

APPENDIX

Appendix 3-B An XSLT stylesheet to create data set templates

```
<?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/v1.2"
                xmlns:def="http://www.cdisc.org/ns/def/v1.0"
                xmlns:xlink="http://www.w3c.org/1999/xlink">

<xsl:output method="text" indent="no"/>   <!—define linefeed -->
<xsl:variable name="lf" select="'&#13;&#10;'"/>

<xsl:template match="/">
  <xsl:text>PROC SQL;</xsl:text>
  <xsl:value-of select="$lf"/>
  <xsl:call-template name="ItemGroupDefs"/>
  <xsl:text>QUIT;</xsl:text>
</xsl:template>

<!-- Template: ItemGroupDefs
     Purpose:  Process all ItemGroups within a define file. -->
<xsl:template name="ItemGroupDefs">

  <xsl:for-each select="/odm:ODM/odm:Study/odm:MetaDataVersion/odm:ItemGroupDef">
    <xsl:variable name="itemOID" select="@ItemOID"/>
    <xsl:text>  CREATE TABLE </xsl:text>

    <xsl:choose>
      <xsl:when test="@SASDatasetName">
         <xsl:value-of select="@SASDatasetName"/>
      </xsl:when>
      <xsl:otherwise>
          <xsl:value-of select="substring(translate(@Name, ' ',''),1,8)"/>
      </xsl:otherwise>
    </xsl:choose>

    <xsl:text />(LABEL = "<xsl:value-of select="@def:Label"/><xsl:text>&quot;) (</xsl:text>
    <xsl:value-of select="$lf"/>
    <xsl:call-template name="ItemRefs"/>
    <xsl:text>    );</xsl:text>
    <xsl:value-of select="$lf"/>
  </xsl:for-each>

 </xsl:template>


<!--  Template: ItemRefs
      Purpose:  Process all ItemRefs within an ItemGroup. -->
 <xsl:template name="ItemRefs">

    <xsl:for-each select="./odm:ItemRef">
      <xsl:sort select="@OrderNumber" data-type="number" />
      <xsl:variable name="itemDefOid" select="@ItemOID"/>
      <xsl:variable name="itemDef" select="../../odm:ItemDef[@OID=$itemDefOid]"/>
      <xsl:if test="$itemDef">
      <xsl:text>    </xsl:text>

        <xsl:choose>
         <xsl:when test="$itemDef/@SASFieldName">
             <xsl:value-of select="$itemDef/@SASFieldName"/>
         </xsl:when>
         <xsl:otherwise>
             <xsl:value-of select="substring(translate($itemDef/@Name, ' ',''),1,8)"/>
         </xsl:otherwise>
       </xsl:choose>

      <xsl:text> </xsl:text>

      <xsl:choose>
        <xsl:when test="$itemDef/@DataType='text' or
                        $itemDef/@DataType='datetime' or
                        $itemDef/@DataType='date' or
                        $itemDef/@DataType='time'">
           <xsl:text>CHAR</xsl:text><xsl:text>(</xsl:text><xsl:value-of
select="$itemDef/@Length"/><xsl:text>)</xsl:text>
```

```
        </xsl:when>
        <xsl:when test="$itemDef/@DataType='integer' or
                        $itemDef/@DataType='float'">
           <xsl:text>NUMERIC</xsl:text>
        </xsl:when>
      </xsl:choose>

      <xsl:if test="$itemDef/@def:Label">
         <xsl:text> LABEL=&quot;</xsl:text><xsl:value-of select="normalize-
space($itemDef/@def:Label)"/><xsl:text>&quot;</xsl:text>
      </xsl:if>

      <xsl:if test="position() != last()">
        <xsl:text>,</xsl:text>
      </xsl:if>

      <xsl:value-of select="$lf"/>
      </xsl:if>
 </xsl:for-each>
 </xsl:template>


</xsl:stylesheet>
```

