Paper CD10

Converting the define.xml to a Relational Database to Enable Printing and Validation

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.

CDISC has made a basic style sheet available on their website to display the define.xml in a human readable form in a web browser. However, this style sheet is not suitable to generate a paper-based presentation of the define.xml. During the first CDISC SDTM / ADaM Pilot Project, the inability to print the define.xml in a useful way was identified by the regulatory review team as a major issue. It was emphasized that the ability to print the define.xml would be essential for the future use of XML files.

When using the define.xml as a mechanism to describe the metadata of submitted data sets, it is also essential that the define.xml accurately describes the format and content of these data sets. When assessing the correctness and quality of the define.xml file, we can distinguish various aspects:

- Is the define.xml well-formed
- Is the define.xml valid in relation to the schema
- Is the define.xml conforming to the rules as described in the specification
- Is the define.xml consistent with the submitted SAS data sets
- Is the define.xml consistent with the metadata that describes submission standard (SDTM)

Traditional XML based tools typically are able to address some of these aspects, i.e. well-formedness and schema validation. Rule based XML validation languages, like Schematron, are able to check the conformance of the define.xml with regards to rules that can be distilled from the specification.
However, when part of the artifacts that are needed to perform a complete quality assessment of the define.xml, are not XML based (i.e. SDTM metadata, SAS datasets), it is clear that we need to complement our XML tool set.

This presentation addresses both the printing issue and the quality issue. It illustrates how the define.xml can be transformed into relational SAS data sets by using SAS XML Mapper technology. Once the data is available in SAS data sets it can be rendered as a PDF file for printing with the use of basic SAS technology.
The relational SAS datasets can also be used to validate the metadata contained in the define.xml file against the metadata in the SAS transport files and the different types of SDTM metadata.

INTRODUCTION

The FDA issued the Final Guidance on Electronic Submissions using the eCTD specifications [1] in April 2006.
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 [3], 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) [4]. 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) [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.

In July 2007 The CDISC Submission Data Standards (SDS) Metadata Team released a draft version of the Metadata Submission Guidelines, Appendix to the Study Data Tabulation Model Implementation Guide 3.1.1 for review [9]. This release included a sample electronic submission that contains examples of CRF annotations, metadata associated with the submission domains, SDTM domains, and an example of a define.xml file.

WHY CONVERT THE DEFINE.XML INTO RELATIONAL SAS DATA SETS?

In this section we will present business cases that justify the effort of converting the metadata that is contained in a define.xml file into relational SAS data sets.

PRINTING THE DEFINE.XML

In January 2008 CDISC published a report that summarizes the work, experiences and findings of the first CDISC SDTM / ADaM Pilot Project [10]. The objective of the pilot project was to test how well the submission of CDISC-adherent data sets and associated metadata met the needs and the expectations of both medical and statistical FDA reviewers.

This pilot report mentions the following issue:

> *"A major issue identified by the regulatory review team was the difficulty in printing the Define file. The style sheet used in the pilot submission package was developed with the primary target of web browser rendering, which is not readily suited to printing. Reviewers who attempted to print the Define file found that the file did not fit on portrait pages, that page breaks were not clean, and that printing only a portion of the file was difficult. Opening the document in another application (e.g., Microsoft Word) provided a work-around, but was not an option that was user friendly or efficient."*
>
> *…*
>
> *As stated previously, style sheets used for viewing of the Define file do not facilitate printing the file in such a way as to produce a reasonably formatted document. Solutions to allow both easy viewing and printing of Define files have not been identified. This problem could be viewed as an implementation issue that sponsors will need to handle, after discussing the issue with their FDA reviewers. For example, a sponsor might choose to provide two versions of the style sheet – XML for viewing and PDF for printing. Ideally, a reminder of the issue would be included somewhere in the CRT-DDS guidance (e.g., a note that consideration be given to how the sponsor will respond to a request from reviewers for a print-friendly version of the style sheet). It should be noted that the regulatory review team for the pilot project emphasized that the ability to print the document would be essential for the future use of XML files.* [10]

Once the metadata that is contained in the define.xml is converted to relational SAS data sets, we can use the extensive reporting capabilities of SAS to present the define.xml metadata in a variety of output formats like PDF, RTF or Excel. The PDF representation of the define.xml will allow us to print the metadata contained in the define.xml.

