PharmaSUG2012 - Paper HW02-SAS

Using the SAS® Clinical Standards Toolkit 1.4 for define.xml creation

Lex Jansen, SAS Institute Inc., Cary, NC

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 (CRT-DDS, also known as define.xml) format as prepared by the CDISC define.xml team.

This workshop will provide an introduction to the structure and content of the define.xml file. The SAS® Clinical Standards Toolkit will then be used to create the define.xml file, including Value Level metadata.

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 The previous guidance, originally issued in 1999, has been withdrawn as of the same date.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].

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.

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.

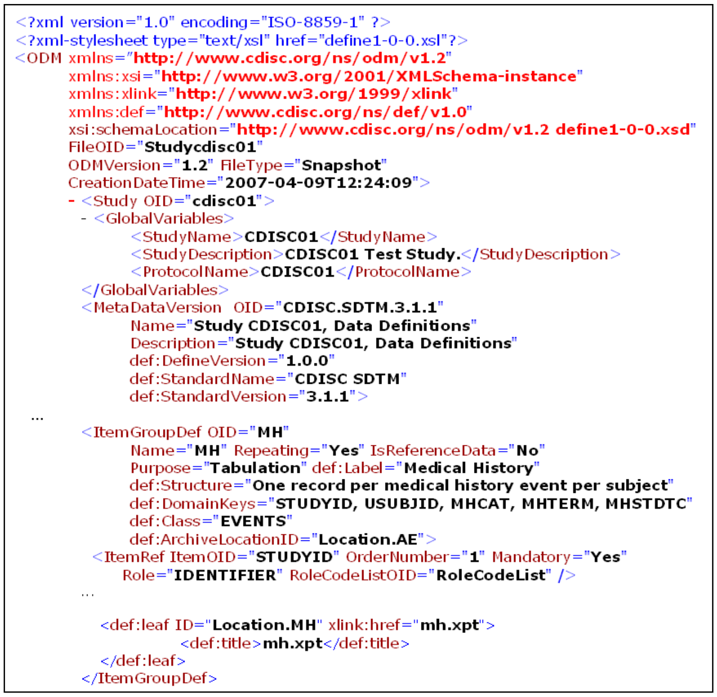
syntax	CDATA	example
`<![CDATA unparsed character data ]]>`		<TranslatedText xml:lang="**en**"><![CDATA[Classifies subjects based on Hy's Law - 2 variations - For each variation, subject is classified as normal or abnormal at baseline and as normal or abnormal based on most extreme value during treatment,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

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 “&lt;”

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

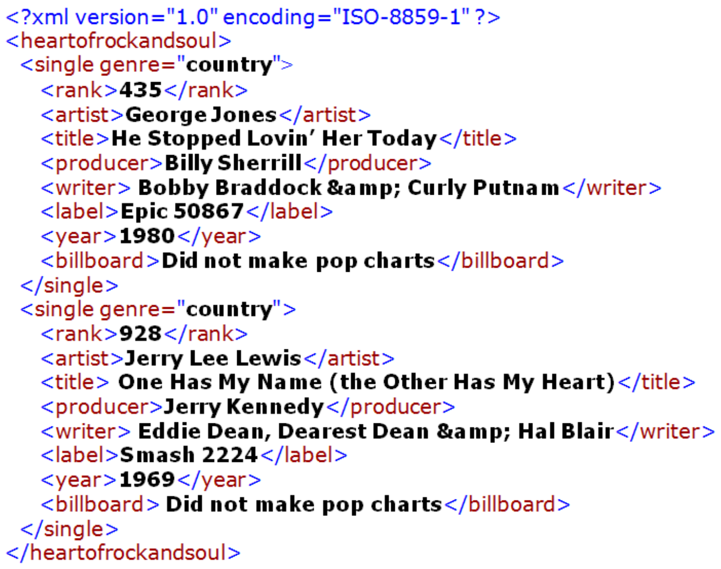
An **XML schema** is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type. This description goes above and beyond the basic syntax constraints imposed by well-formedness of an XML document [11].

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*[1]



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<heartofrockandsoul>
  <single genre="country">
    <rank>435</rank>
    <artist>George Jones</artist>
    <title>He Stopped Lovin’ Her Today</title>
    <producer>Billy Sherrill</producer>
    <writer> Bobby Braddock &amp; Curly Putnam</writer>
    <label>Epic 50867</label>
    <year>1980</year>
    <billboard>Did not make pop charts</billboard>
  </single>
  <single genre="country">
    <rank>928</rank>
    <artist>Jerry Lee Lewis</artist>
    <title> One Has My Name (the Other Has My Heart)</title>
    <producer>Jerry Kennedy</producer>
    <writer> Eddie Dean, Dearest Dean &amp; Hal Blair</writer>
    <label>Smash 2224</label>
    <year>1969</year>
    <billboard> Did not make pop charts</billboard>
  </single>
</heartofrockandsoul>
```

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 **`/heartofrockandsoul/single/artist`** returns all artist elements: “George Jones” and “Jerry Lee Lewis”.
- The XPath location path **`/heartofrockandsoul/single/@genre`** returns the “country” attribute.

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.

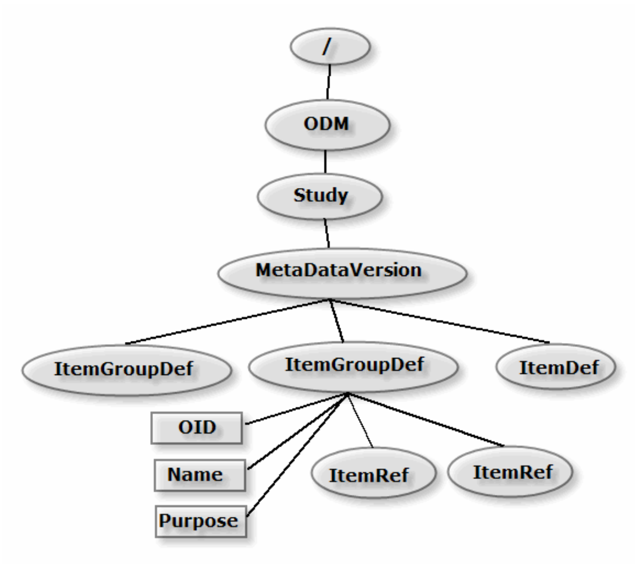
---

[1] This particular document may not be a valid define.xml document, but it is certainly appropriate for a PharmaSUG in Nashville, home of the Grand Ole Opry.

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

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

Figure 3: define.xml as a tree



In Figure 3 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.

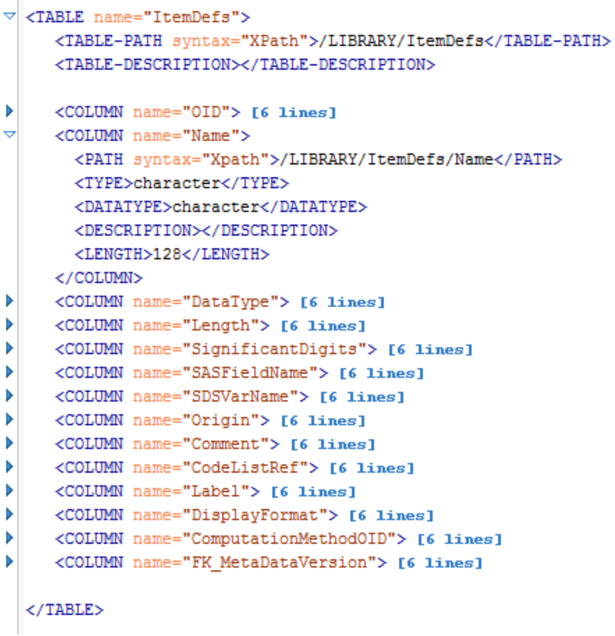
Examples:

- 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.

Figure 4 shows part of an XMLMAP. It specifies that the data set **ItemDef** has a variable named **Name**, whose content is defined by the XPath specification **/LIBRARY/ItemDefs/Name**.

Figure 4: *XMLMAP example*



```
<TABLE name="ItemDefs">
   <TABLE-PATH syntax="XPath">/LIBRARY/ItemDefs</TABLE-PATH>
   <TABLE-DESCRIPTION></TABLE-DESCRIPTION>

   <COLUMN name="OID"> [6 lines]
   <COLUMN name="Name">
     <PATH syntax="Xpath">/LIBRARY/ItemDefs/Name</PATH>
     <TYPE>character</TYPE>
     <DATATYPE>character</DATATYPE>
     <DESCRIPTION></DESCRIPTION>
     <LENGTH>128</LENGTH>
   </COLUMN>
   <COLUMN name="DataType"> [6 lines]
   <COLUMN name="Length"> [6 lines]
   <COLUMN name="SignificantDigits"> [6 lines]
   <COLUMN name="SASFieldName"> [6 lines]
   <COLUMN name="SDSVarName"> [6 lines]
   <COLUMN name="Origin"> [6 lines]
   <COLUMN name="Comment"> [6 lines]
   <COLUMN name="CodeListRef"> [6 lines]
   <COLUMN name="Label"> [6 lines]
   <COLUMN name="DisplayFormat"> [6 lines]
   <COLUMN name="ComputationMethodOID"> [6 lines]
   <COLUMN name="FK_MetaDataVersion"> [6 lines]