APPENDIX

Appendix 3-B Output from an XLST stylesheet to create a data set templates from the define.xml

```
PROC SQL;
  CREATE TABLE DM(LABEL = "Demographics") (
    STUDYID CHAR(8) LABEL="STUDY IDENTIFIER",
    DOMAIN CHAR(2) LABEL="DOMAIN ABBREVIATION",
    USUBJID CHAR(60) LABEL="UNIQUE SUBJECT IDENTIFIER",
    SUBJID CHAR(60) LABEL="SUBJECT IDENTIFIER",
    RFSTDTC CHAR(26) LABEL="SUBJECT REFERENCE START DATE/TIME",
    RFENDTC CHAR(26) LABEL="SUBJECT REFERENCE END DATE/TIME",
    SITEID CHAR(161) LABEL="STUDY SITE IDENTIFIER",
    INVID CHAR(20) LABEL="INVESTIGATOR IDENTIFIER",
    BRTHDTC CHAR(26) LABEL="DATE/TIME OF BIRTH",
    AGE NUMERIC LABEL="AGE IN AGEU AT REFERENCE DATE/TIME",
    AGEU CHAR(13) LABEL="AGE UNITS",
    SEX CHAR(80) LABEL="SEX",
    RACE CHAR(8) LABEL="RACE",
    ARMCD CHAR(80) LABEL="ARM CODE",
    ARM CHAR(80) LABEL="DESCRIPTION OF ARM",
    COUNTRY CHAR(80) LABEL="COUNTRY",
    DMDTC CHAR(26) LABEL="DATE/TIME OF COLLECTION",
    DMDY NUMERIC LABEL="STUDY DAY OF COLLECTION"
    );
  CREATE TABLE VS(LABEL = "Vital Signs") (
    STUDYID CHAR(8) LABEL="STUDY IDENTIFIER",
    DOMAIN CHAR(2) LABEL="DOMAIN ABBREVIATION",
    USUBJID CHAR(60) LABEL="UNIQUE SUBJECT IDENTIFIER",
    VSSEQ NUMERIC LABEL="SEQUENCE NUMBER",
    VSTESTCD CHAR(8) LABEL="VITAL SIGNS TEST SHORT NAME",
    VSTEST CHAR(40) LABEL="VITAL SIGNS TEST NAME",
    VSPOS CHAR(8) LABEL="VITAL SIGNS POSITION OF SUBJECT",
    VSORRES CHAR(8) LABEL="RESULT OR FINDING IN ORIGINAL UNITS",
    VSORRESU CHAR(15) LABEL="ORIGINAL UNITS",
    VSSTRESC CHAR(20) LABEL="CHARACTER RESULT/FINDING IN STD FORMAT",
    VSSTRESN NUMERIC LABEL="NUMERIC RESULT/FINDING IN STANDARD UNITS",
    VSSTRESU CHAR(15) LABEL="STANDARD UNITS",
    VSSTAT CHAR(8) LABEL="VITALS STATUS",
    VSLOC CHAR(1) LABEL="LOCATION OF VITAL SIGNS MEASUREMENT",
    VISITNUM NUMERIC LABEL="VISIT NUMBER",
    VISIT CHAR(30) LABEL="VISIT NAME",
    VSDTC CHAR(26) LABEL="DATE/TIME OF MEASUREMENTS",
    VSDY NUMERIC LABEL="STUDY DAY OF VITAL SIGNS"
    );
  CREATE TABLE CO(LABEL = "Comments") (
    STUDYID CHAR(8) LABEL="STUDY IDENTIFIER",
    DOMAIN CHAR(2) LABEL="DOMAIN ABBREVIATION",
    RDOMAIN CHAR(2) LABEL="RELATED DOMAIN ABBREVIATION",
    USUBJID CHAR(60) LABEL="UNIQUE SUBJECT IDENTIFIER",
    COSEQ NUMERIC LABEL="SEQUENCE NUMBER",
    IDVAR CHAR(8) LABEL="IDENTIFIER VARIABLE NAME",
    IDVARVAL CHAR(8) LABEL="IDENTIFIER VARIABLE VALUE",
    COREF CHAR(1) LABEL="COMMENT REFERENCE",
    CODTC CHAR(19) LABEL="DATE/TIME OF COMMENT",
    COVAL CHAR(200) LABEL="COMMENT",
    COEVAL CHAR(1) LABEL="EVALUATOR"
    );
QUIT;
```