ASSESSING THE QUALITY OF THE DEFINE.XML

When submitting clinical study data in electronic (SDTM) format to the FDA it is obvious that the metadata that is contained in the define.xml file, which comes along with this data, should accurately describe the data. The process used to create the define.xml file may not guarantee this. For this reason, it is important to validate the define.xml file against the data independently from the process that created the define.xml file. Once the metadata that is contained in the define.xml is converted to relational SAS data sets, we can use SAS to perform various checks to validate the define.xml against the clinical study data.

XML 101

In this section we present a short introduction to XML.

BASIC SYNTAX

The Extensible Markup Language (XML) [11] 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 [12].

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"`
		`link: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 [13]

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 “&let;”

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 [14].
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" ?>
<heartofrockandsoul rank="834">
 <entry>
   <artist>Slim Harpo</artist>
   <title>Baby Scratch My Back</title>
   <producer>J.D. Miller</producer>
   <writer>James Moore</writer>
   <label>Excello 2273</label>
   <year>1966</year>
   <billboard>#16</billboard>
 </entry>
 <entry rank="860">
   <artist>Bo Diddley</artist>
   <title>You Can't Judge a Book by Looking at the Cover</title>
   <producer>Leonard Chess</producer>
   <writer>Willie Dixon</writer>
   <label>Checker 1019</label>
   <year>1962</year>
   <billboard>#48</billboard>
 </entry>
</heartofrockandsoul>
```

OTHER XML STANDARDS

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.

To be able to extract information from an XML document the **XPath** standard 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 `/heartofrockandsoul/entry/artist` returns all artist elements: “Slim Harpo” and “Bo Diddley”.
- The XPath location path `/heartofrockandsoul/entry/@rank` returns the “834” and “860” attributes.

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

XSL is the Extensible Style sheet Language [15], 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.

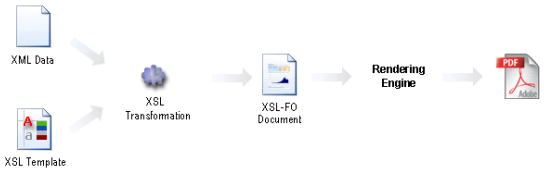
The following three languages can be distinguished:

- 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.

Earlier in this document we saw that the CDISC SDTM / ADaM Pilot Project used an XSL style sheet to display the define.xml file in a browser. It should be noticed that the way the define.xml file is displayed in a browser will vary depending on which browser version is used (Internet Explorer 6.0, Internet Explorer 7.0, Firefox, Google Chrome, Safari, …).