</TABLE>
```

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

XSL is the Extensible Style sheet Language [12], 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).

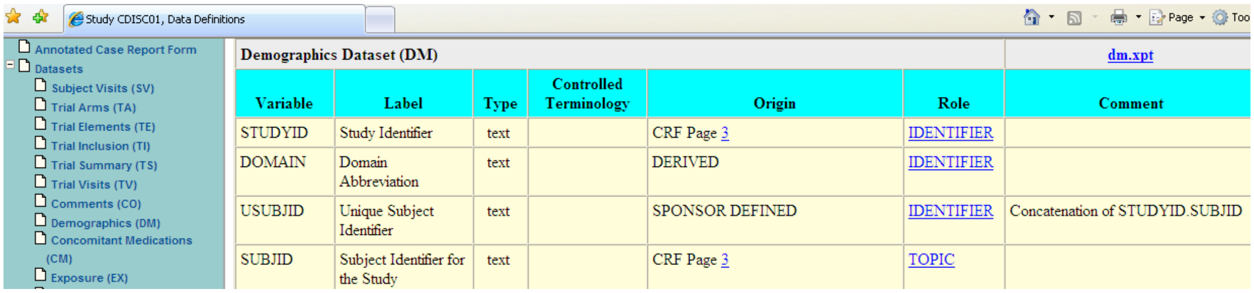
XSL is actually a family of transformation languages which allows one to describe how files encoded in the XML standard are to be formatted or transformed.

The following three languages can be distinguished:

- XSL Transformations (XSLT): an XML language for transforming XML documents into other XML documents, into HTML documents, or into any other text based documents (like a SAS program), or even a PDF file.
- 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.

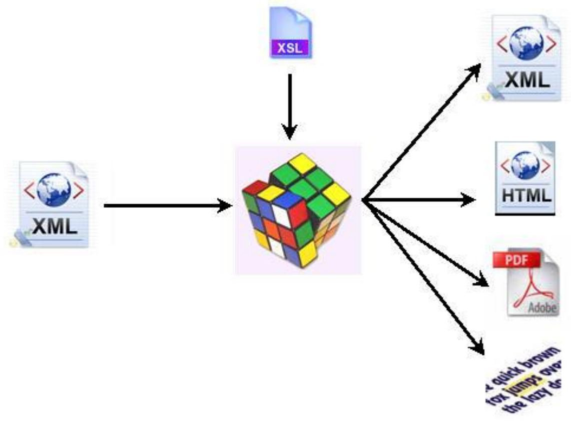
Figure 5: *Define.xml displayed in a browser using an XSL style sheet*



Demographics Dataset (DM) dm.xpt

Variable	Label	Type	Controlled Terminology	Origin	Role	Comment
STUDYID	Study Identifier	text		CRF Page 3	IDENTIFIER	
DOMAIN	Domain Abbreviation	text		DERIVED	IDENTIFIER	
USUBJID	Unique Subject Identifier	text		SPONSOR DEFINED	IDENTIFIER	Concatenation of STUDYID.SUBJID
SUBJID	Subject Identifier for the Study	text		CRF Page 3	TOPIC	

Figure 6: *Transforming with XSLT*



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:

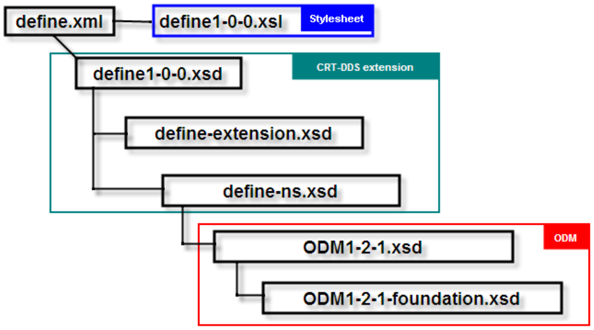
- a *foundation* XML Schema file (ODM1-2-1-foundation.xsd), which defines the elements, attributes and structure of the base ODM schema.
- an *application* XML Schema file (ODM1-2-1.xsd.) which imports the foundation XML Schema and other schema definitions needed by ODM, such as the core W3C XML schema (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 [13]. This white paper, created by the CDISC XML Technology 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 7: *The define.xml (CRT-DDS) and the associated style sheet and schemas*



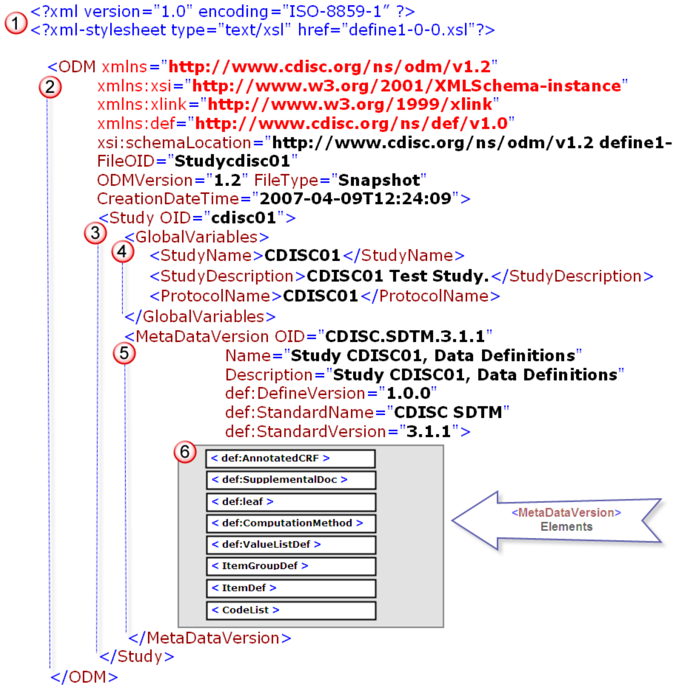
DEFINE.XML BUILDING BLOCKS

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 8 shows a high-level overview of a define.xml file.

Figure 8: *The building blocks of the define.xml (CRT-DDS)*



```
<?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">
         < def:AnnotatedCRF >
         < def:SupplementalDoc >
         < def:leaf >
         < def:ComputationMethod >
         < def:ValueListDef >
         < ItemGroupDef >
         < ItemDef >
         < CodeList >
       </MetaDataVersion>
     </Study>
</ODM>
```

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.

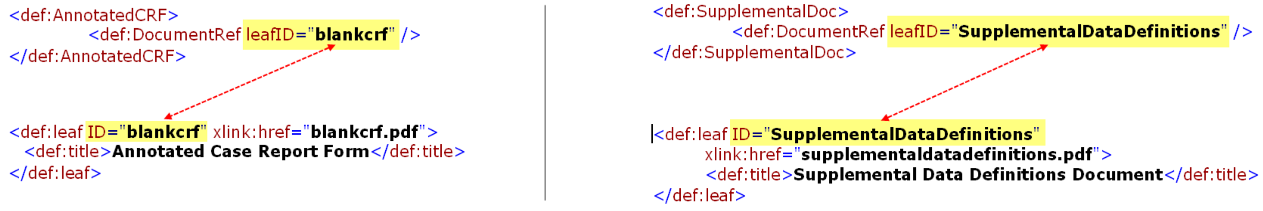
METADATAVERSION ELEMENT DETAILS

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 9 shows the relation between these elements. The **def:DocumentRef/@leafID** must match the corresponding **def:leaf/@ID**.

Figure 9: *Referencing external PDF files*



```
<def:AnnotatedCRF>
          <def:DocumentRef leafID="blankcrf" />
</def:AnnotatedCRF>


<def:leaf ID="blankcrf" xlink:href="blankcrf.pdf">
  <def:title>Annotated Case Report Form</def:title>
</def:leaf>
```

```
<def:SupplementalDoc>
          <def:DocumentRef leafID="SupplementalDataDefinitions" />
</def:SupplementalDoc>


<def:leaf ID="SupplementalDataDefinitions"
       xlink:href="supplementaldatadefinitions.pdf">
       <def:title>Supplemental Data Definitions Document</def:title>
</def:leaf>
```

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*

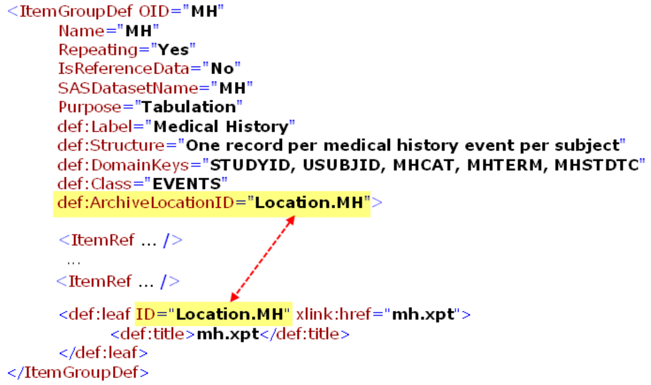
Attribute	Status	Description
OID	Required	The unique identifier of the domain.
Name	Required	The file name of the data set or data domain name (e.g., “DM” for Demographics)
Repeating	Required	“Yes” for domains with more than one record per subject whereas, “No” for domains with 1 record per subject.
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[2]	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 10.

Figure 10: *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>
```

<u>ItemRef</u> and <u>ItemDef</u>

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.

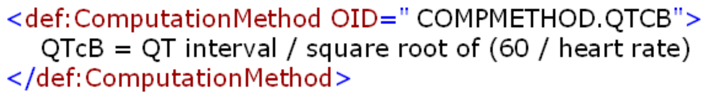
---

[2] 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.

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 11: *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[3]	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.

---

[3] 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.

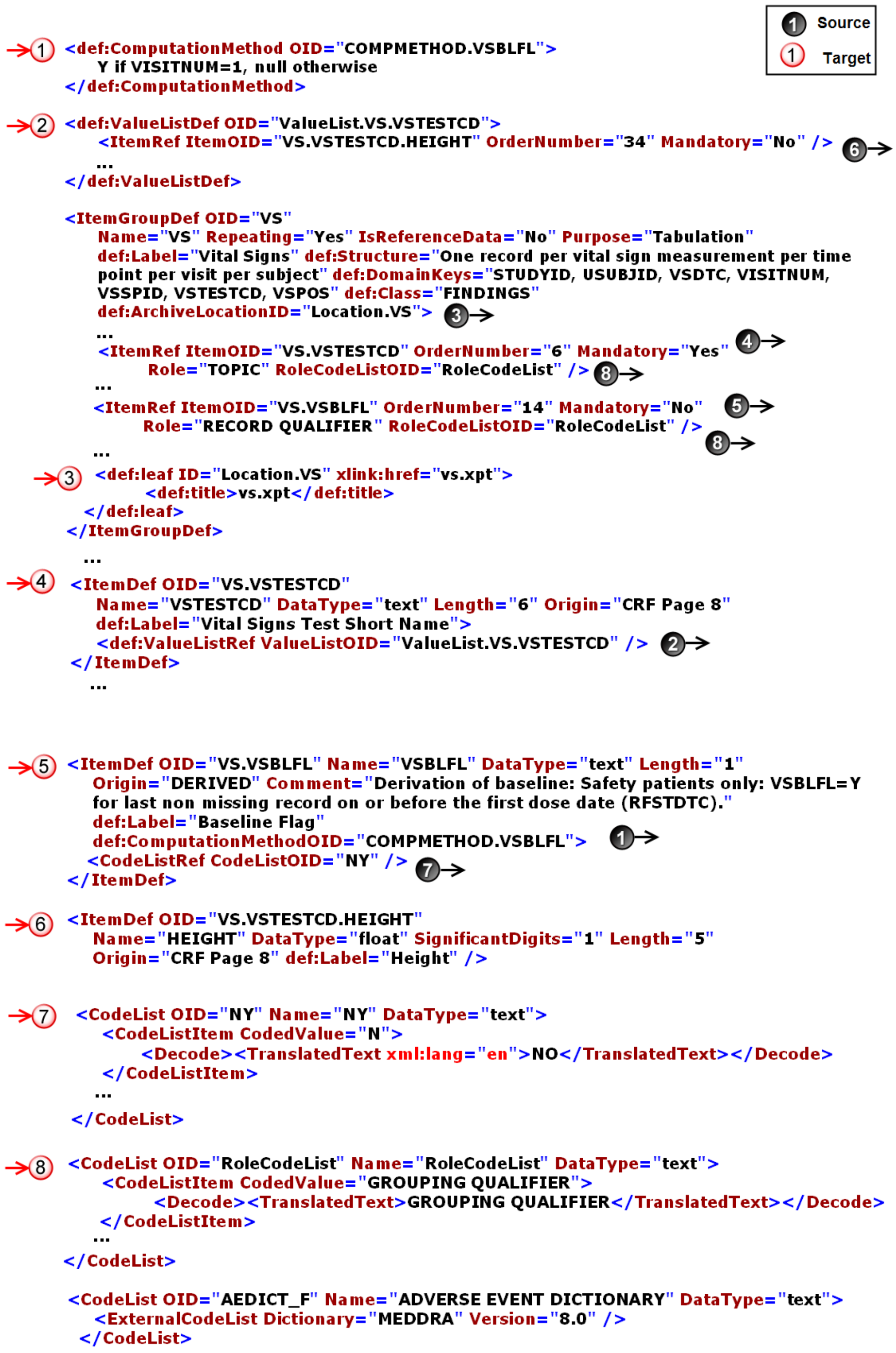
The def:ValueListDef element provides additional value-level metadata for certain variables that are part of a normalized data structure. For example, the Vital Signs domain has a measurement parameter (VSTESTCD) that stores the name of a measurement (height, weight, systolic blood pressure, …). A corresponding 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 12 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 12: Relations within a define.xml



```
<def:ComputationMethod OID="COMPMETHOD.VSBLFL">
     Y if VISITNUM=1, null otherwise
</def:ComputationMethod>

<def:ValueListDef OID="ValueList.VS.VSTESTCD">
     <ItemRef ItemOID="VS.VSTESTCD.HEIGHT" OrderNumber="34" Mandatory="No" />
     ...
</def:ValueListDef>

<ItemGroupDef OID="VS"
     Name="VS" Repeating="Yes" IsReferenceData="No" Purpose="Tabulation"
     def:Label="Vital Signs" def:Structure="One record per vital sign measurement per time
     point per visit per subject" def:DomainKeys="STUDYID, USUBJID, VSDTC, VISITNUM,
     VSSPID, VSTESTCD, VSPOS" def:Class="FINDINGS"
     def:ArchiveLocationID="Location.VS">
     ...
     <ItemRef ItemOID="VS.VSTESTCD" OrderNumber="6" Mandatory="Yes"
          Role="TOPIC" RoleCodeListOID="RoleCodeList" />
     ...
     <ItemRef ItemOID="VS.VSBLFL" OrderNumber="14" Mandatory="No"
          Role="RECORD QUALIFIER" RoleCodeListOID="RoleCodeList" />
     ...
     <def:leaf ID="Location.VS" xlink:href="vs.xpt">
          <def:title>vs.xpt</def:title>
   </def:leaf>
</ItemGroupDef>

   ...
 <ItemDef OID="VS.VSTESTCD"
     Name="VSTESTCD" DataType="text" Length="6" Origin="CRF Page 8"
     def:Label="Vital Signs Test Short Name">
     <def:ValueListRef ValueListOID="ValueList.VS.VSTESTCD" />
 </ItemDef>
    ...


<ItemDef OID="VS.VSBLFL" Name="VSBLFL" DataType="text" Length="1"
    Origin="DERIVED" Comment="Derivation of baseline: Safety patients only: VSBLFL=Y
    for last non missing record on or before the first dose date (RFSTDTC)."
    def:Label="Baseline Flag"
    def:ComputationMethodOID="COMPMETHOD.VSBLFL">
   <CodeListRef CodeListOID="NY" />
</ItemDef>

<ItemDef OID="VS.VSTESTCD.HEIGHT"
    Name="HEIGHT" DataType="float" SignificantDigits="1" Length="5"
    Origin="CRF Page 8" def:Label="Height" />


 <CodeList OID="NY" Name="NY" DataType="text">
    <CodeListItem CodedValue="N">
         <Decode><TranslatedText xml:lang="en">NO</TranslatedText></Decode>
    </CodeListItem>
    ...
 </CodeList>

<CodeList OID="RoleCodeList" Name="RoleCodeList" DataType="text">
    <CodeListItem CodedValue="GROUPING QUALIFIER">
         <Decode><TranslatedText>GROUPING QUALIFIER</TranslatedText></Decode>
    </CodeListItem>
    ...
</CodeList>

<CodeList OID="AEDICT_F" Name="ADVERSE EVENT DICTIONARY" DataType="text">
    <ExternalCodeList Dictionary="MEDDRA" Version="8.0" />
  </CodeList>
```

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 within 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 a number of SAS data set with rows and columns. SAS has defined a relational data model that represents the define.xml.

The highly structured nature of CDISC CRT-DDS data requires that any mapping to a relational format include a large number of data sets, with foreign key relationships to help preserve the intended non-relational object structure. In the SAS Clinical Standards Toolkit, foreign key relationships are enforced when validating the CDISC CRT-DDS data sets. Appendix 1 and Appendix 2 show the 39 SAS data sets that represent the CRT-DDS data model.

One could use SAS/Base to construct the information that needs to go into the 39 tables, however, it is important to understand what the relation is between the CRT-DDS tables and what information is needed in the tables. Some of the information in these 39 tables can be easily derived from the SDTM datasets that need to be described by the define.xml (dataset label, dataset name, variable names and labels, …). Other information cannot automatically be created and needs to be created through a process.

The first step in creating a define.xml file with SAS Clinical Standards Toolkit means populating the SAS data set representation of the CRT-DDS model from the SDTM domain metadata (source_tables and source_columns data sets) and the study metadata (source_study data set) by running the crtdds_sdtmtodefine macro. Depending on the completeness of this source data the crtdds_sdtmtodefine macro can (partially) populate 12 of the 39 CRT-DDS SAS tables:

clitemdecodetranslatedtext
codelistitems
codelists
computationmethods
definedocument
itemdefs
itemgroupdefitemrefs
itemgroupdefs
itemgroupleaf
itemgroupleaftitles
metadataversion
study

The remainder of the tables will not be automatically populated by SAS Clinical Standards Toolkit. Typically, the following tables need be filled by a process not included in SAS Clinical Standards Toolkit to create a CRT-DDS define.xml file that can be considered more complete.

annotatedcrfs (contains document references to the annotated case report form)
supplementaldocs (contains document references to the supplemental documentation)
mdvleaf (contains links for each of the documents listed in the AnnotatedCRF and SupplementalDocs tables)
mdvleaftitles (contains a descriptive title for each MDVLeaf item)
itemvaluelistrefs (associates each value list item to a row in the ItemDefs table)
valuelists (contains id of value lists)
valuelistitemrefs (contains id of each item in a value list)
externalcodelists (contains name and versions of external code lists)

Version 1.5 of the SAS Clinical Standards Toolkit will contain macros that will make the process of populating these 8 tables easier, compared to doing this with custom SAS/Base code.