XSL Formatting Objects (**XSL-FO**) is an XML language that describes the format of an XSL-FO document. Once an XSL-FO document has been generated, it is then passed to an application called an FO processor (rendering engine). An FO processor can convert an XSL-FO document into something that is readable, printable or both. The most common output of XSL-FO is a PDF file.

Figure 3: *Using XSL-FO to convert an XML document to PDF*



Being able to create an XSL template that will transform an XML document to an XSL-FO document that can be rendered to a PDF file is complicated. It is not for the faint hearted. The XSL-FO specification contains over 400 pages. In a future where data and metadata becomes more XML based, it might be well worth investing time in mastering XSL-FO to generate PDF renditions of XML documents.

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).

Extending the ODM has the following requirements:

- The vendor must supply an XML-Schema fully describing their extended ODM document.
- Extended ODM files should reference the proper XML-schema extension.
- The extension may add new XML elements and attributes, but may not render any standard ODM file obsolete. Vendor extensions cannot be used for information that can be expressed using other ODM elements.
- All new element and attribute names must use distinct XML namespaces to insure that there are no naming conflicts with other vendor extensions.
- Removing all vendor extensions from an extended ODM file must result in a meaningful and accurate standard ODM file.
- Vendors should be able to produce ODM files free of any vendor extensions upon request.

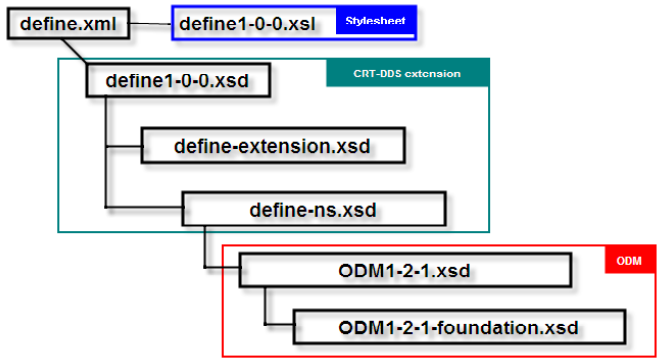
To create the define.xml extension three files have been provided:

- 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.

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

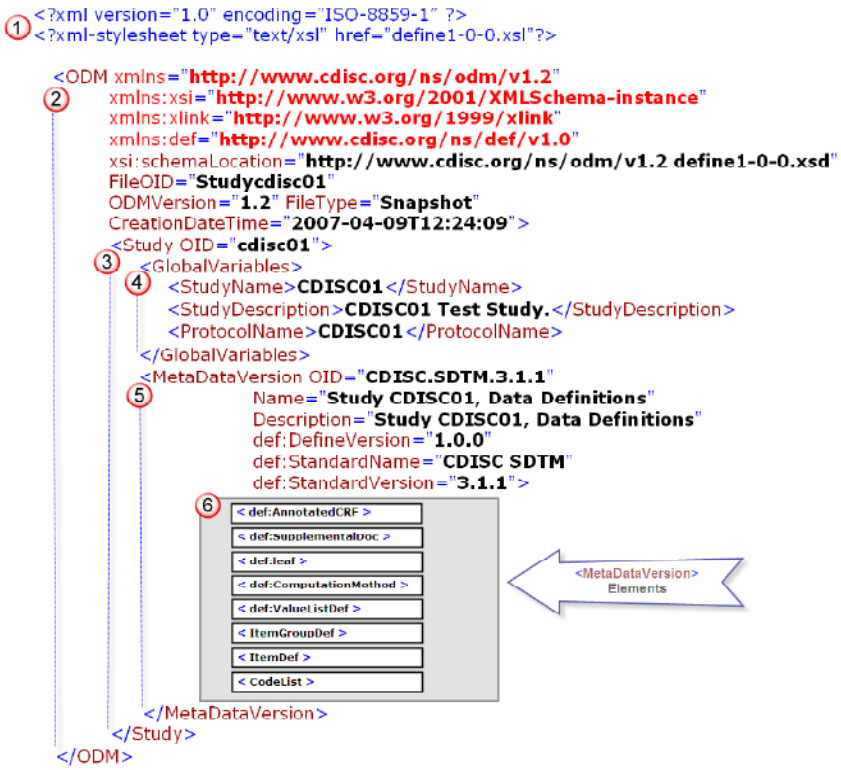


DEFINE.XML BUILDING BLOCKS

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	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.

Figure 7: *Referencing external data sets*

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.

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.
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.

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>
```

The next version of the define.xml will be an extension of a more recent ODM version, where the ComputationMethod element will be replaced by a more structured MethodDef (Type="Computation") element.

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.