The remaining 19 tables are typically not used in a define.xml, but have been defined to make the data model for the metadata complete.

THE SAS CLINICAL STANDARDS TOOLKIT

The SAS Clinical Standards Toolkit is a technology that exists as a set of SAS macros and metadata SAS datasets. The macros that are part of the toolkit are used by SAS® Clinical Data Integration, which can be used to populate metadata datasets and perform macro calls through a user interface.

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 an XML schema. As mentioned earlier in this paper, the CDISC XML Technologies Team has published a XML Schema Validation for Define.xml white paper that provides guidance on validating define.xml documents against the define.xml XML schemas and proposes practices and tools to improve define.xml schema validation [13].

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 an **ItemRef** element must be uniquely defined within an **ItemGroupDef** , through the uniqueness of the **ItemOID** attribute of the **ItemRef**.
- 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).

Once we have the define.xml converted into relational SAS data sets, it will be easy to perform these validation checks using the SAS Clinical Standards Toolkit.

USING THE SAS CLINICAL STANDARDS TOOLKIT

Before the macros in the SAS Clinical Standards Toolkit can be used there are a few tasks that need to be performed.:

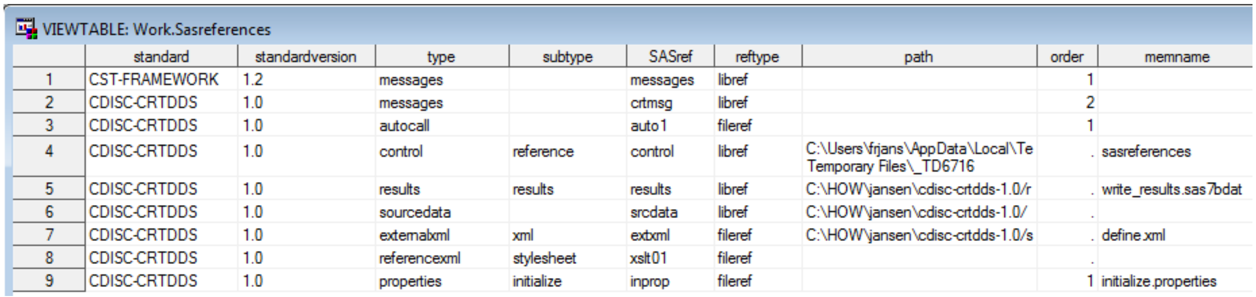
- Setup of global macro variables.
  This can be done either directly through **%let** statements, or by loading properties (used to create macro variables) with the **cst_setStandardProperties** macro:

```
%cst_setStandardProperties(_cstStandard=CST-FRAMEWORK,_cstSubType=initialize);
%cst_setStandardProperties(_cstStandard=CDISC-CRTDDS,
                           _cstStandardVersion=1.0,_cstSubType=initialize);
```

- Process input and output has to be defined by creating a process **SASReferences** data set. This data set is then communicated with the **cstutil_processsetup** macro.

An example of a SASReferences dataset for the process that creates a define.xml file is presented in Figure 13.

Figure 13: SASReferences data set



VIEWTABLE: Work.Sasreferences

	standard	standardversion	type	subtype	SASref	reftype	path	order	memname
1	CST-FRAMEWORK	1.2	messages		messages	libref		1	
2	CDISC-CRTDDS	1.0	messages		crtmsg	libref		2	
3	CDISC-CRTDDS	1.0	autocall		auto1	fileref		1	
4	CDISC-CRTDDS	1.0	control	reference	control	libref	C:\Users\frjans\AppData\Local\Te Temporary Files_TD6716	.	sasreferences
5	CDISC-CRTDDS	1.0	results	results	results	libref	C:\HOW\jansen\cdisc-crtdds-1.0/r	.	write_results.sas7bdat
6	CDISC-CRTDDS	1.0	sourcedata		srcdata	libref	C:\HOW\jansen\cdisc-crtdds-1.0/	.	
7	CDISC-CRTDDS	1.0	externalxml	xml	extxml	fileref	C:\HOW\jansen\cdisc-crtdds-1.0/s	.	define.xml
8	CDISC-CRTDDS	1.0	referencexml	stylesheet	xslt01	fileref		.	
9	CDISC-CRTDDS	1.0	properties	initialize	inprop	fileref		1	initialize.properties

The following toolkit macros are typically used in the process to create a define.xml or to read a define.xml:

- **crtdds_sdtmtodefine**
  This macro creates the 39 CRT-DDS data sets by using SDTM table, column, and study metadata (datasets source_tables, source_columns and source_study).
- **crtdds_validate**
  This macro validates the CRT-DDS data sets to make sure that they are in compliance with business rules associated with the CRT-DDS standard
- **crtdds_write**
  This macro creates the XML rendition of the CRT-DDS datasets, i.e. the define.xml

- **crtdds_xmlvalidate**
  This macro performs XML schema validation of the define.xml file
- **crtdds_read**
  This macro reads the define.xml file into the SAS representation of the CRT-DDS data model

CREATING ADDITIONAL METADATA FOR THE DEFINE.XML FILE

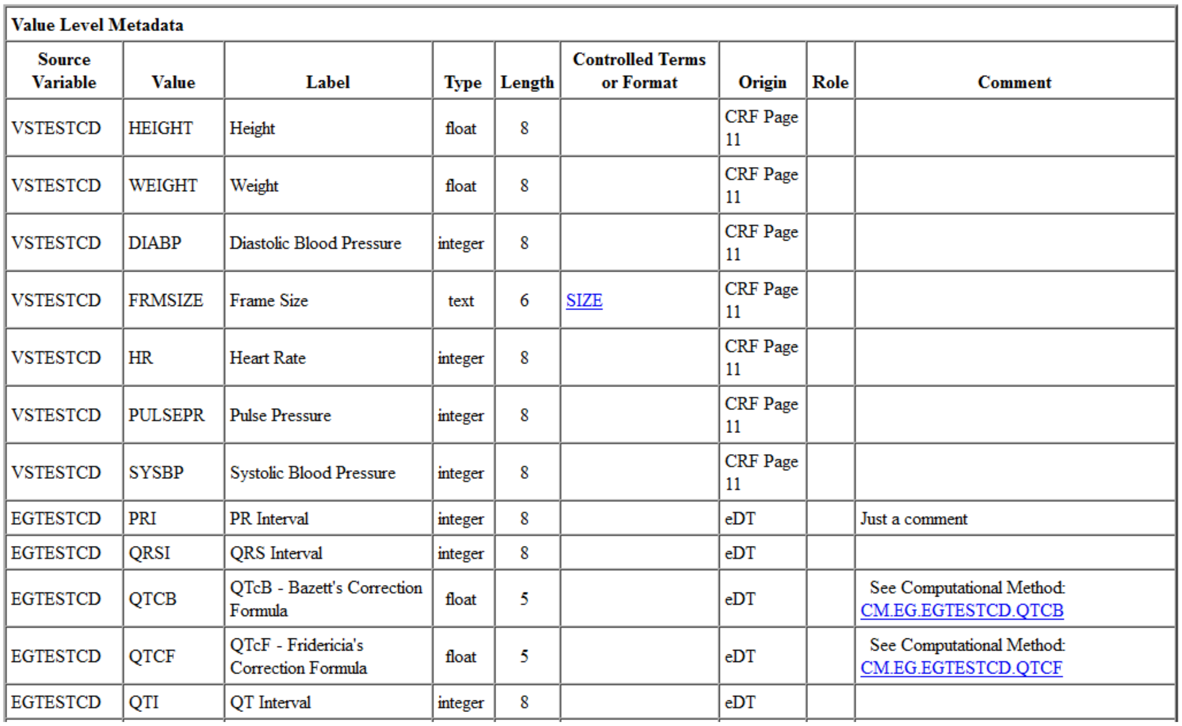
Some of the information that needs to be represented in the define.xml file cannot be automatically generated by SAS and needs to be provided by the user. The user will need to add metadata directly to the CRT-DDS datasets, and then generate the define.xml with the **crtdds_write** macro to include this additional metadata.

Examples include value level metadata and metadata describing supporting documents, such as annotated CRFs and supplemental data definitions.

Future versions of the SAS Clinical Standards Toolkit will contain macros to further automate addition of this additional metadata.