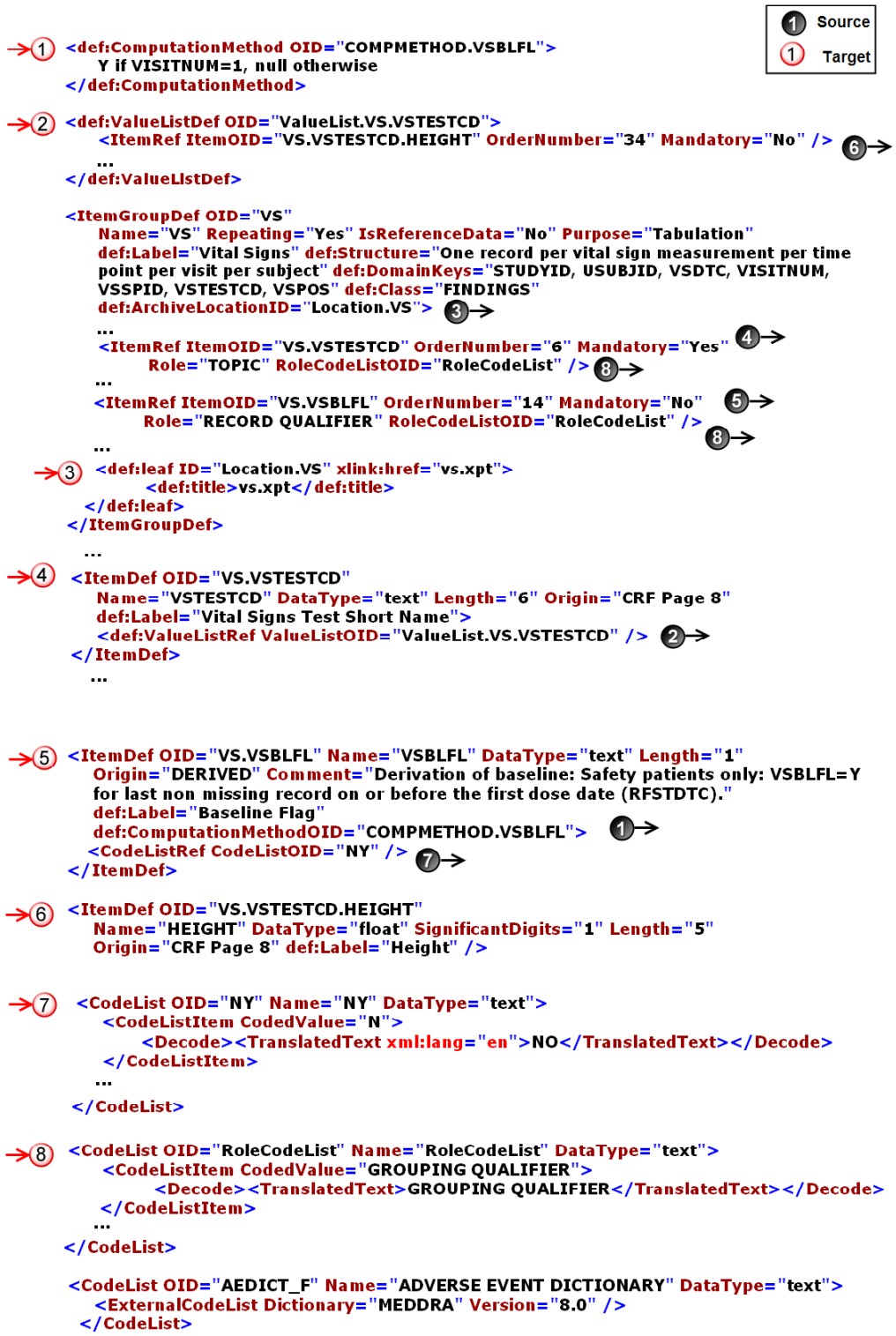
The def:ValueListDef element provides additional value-level metadata for certain variables that are part of a normalized data structure. For example, the Vital Signs domain has a measurement parameter (VSTESTCD) that stores the name of a measurement (height, weight, systolic blood pressure, …). A corresponding 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”;

Figure 9: *Relations within a define.xml*



DESIGNING A RELATIONAL DATA STRUCTURE FOR THE DEFINE.XML

In the previous paragraphs we have seen the structure of the define.xml file. Also, we have seen the relations between the different parts of the define.xml file. Since the define.xml file does not have a 2-dimensional data structure, it is not a trivial task to translate the define.xml to number of 2-dimensional SAS data set with rows and columns. We will need to create a relational data model from the define.xml.
According to the rules of ODM the OID (object identifier) for a MetaDataVersion, ItemGroupDef, ItemDef or CodeList element must be unique within a single study. Together with the Study OID, these object identifiers will serve as keys in the relational data model.

We can map the define.xml to the following data sets (all data sets also contain the Study and MetaDataVersion OID, or SAS variables Study_OID and MetaDataVersion_OID):

- **MetaDataVersion** which contains one record:
  - ODM attributes (xsi:schemaLocation, FileOID, ODMVersion, FileType, CreationDateTime)
  - Study OID attribute
  - Text of GlobalVariables child elements: StudyName, StudyDescription and ProtocolName
  - MetaDataVersion attributes: OID, Name, Description, def:DefineVersion, def:StandardName and def:StandardVersion
- **AnnotatedCRF** which contains:
  - For every def:AnnotatedCRF element and every def:DocumentRef child element the leafID attribute
- **SupplementalDoc** which contains:
  - For every def:SupplementalDoc element and every def:DocumentRef child element the leafID attribute
- **Leaf** which contains:
  - For every def:AnnotatedCRF element the leafID attribute and the corresponding def:leaf@xlink:href and def:leaf/title
  - For every def:SupplementalDoc element the leafID attribute and the corresponding def:leaf@xlink:href and def:leaf/title
- **ItemGroupDef** which contains for every ItemGroupDef:
  - the ItemGroupDef attributes: OID, Name, Repeating, IsReferenceData, Purpose, def:Label, def:Structure, def:DomainKeys, def:Class, def:ArchiveLocationID
  - from ItemGroupDef/def:leaf the ID attribute, xlink:href attribute and the contents of the def:title child element (this is possible since every ItemGroupDef has exactly one def:leaf child element)
- **ItemGroupDef_ItemRef** which contains for every ItemRef element in an ItemGroupDef element:
  - The OID attribute of the parent ItemGroupDef element
  - ItemRef attributes: ItemOID, OrderNumber, Mandatory, Role and RoleCodeListOID
- **ItemDef** which contains for every ItemDef element:
  - ItemDef attributes: OID, Name, DataType, Length, SignificantDigits, SASFieldName, Origin, Comment, Def:Label, Def:DisplayFormat and Def:ComputationMethodOID
  - If applicable, the OID of the associated codelist: CodeListRef/@CodeListOID
  - If applicable, the OID of the associated valuelist: def:ValueListRef/@ValueListOID
- **ComputationMethod** which contains for every def:ComputationMethod element:
  - The OID attribute and the text content of the element.

  The OID for a def:ComputationMethod element must be unique within a single study.
- **CodeList** which contains for every CodeList element:
  - Attributes OID, Name, DataType and SASFormatName
- **CodeList_CodeListItem** which contains for every CodeListItem element:
  - OID Attribute of the parent CodeList element
  - CodeListItem attributes (CodedValue and Rank)
  - Content of CodeListItem/TranslatedText element and the CodeListItem/TranslatedText/@xml:lang attribute
- **ExternalCodeList** which contains for every ExternalCodeList element:
  - OID Attribute of the parent CodeList element
  - ExternalCodeList attributes (Dictionary and Version)
- **ValueList** which contains for every def:ValueListDef element:
  - The OID attributes

  The OID for a def:ValueListDef element must be unique within a single study.
- **ValueListItem** which contains for every def:ValueListDef/ItemRef:

- OID Attribute of the parent def:ValueListDef element
- ItemRef attributes: ItemOID, OrderNumber and Mandatory

Appendix 2 shows how the define.xml translates to a relational database.