VALUE LEVEL METADATA

Value level metadata is needed for SDTM Findings domains like Vital Signs (VS) domain. The VSTESTCD variable contains values like "WEIGHT" and "HEIGHT". We need to define metadata based on the value of VSTESTCD:



Value Level Metadata

Source Variable	Value	Label	Type	Length	Controlled Terms or Format	Origin	Role	Comment
VSTESTCD	HEIGHT	Height	float	8		CRF Page 11		
VSTESTCD	WEIGHT	Weight	float	8		CRF Page 11		
VSTESTCD	DIABP	Diastolic Blood Pressure	integer	8		CRF Page 11		
VSTESTCD	FRMSIZE	Frame Size	text	6	SIZE	CRF Page 11		
VSTESTCD	HR	Heart Rate	integer	8		CRF Page 11		
VSTESTCD	PULSEPR	Pulse Pressure	integer	8		CRF Page 11		
VSTESTCD	SYSBP	Systolic Blood Pressure	integer	8		CRF Page 11		
EGTESTCD	PRI	PR Interval	integer	8		eDT		Just a comment
EGTESTCD	QRSI	QRS Interval	integer	8		eDT		
EGTESTCD	QTCB	QTcB - Bazett's Correction Formula	float	5		eDT		See Computational Method: CM.EG.EGTESTCD.QTCB
EGTESTCD	QTCF	QTcF - Fridericia's Correction Formula	float	5		eDT		See Computational Method: CM.EG.EGTESTCD.QTCF
EGTESTCD	QTI	QT Interval	integer	8		eDT		

In XML this would look as follows:

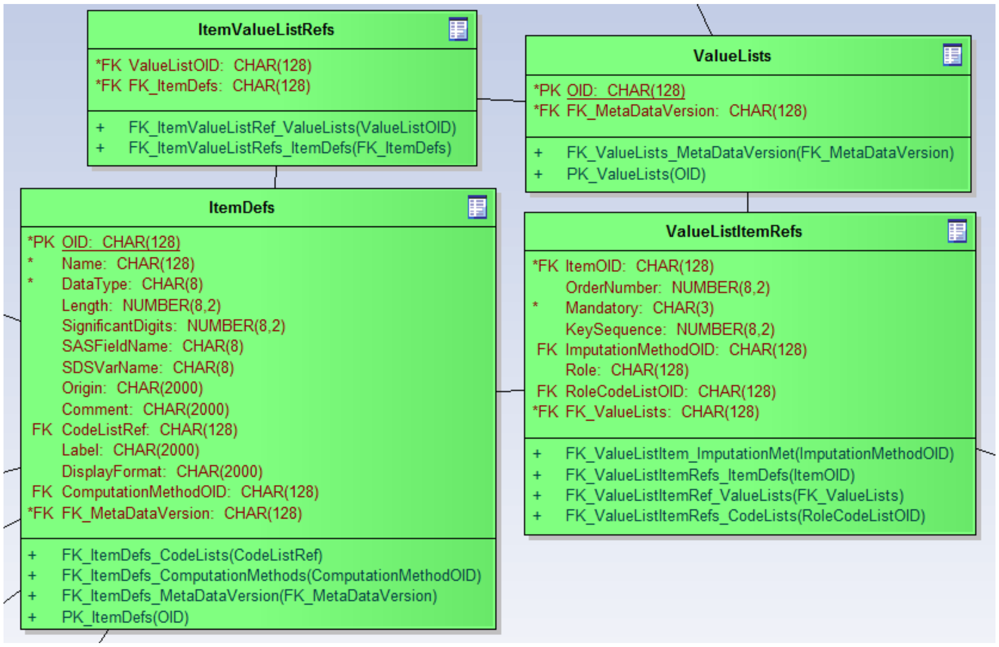
```
<def:ValueListDef OID="VL.VS.VSTESTCD">
  <ItemRef ItemOID="IT.VS.VSTESTCD.HEIGHT"  OrderNumber="1" Mandatory="No"/>
  <ItemRef ItemOID="IT.VS.VSTESTCD.WEIGHT"  OrderNumber="2" Mandatory="No"/>
  <ItemRef ItemOID="IT.VS.VSTESTCD.DIABP"  OrderNumber="3" Mandatory="No"/>
  <ItemRef ItemOID="IT.VS.VSTESTCD.FRMSIZE"  OrderNumber="4" Mandatory="No"/>
…
</def:ValueListDef>

<ItemGroupDef OID="VS1" Name="VS" Repeating="Yes" IsReferenceData="No"
    SASDatasetName="VS" Domain="VS" Purpose="Tabulation"
    def:Label="Vital Signs" def:Class="Findings"
    def:Structure="One record per vital sign measurement per time point per visit per
subject"
    def:DomainKeys="STUDYID USUBJID VSTESTCD VISITNUM VSTPTREF VSTPTNUM"
    def:ArchiveLocationID="VS1">
  <ItemRef ItemOID="COL703" Mandatory="Yes" OrderNumber="1" Role="Identifier"/>
  <ItemRef ItemOID="COL704" Mandatory="Yes" OrderNumber="2" Role="Identifier"/>
  <ItemRef ItemOID="COL705" Mandatory="Yes" OrderNumber="3" Role="Identifier"/>
  <ItemRef ItemOID="COL706" Mandatory="Yes" OrderNumber="4" Role="Identifier"/>
  <ItemRef ItemOID="COL707" Mandatory="No" OrderNumber="5" Role="Identifier"/>
  <ItemRef ItemOID="COL708" Mandatory="No" OrderNumber="6" Role="Identifier"/>
  <ItemRef ItemOID="COL709" Mandatory="Yes" OrderNumber="7" Role="Topic"/>
…
  <ItemRef ItemOID="COL733" Mandatory="No" OrderNumber="31" Role="Timing"/>
  <def:leaf ID="VS1" xlink:href="../transport/vs.xpt">
    <def:title>Vital Signs SAS transport file</def:title>
  </def:leaf>
</ItemGroupDef>

<ItemDef OID="COL709" Name="VSTESTCD" DataType="text" Length="8" SASFieldName="VSTESTCD"
Comment="Short name of the measurement, test, or examination described in VSTEST. It can be
used as a column name when converting a dataset from a vertical to a horizontal format. The
value in VSTESTCD cannot be longer than 8 characters, nor can it start with a number
(e.g.'1TEST'). VSTESTCD cannot contain characters other than letters, numbers, or
underscores. Examples: SYSBP, DIABP, BMI." def:Label="Vital Signs Test Short Name">
  <CodeListRef CodeListOID="VSTESTCD"/>
  <def:ValueListRef ValueListOID="VL.VS.VSTESTCD"/>
</ItemDef>
<ItemDef OID="IT.VS.VSTESTCD.HEIGHT" Name="HEIGHT" DataType="float" Length="8"
         Origin="CRF Page 11" def:Label="Height"/>
<ItemDef OID="IT.VS.VSTESTCD.WEIGHT" Name="WEIGHT" DataType="float" Length="8"
         Origin="CRF Page 11" def:Label="Weight"/>
<ItemDef OID="IT.VS.VSTESTCD.DIABP" Name="DIABP" DataType="integer" Length="8"
         Origin="CRF Page 11" def:Label="Diastolic Blood Pressure"/>
<ItemDef OID="IT.VS.VSTESTCD.FRMSIZE" Name="FRMSIZE" DataType="text" Length="6"
         Origin="CRF Page 11" def:Label="Frame Size">
  <CodeListRef CodeListOID="SIZE"/>
</ItemDef>
```

To be able to represent this information in the SAS CRT-DDS datasets we will need to add information to several datasets, as can be seen in Figure 14.

The following datasets are involved:

ItemDefs	contains the column metadata
ValueLists	contains id of value lists
ValueListItemRefs	contains id of each item in a value list – associates each value list item to a row in the ItemDefs table
ItemValueListRefs	associates each value list item to a row in the ItemDefs table

Figure 14: CRT-DDS Datasets related to value level metadata

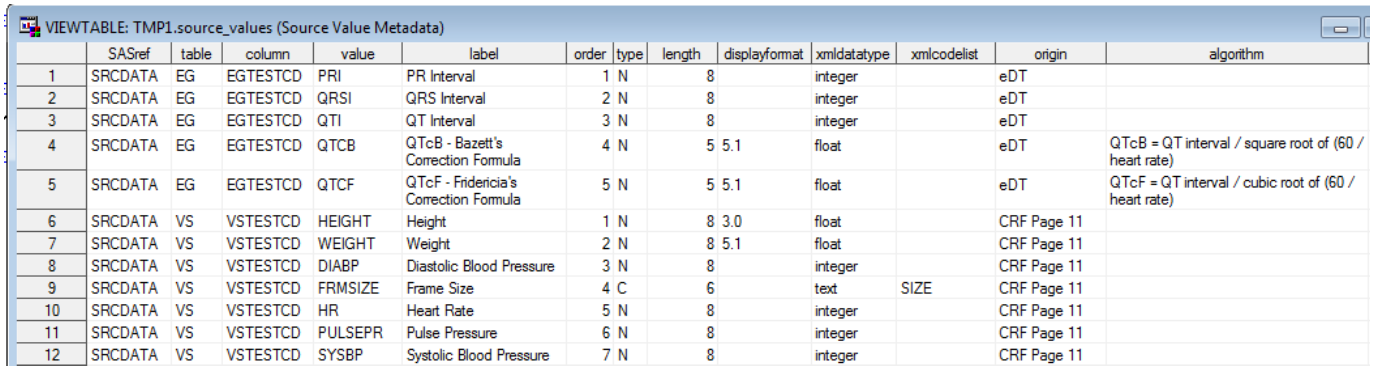

ItemValueListRefs
*FK ValueListOID: CHAR(128)
*FK FK_ItemDefs: CHAR(128)
+ FK_ItemValueListRef_ValueLists(ValueListOID)
+ FK_ItemValueListRefs_ItemDefs(FK_ItemDefs)
ValueLists
*PK OID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_ValueLists_MetaDataVersion(FK_MetaDataVersion)
+ PK_ValueLists(OID)
ItemDefs
*PK OID: CHAR(128)
* Name: CHAR(128)
* DataType: CHAR(8)
Length: NUMBER(8,2)
SignificantDigits: NUMBER(8,2)
SASFieldName: CHAR(8)
SDSVarName: CHAR(8)
Origin: CHAR(2000)
Comment: CHAR(2000)
FK CodeListRef: CHAR(128)
Label: CHAR(2000)
DisplayFormat: CHAR(2000)
FK ComputationMethodOID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_ItemDefs_CodeLists(CodeListRef)
+ FK_ItemDefs_ComputationMethods(ComputationMethodOID)
+ FK_ItemDefs_MetaDataVersion(FK_MetaDataVersion)
+ PK_ItemDefs(OID)
ValueListItemRefs
*FK ItemOID: CHAR(128)
OrderNumber: NUMBER(8,2)
* Mandatory: CHAR(3)
KeySequence: NUMBER(8,2)
FK ImputationMethodOID: CHAR(128)
Role: CHAR(128)
FK RoleCodeListOID: CHAR(128)
*FK FK_ValueLists: CHAR(128)
+ FK_ValueListItem_ImputationMet(ImputationMethodOID)
+ FK_ValueListItemRefs_ItemDefs(ItemOID)
+ FK_ValueListItemRef_ValueLists(FK_ValueLists)
+ FK_ValueListItemRefs_CodeLists(RoleCodeListOID)