USING THE SAS XML MAPPER

To be able to read the define.xml file 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 file to rows and columns in the rectangular SAS tables. An XMLMAP file is an XML file itself. Once we have an XMLMAP we can easily create SAS data sets with the following code:

```
FILENAME define  "c:\Projects\CDISC\Define.xml\define.xml";
FILENAME sxlemap "c:\Projects\CDISC\Define.xml\DefineXML.map";
LIBNAME  define XML XMLMAP=sxlemap access=READONLY;
PROC COPY IN=define OUT=outlib;
RUN;
```

Figure 10 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 10: *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>
```

An XMLMAP can be created with the SAS XML Mapper, which is a Java-based graphical application that helps in creating and modifying XMLMaps [17]. Either the specific XML file (define.xml) or the schema that defines the rules of the XML file can be used as input for the XML Mapper. The advantage of using the schema is that it contains definitions of optional XML elements and attributes that may not be a part of the specific define.xml file. This will allow for the creation of a more generic XMLMAP. Figure 11 shows the SAS XML Mapper interface with the define.xml loaded in the left panel.

VALIDATING THE DEFINE.XML

Earlier we talked about two different ways to verify whether XML files are coded correctly:

Well-formed	The XML code must be syntactically correct.
Valid	If the XML file has an associated XML Schema, the elements must appear in the defined structure and the content of the individual elements must conform to the declared data types specified in the schema.

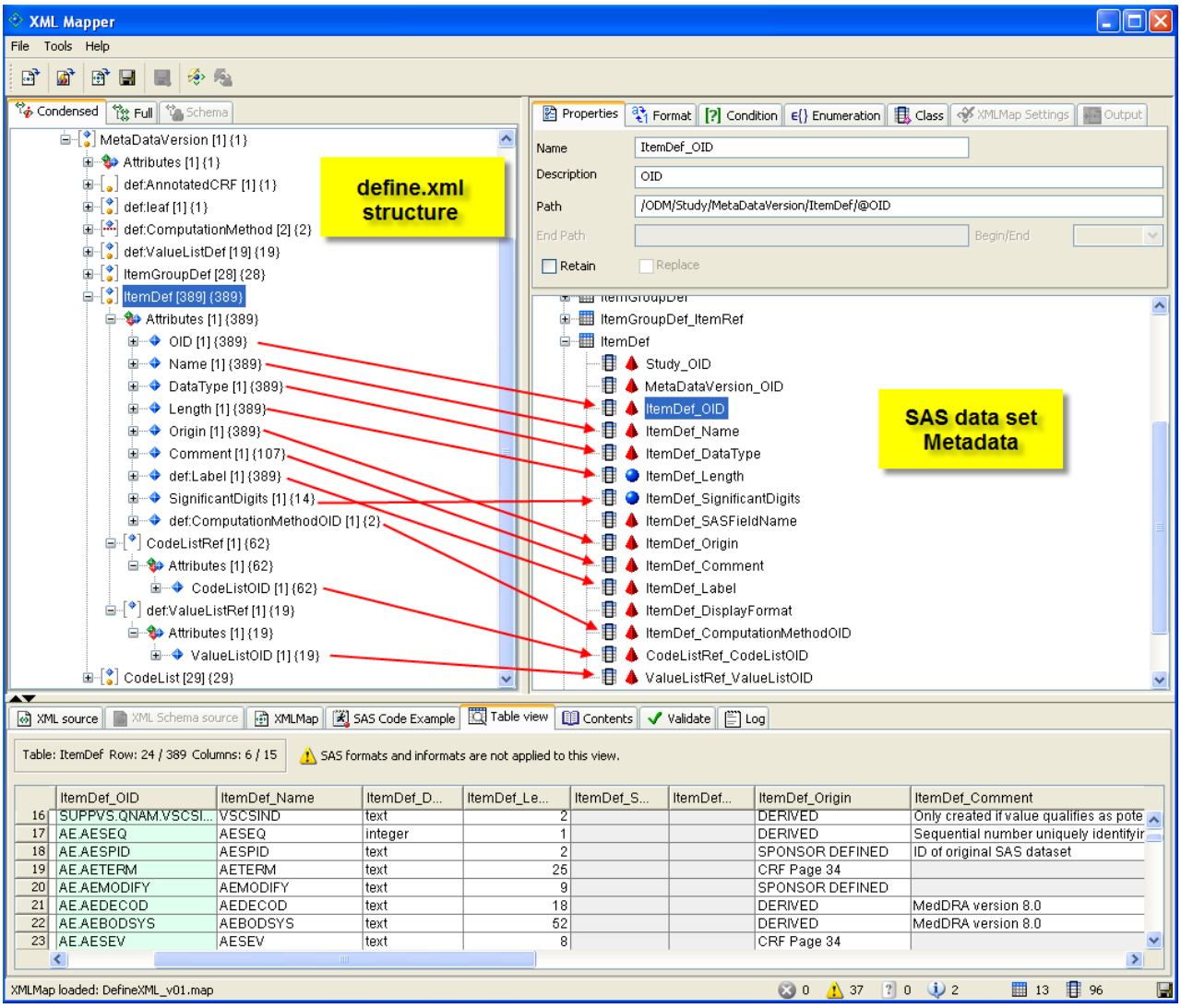
There are numerous tools available that can check well-formedness and validate an XML file against a schema. Also the latest version of the SAS XML Mapper (92.110 09w09 - FEB 2009) will validate an XML file against it's schema[1].