This metadata can be specified by the user in a **source_values** dataset in a similar way as the **source_columns** dataset. The data **source_values** set has the following structure:

```
create table srcmdata.source_values(label='Source Value Metadata') (
   SASref char(8) format=$8. label='SASreferences sourcedata libref',
   table char(32) format=$32. label='Table Name',
   column char(32) format=$32. label='Column Name',
   value char(32) format=$32.  label='Column Value',
   label char(200) format=$200. label='Column Description',
   order num format=8. label='Column Order',
   type char(1) format=$1. label='Column Type',
   length num format=8. label='Column Length',
   displayformat char(32) format=$32. label='Display Format',
   xmldatatype char(8) format=$8. label='XML Data Type',
   xmlcodelist char(32) format=$32. label='SAS Format/XML Codelist',
   core char(10) format=$10. label='Column Required or Optional',
   origin char(40) format=$40. label='Column Origin',
   role char(128) format=$128. label='Column Role',
   term char(80) format=$80. label='Controlled Term or Format in Standard',
   algorithm char(2000) format=$2000. label='Computational Algorithm or Method',
   qualifiers char(200) format=$200. label='Column qualifiers (space delimited)',
   standard char(20) format=$20. label='Name of Standard',
   standardversion char(20) format=$20. label='Version of Standard',
   standardref char(200) format=$200. label='Associated reference(s) in Standard',
   comment char(1000) format=$1000. label='Comment'
);
```

The only difference between the source_columns and the source_values data set is the extra 'value' column in the source_values dataset. Figure 15 shows an example of the **source_values** dataset.

Figure 15: Sample source_values dataset



VIEWTABLE: TMP1.source_values (Source Value Metadata)

	SASref	table	column	value	label	order	type	length	displayformat	xmldatatype	xmlcodelist	origin	algorithm
1	SRCDATA	EG	EGTESTCD	PRI	PR Interval	1	N	8		integer		eDT	
2	SRCDATA	EG	EGTESTCD	QRSI	QRS Interval	2	N	8		integer		eDT	
3	SRCDATA	EG	EGTESTCD	QTI	QT Interval	3	N	8		integer		eDT	
4	SRCDATA	EG	EGTESTCD	QTCB	QTcB - Bazett's Correction Formula	4	N	5	5.1	float		eDT	QTcB = QT interval / square root of (60 / heart rate)
5	SRCDATA	EG	EGTESTCD	QTCF	QTcF - Fridericia's Correction Formula	5	N	5	5.1	float		eDT	QTcF = QT interval / cubic root of (60 / heart rate)
6	SRCDATA	VS	VSTESTCD	HEIGHT	Height	1	N	8	3.0	float		CRF Page 11	
7	SRCDATA	VS	VSTESTCD	WEIGHT	Weight	2	N	8	5.1	float		CRF Page 11	
8	SRCDATA	VS	VSTESTCD	DIABP	Diastolic Blood Pressure	3	N	8		integer		CRF Page 11	
9	SRCDATA	VS	VSTESTCD	FRMSIZE	Frame Size	4	C	6		text	SIZE	CRF Page 11	
10	SRCDATA	VS	VSTESTCD	HR	Heart Rate	5	N	8		integer		CRF Page 11	
11	SRCDATA	VS	VSTESTCD	PULSEPR	Pulse Pressure	6	N	8		integer		CRF Page 11	
12	SRCDATA	VS	VSTESTCD	SYSBP	Systolic Blood Pressure	7	N	8		integer		CRF Page 11	

The SAS program in Appendix 3 shows example SAS code to create the source_values data set. The code in Appendix 4 is an example that can be used to transform the source_values dataset to the SAS representation of the define.xml.

DOCUMENT METADATA

The define.xml supports metadata for additional documents that can help a reviewer understand the submission: Annotated CRFs and Supplemental Data Definitions.

An example of how this metadata looks in XML:

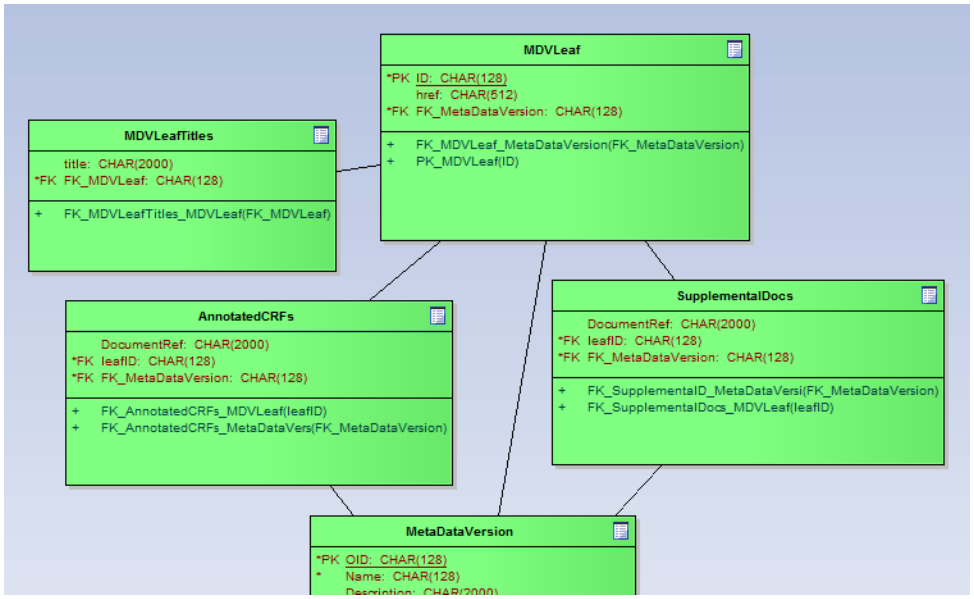
```
<def:AnnotatedCRF>
  <def:DocumentRef leafID="lf.blankcrf"/>
</def:AnnotatedCRF>
<def:AnnotatedCRF>
  <def:DocumentRef leafID="lf.blankcrf2"></def:DocumentRef>
</def:AnnotatedCRF>
<def:SupplementalDoc>
  <def:DocumentRef leafID="lf.supplementaldoc"/>
</def:SupplementalDoc>

<def:leaf ID="lf.blankcrf" xlink:href="blankcrf.pdf">
  <def:title>Annotated Case Report Forms</def:title>
</def:leaf>
<def:leaf ID="lf.blankcrf2" xlink:href="blankcrf-add.pdf">
  <def:title>Annotated Case Report Forms (Addendum)</def:title>
</def:leaf>
<def:leaf ID="lf.supplementaldoc" xlink:href="supplementaldoc.pdf">
  <def:title>Reviewers Guide</def:title>
</def:leaf>
```

To be able to represent this information in the SAS CRT-DDS datasets we will need to add information to several datasets. The following datasets are involved:

AnnotatedCRF	contains document references to the annotated case report form
SupplementalDocs	contains document references to the supplemental documentation
MDVLeaf	contains the href link for each of the documents listed in the AnnotatedCRF and SupplementalDocs tables.
MDVLeafTitles	contains a descriptive title for each MDVLeaf item.

Figure 15 shows the relation between these tables in the SAS representation of the define.xml.

Figure 15: CRT-DDS Datasets related to document metadata

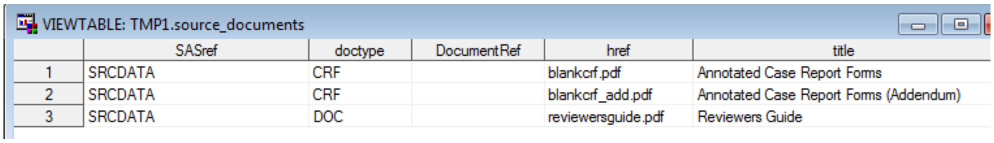

MDVLeaf
*PK ID: CHAR(128)
href: CHAR(512)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_MDVLeaf_MetaDataVersion(FK_MetaDataVersion)
+ PK_MDVLeaf(ID)
MDVLeafTitles
title: CHAR(2000)
*FK FK_MDVLeaf: CHAR(128)
+ FK_MDVLeafTitles_MDVLeaf(FK_MDVLeaf)
AnnotatedCRFs
DocumentRef: CHAR(2000)
*FK leafID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_AnnotatedCRFs_MDVLeaf(leafID)
+ FK_AnnotatedCRFs_MetaDataVers(FK_MetaDataVersion)
SupplementalDocs
DocumentRef: CHAR(2000)
*FK leafID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_SupplementalD_MetaDataVersi(FK_MetaDataVersion)
+ FK_SupplementalDocs_MDVLeaf(leafID)
MetaDataVersion
*PK OID: CHAR(128)
* Name: CHAR(128)
Description: CHAR(2000)

This metadata can be specified by the user in a **source_documents** dataset, which has the following structure:

```
create table srcmdata.source_documents (
   SASref        char(8)     format=$8.     label='SASreferences sourcedata libref',
   doctype       char(10)    format=$10.
     label='Document Type (CRF/DOC)',
   DocumentRef   char(2000)  format=$2000.
     label='The referenced Annotated CRF document',
   href          char(512)   format=$512.
     label='The pathname and filename of the target dataset relative to the define.xml',
   title         char(2000)  format=$2000.
     label='Meaningful description, label, or location of the document leaf'
   );
```

Figure 17 gives an example of the source_documents dataset.