[1] To make sure that the SAS XML Mapper will validate XML files, we have to set an option within the SAS XML Mapper application. Start the SAS XML Mapper and choose Tools - Options - Processing. Then make sure that the XML parsing validation level is set to high.

To further ensure the quality of the define.xml, more validation needs to be performed:

- Validate against the rules as specified in the Case Report Tabulation Data Definition Specification (define.xml) version 1.0 [7]. An example of this is the rule that the **ItemOID** attribute in an **ItemRef** element must match the **OID** attribute of a corresponding **ItemDef** element[2].
- Check the consistency between the define.xml content and the SAS transport files. For example, an **ItemGroupDef** element must contain an **ItemRef** element for each variable included in the corresponding SAS transport file.
- Check the consistency between the define.xml and the SDTM metadata. An example of this is the "Mandatory" attribute of the **ItemRef** element, which needs to be consistent with the SDTM Core attribute (Required, Expected, Permissible).

Figure 11: *SAS XML Mapper interface*



[2] These kind of rules can be checked with Schematron, a rule-based XML validation language.

Once we have the define.xml converted into relational SAS data sets, it will be easy to perform these validation checks. Some code examples:

To check for duplicate **ItemDef** elements:

```
PROC SQL;
  SELECT ItemDef_OID, ItemDef_Name, ItemDef_Label
  FROM ItemDef
  GROUP BY ItemDef_OID
  HAVING COUNT(*) > 1
  ORDER BY ItemDef_OID;
QUIT;
```

To check for duplicate **ItemRef** elements within an **ItemGroupDef** element:

```
PROC SQL;
  SELECT ItemGroupDef_OID, ItemRef_ItemOID, ItemRef_OrderNumber
  FROM ItemgroupDef_itemref
  GROUP BY ItemGroupDef_OID, ItemRef_ItemOID
  HAVING COUNT(*) > 1
  ORDER BY ItemGroupDef_OID, ItemRef_ItemOID;
QUIT;
```

To check for **ItemRef** elements within an **ItemDefGroup** without an **ItemDef** element:

```
PROC SQL;
    SELECT ItemRef_ItemOID
    FROM ItemgroupDef_itemref
  EXCEPT
    SELECT ItemDef_OID
    FROM ItemDef
QUIT;
```

CONCLUSION

SAS/Base together with the SAS XML Mapper enable the creation of a PDF rendition of the define.xml file that solves the printing issue that was identified in the first CDISC SDTM / ADaM Pilot Project. Converting the define.xml file into relational SAS data sets also enables the use of SAS to perform various checks to validate the define.xml against the clinical study data. The methodology described in this paper is applicable to the current version of the define.xml file, but also to future extensions.

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". April 2006.
   (http://www.fda.gov/cder/guidance/7087rev.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 1.4, August 1, 2007
   (http://www.fda.gov/cder/regulatory/ersr/Studydata.pdf)
3. U.S. Department of Health and Human Services Food and Drug Administration Center for Drug Evaluation and Research (CDER). "Guidance for Industry: Providing Regulatory Submissions in Electronic Format — NDAs". January 2006.
   (http://www.fda.gov/cder/regulatory/ersr/2007_Waiver/FDA_eSub-1999_eNDA.pdf and
   http://www.fda.gov/cder/regulatory/ersr/waiver.htm)
4. U.S. Department of Health and Human Services, Food and Drug Administration (FDA), [Docket Nos. 1999D-0054, 2001D-0475, and 2003D-0364]. "Guidances on Providing Regulatory Submissions in Electronic Format; Withdrawal of Guidances". September 22, 2006.
   (http://www.cdisc.org/publications/99d-0054-nwl0001.pdf)
5. CDISC Study Data Tabulation Model Version 1.1, April 28, 2005
   (http://www.cdisc.org/models/sdtm/v1.1/index.html)
6. CDISC SDTM Implementation Guide V3.1.1, August 26, 2005
   (http://www.cdisc.org/models/sdtm/v1.1/index.html)
7. Case Report Tabulation Data Definition Specification (define.xml), Version 1.0, February 9, 2005
   (http://www.cdisc.org/models/def/v1.0/index.html)
8. CDISC Operational Data Model (ODM), Version 1.2.1, January, 2005
   (http://www.cdisc.org/models/odm/v1.2.1/index.html)
9. CDISC Metadata Submission Guidelines, Appendix to the Study Data Tabulation Model Implementation Guide 3.1.1, Draft version 0.9, July 25, 2007 (http://www.cdisc.org/models/sdtm/v1.1/index.html)
10. CDISC SDTM/ADaM Pilot Project Report. January 31, 2008.
    (http://www.cdisc.org/publications/SDTMADaMPilotProjectReport.pdf)
11. Extensible Markup Language (XML) 1.0, Fourth Edition, August 16, 2006
    (http://www.w3.org/TR/2006/REC-xml-20060816)
12. Eric T. Ray, 2003, Learning XML, *Creating Self-Describing Data*. 2nd Edition, (O'Reilly and Associates)
13. Bert Bos, 2000, XML in 10 points. (http://www.w3.org/XML/1999/XML-in-10-points)
14. Eric van der Vlist, 2002, XML Schema, *The W3C's Object-Oriented Descriptions for XML* (O'Reilly and Associates)
15. Dough Tidwell, 2001, XSLT, *Mastering XML Transformations*. (O'Reilly and Associates)
16. Dave Pawson, 2002, XSL-FO, *Making XML Look Good in Print.* (O'Reilly and Associates)
17. SAS Institute Inc.,(2004): SAS® 9.1.3 XML LIBNAME Engine User's Guide , Cary NC
    (http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/base_xmlug_10166.pdf#page=99)
18. 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 9000 other papers that were presented at SAS usergroups.

APPENDICES

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 2 **Define.xml** as a **relational SAS database**