Figure 17: Sample source_documents dataset



	SASref	doctype	DocumentRef	href	title
1	SRCDATA	CRF		blankcrf.pdf	Annotated Case Report Forms
2	SRCDATA	CRF		blankcrf_add.pdf	Annotated Case Report Forms (Addendum)
3	SRCDATA	DOC		reviewersguide.pdf	Reviewers Guide

The SAS program in Appendix 5 shows example SAS code to create the source_documents data set. The SAS code in Appendix 6 is an example that can be used to transform the source_documents dataset to the SAS representation of the define.xml.

CONCLUSION

The SAS Clinical Standards Toolkit provides metadata and macros to work with the CRT-DDS dataset representation of the define.xml. The Toolkit is able to transform metadata into a define.xml that can be validated for both internal consistency and consistency with the xml CRT-DDS schema.

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. Eric van der Vlist, 2002, XML Schema, *The W3C’s Object-Oriented Descriptions for XML* (O’Reilly and Associates)
12. Dough Tidwell, 2001, XSLT, *Mastering XML Transformations*. (O’Reilly and Associates)
13. XML Schema Validation for Define.xml, Version 1.0, November 30, 2009 (http://www.cdisc.org/define-xml)
14. SAS Institute Inc. 2011. SAS® Clinical Standards Toolkit 1.4: User's Guide . Cary, NC: SAS Institute Inc. (http://support.sas.com/documentation/onlinedoc/clinical/index.html)
15. Gene Lightfoot (2010). Implementing, Managing, and Validating a Clinical Standard Using SAS Clinical Standards Toolkit 1.3. Proceedings of the 6th Pharmaceutical Users Software Exchange (PhUSE 2010, Berlin, Germany) (http://www.lexjansen.com/phuse/2010/cd/CD14.pdf)
16. Julie Maddox, Mark Lambrecht (2010). define.xml – Tips and Techniques for creating CRT-DDS. Proceedings of the 6th Pharmaceutical Users Software Exchange (PhUSE 2010, Berlin, Germany) (http://www.lexjansen.com/phuse/2010/cd/CD09.pdf)
17. Lex Jansen (2011). Using the SAS® Clinical Standards Toolkit for define.xml creation. Proceedings of the Pharmaceutical Industry SAS® Users Group (PharmaSUG 2011, Nashville, TN) (http://www.lexjansen.com/pharmasug/2011/sas/pharmasug-2011-sas-hw02.pdf)

CONTACT INFORMATION

Lex Jansen,
SAS Institute Inc.
919-531-9860
Email: mailto:lex.jansen@sas.com

APPENDIX 1

Complete SAS Data Sets representation of CRT-DDS version 1.0 (define.xml)

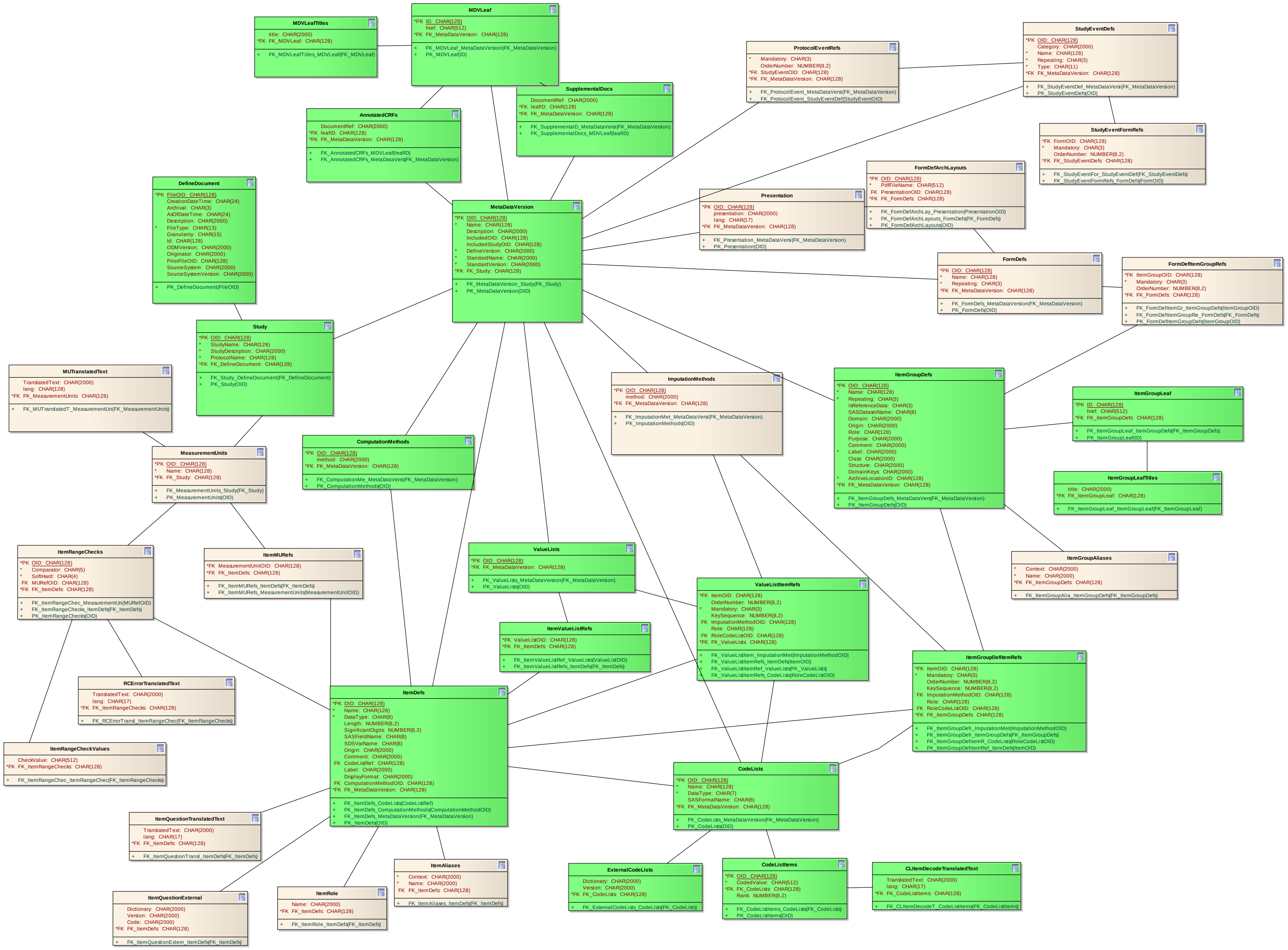
The data sets in bold are typically used in a define.xml meant for submission.

DefineDocument	contains information specifically about he define.xml file, including description and file creation date/time
Study	contains descriptive information about each study included in the define.xml file
MetaDataVersion	contains information about the data standards used in the study.
AnnotatedCRF	contains document references to the annotated case report form
SupplementalDocs	contains document references to the supplemental documentation
MDVLeaf	contains the href link for each of the documents listed in the AnnotatedCRF and SupplementalDocs tables.
MDVLeafTitles	contains a descriptive title for each MDVLeaf item.
ComputationMethods	contains description of computation methods
ValueLists	contains id of value lists
ValueListItemRefs	contains id of each item in a value list – associates each value list item to a row in the ItemDefs table
ItemValueListRefs	associates each value list item to a row in the ItemDefs table
ProtocolEventRefs	contains a list of study events and the order they occur
StudyEventDefs	contains descriptive information on the study events
StudyEventFormRefs	associates a particular form to a specific study event
FormDefs	contains a list of forms for the study
FormDefItemGroupRefs	associates a table with a form definition
FormDefArchLayouts	contains document references for the form layouts
Presentation	contains how information on the study is presented
ItemGroupDefs	contains metadata for each of the tables included in the study
ItemGroupAliases	contains context and alias names for the table
ItemGroupLeaf	contains href location to the specified table
ItemGroupLeafTitles	contains descriptive title to table location
ItemGroupDefItemRefs	associates the columns to the study tables
ItemDefs	contains the column metadata
ItemQuestionTranslatedText	contains translated text of questions
ItemQuestionExternal	contains the name and version of external question dictionaries
ItemMURefs	associates a column to a measurement unit
ItemRangeChecks	contains information on range checks for data in a column
ItemRangeCheckValues	contains information on range checks for data in a column
RCErrorTranslatedText	error message to be used when a range check is violated
ItemRole	contains the role of the column
ItemAliases	contains context and alias names for the column
MeasurementUnits	contains metadata about measurement units
MUTranslatedText	contains translated text of measurement units
CodeLists	contains name, type and SAS Format name of code lists
ExternalCodeLists	contains name and versions of external code lists
CodeListItems	contains a list of coded items for each code list
CLItemDecodeTranslatedText	contains the full text for a coded item in the specified language
ImputationMethods	contains description of imputation methods

APPENDIX 2

Data model for CRT-DDS (define.xml)

The tables in green are typically used in submission.



MDVLeafTitles
title: CHAR(2000)
*FK FK_MDVLeaf: CHAR(128)
+ FK_MDVLeafTitles_MDVLeaf(FK_MDVLeaf)
MDVLeaf
*PK ID: CHAR(128)
href: CHAR(512)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_MDVLeaf_MetaDataVersion(FK_MetaDataVersion)
+ PK_MDVLeaf(ID)
SupplementalDocs
DocumentRef: CHAR(2000)
*FK leafID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_SupplementalD_MetaDataVers(FK_MetaDataVersion)
+ FK_SupplementalDocs_MDVLeaf(leafID)
AnnotatedCRFs
DocumentRef: CHAR(2000)
*FK leafID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_AnnotatedCRFs_MDVLeaf(leafID)
+ FK_AnnotatedCRFs_MetaDataVers(FK_MetaDataVersion)
ProtocolEventRefs
* Mandatory: CHAR(3)
OrderNumber: NUMBER(8,2)
*FK StudyEventOID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_ProtocolEvent_MetaDataVers(FK_MetaDataVersion)
+ FK_ProtocolEvent_StudyEventDef(StudyEventOID)
StudyEventDefs
*PK OID: CHAR(128)
Category: CHAR(2000)
* Name: CHAR(128)
* Repeating: CHAR(3)
* Type: CHAR(11)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_StudyEventDef_MetaDataVers(FK_MetaDataVersion)
+ PK_StudyEventDefs(OID)
StudyEventFormRefs
*FK FormOID: CHAR(128)
* Mandatory: CHAR(3)
OrderNumber: NUMBER(8,2)
*FK FK_StudyEventDefs: CHAR(128)
+ FK_StudyEventFor_StudyEventDef(FK_StudyEventDefs)
+ FK_StudyEventFormRefs_FormDef(FormOID)
DefineDocument
*PK FileOID: CHAR(128)
CreationDateTime: CHAR(24)
Archival: CHAR(3)
AsOfDateTime: CHAR(24)
Description: CHAR(2000)
* FileType: CHAR(13)
Granularity: CHAR(15)
Id: CHAR(128)
ODMVersion: CHAR(2000)
Originator: CHAR(2000)
PriorFileOID: CHAR(128)
SourceSystem: CHAR(2000)
SourceSystemVersion: CHAR(2000)
+ PK_DefineDocument(FileOID)
MetaDataVersion
*PK OID: CHAR(128)
* Name: CHAR(128)
Description: CHAR(2000)
IncludedOID: CHAR(128)
IncludedStudyOID: CHAR(128)
* DefineVersion: CHAR(2000)
* StandardName: CHAR(2000)
* StandardVersion: CHAR(2000)
*FK FK_Study: CHAR(128)
+ FK_MetaDataVersion_Study(FK_Study)
+ PK_MetaDataVersion(OID)
FormDefArchLayouts
*PK OID: CHAR(128)
* PdfFileName: CHAR(512)
FK PresentationOID: CHAR(128)
*FK FK_FormDefs: CHAR(128)
+ FK_FormDefArchLay_Presentation(PresentationOID)
+ FK_FormDefArchLayouts_FormDefs(FK_FormDefs)
+ PK_FormDefArchLayouts(OID)
Presentation
*PK OID: CHAR(128)
presentation: CHAR(2000)
lang: CHAR(17)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_Presentation_MetaDataVers(FK_MetaDataVersion)
+ PK_Presentation(OID)
FormDefs
*PK OID: CHAR(128)
* Name: CHAR(128)
* Repeating: CHAR(3)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_FormDefs_MetaDataVersion(FK_MetaDataVersion)
+ PK_FormDefs(OID)
FormDefItemGroupRefs
*FK ItemGroupOID: CHAR(128)
* Mandatory: CHAR(3)
OrderNumber: NUMBER(8,2)
*FK FK_FormDefs: CHAR(128)
+ FK_FormDefItemGr_ItemGroupDefs(ItemGroupOID)
+ FK_FormDefItemGroupRe_FormDefs(FK_FormDefs)
+ PK_FormDefItemGroupDefs(ItemGroupOID)
Study
*PK OID: CHAR(128)
* StudyName: CHAR(128)
* StudyDescription: CHAR(2000)
* ProtocolName: CHAR(128)
*FK FK_DefineDocument: CHAR(128)
+ FK_Study_DefineDocument(FK_DefineDocument)
+ PK_Study(OID)
MUTranslatedText
TranslatedText: CHAR(2000)
lang: CHAR(128)
*FK FK_MeasurementUnits: CHAR(128)
+ FK_MUTranslatedT_MeasurementUn(FK_MeasurementUnits)
ImputationMethods
*PK OID: CHAR(128)
method: CHAR(2000)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_ImputationMet_MetaDataVers(FK_MetaDataVersion)
+ PK_ImputationMethods(OID)
ItemGroupDefs
*PK OID: CHAR(128)
* Name: CHAR(128)
* Repeating: CHAR(3)
IsReferenceData: CHAR(3)
SASDatasetName: CHAR(8)
Domain: CHAR(2000)
Origin: CHAR(2000)
Role: CHAR(128)
Purpose: CHAR(2000)
Comment: CHAR(2000)
* Label: CHAR(2000)
Class: CHAR(2000)
Structure: CHAR(2000)
DomainKeys: CHAR(2000)
* ArchiveLocationID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_ItemGroupDefs_MetaDataVers(FK_MetaDataVersion)
+ PK_ItemGroupDefs(OID)
ItemGroupLeaf
*PK ID: CHAR(128)
* href: CHAR(512)
*FK FK_ItemGroupDefs: CHAR(128)
+ FK_ItemGroupLeaf_ItemGroupDefs(FK_ItemGroupDefs)
+ PK_ItemGroupLeaf(ID)
ItemGroupLeafTitles
title: CHAR(2000)
*FK FK_ItemGroupLeaf: CHAR(128)
+ FK_ItemGroupLeaf_ItemGroupLeaf(FK_ItemGroupLeaf)
MeasurementUnits
*PK OID: CHAR(128)
* Name: CHAR(128)
*FK FK_Study: CHAR(128)
+ FK_MeasurementUnits_Study(FK_Study)
+ PK_MeasurementUnits(OID)
ComputationMethods
*PK OID: CHAR(128)
method: CHAR(2000)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_ComputationMe_MetaDataVers(FK_MetaDataVersion)
+ PK_ComputationMethods(OID)
ItemRangeChecks
*PK OID: CHAR(128)
* Comparator: CHAR(5)
* SoftHard: CHAR(4)
FK MURefOID: CHAR(128)
*FK FK_ItemDefs: CHAR(128)
+ FK_ItemRangeChec_MeasurementUn(MURefOID)
+ FK_ItemRangeChecks_ItemDefs(FK_ItemDefs)
+ PK_ItemRangeChecks(OID)
ItemMURefs
*FK MeasurementUnitOID: CHAR(128)
*FK FK_ItemDefs: CHAR(128)
+ FK_ItemMURefs_ItemDefs(FK_ItemDefs)
+ FK_ItemMURefs_MeasurementUnits(MeasurementUnitOID)
ValueLists
*PK OID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_ValueLists_MetaDataVersion(FK_MetaDataVersion)
+ PK_ValueLists(OID)
ValueListItemRefs
*FK ItemOID: CHAR(128)
OrderNumber: NUMBER(8,2)
* Mandatory: CHAR(3)
KeySequence: NUMBER(8,2)
FK ImputationMethodOID: CHAR(128)
Role: CHAR(128)
FK RoleCodeListOID: CHAR(128)
*FK FK_ValueLists: CHAR(128)
+ FK_ValueListItem_ImputationMet(ImputationMethodOID)
+ FK_ValueListItemRefs_ItemDefs(ItemOID)
+ FK_ValueListItemRef_ValueLists(FK_ValueLists)
+ FK_ValueListItemRefs_CodeLists(RoleCodeListOID)
ItemGroupAliases
* Context: CHAR(2000)
* Name: CHAR(2000)
*FK FK_ItemGroupDefs: CHAR(128)
+ FK_ItemGroupAlia_ItemGroupDefs(FK_ItemGroupDefs)
ItemValueListRefs
*FK ValueListOID: CHAR(128)
*FK FK_ItemDefs: CHAR(128)
+ FK_ItemValueListRef_ValueLists(ValueListOID)
+ FK_ItemValueListRefs_ItemDefs(FK_ItemDefs)
RCErrorTranslatedText
TranslatedText: CHAR(2000)
lang: CHAR(17)
*FK FK_ItemRangeChecks: CHAR(128)
+ FK_RCErrorTransl_ItemRangeChec(FK_ItemRangeChecks)
ItemGroupDefItemRefs
*FK ItemOID: CHAR(128)
* Mandatory: CHAR(3)
KeySequence: NUMBER(8,2)
OrderNumber: NUMBER(8,2)
FK ImputationMethodOID: CHAR(128)
Role: CHAR(128)
FK RoleCodeListOID: CHAR(128)
*FK FK_ItemGroupDefs: CHAR(128)
+ FK_ItemGroupDefI_ImputationMet(ImputationMethodOID)
+ FK_ItemGroupDefI_ItemGroupDefs(FK_ItemGroupDefs)
+ FK_ItemGroupDefItemR_CodeLists(RoleCodeListOID)
+ FK_ItemGroupDefItemRef_ItemDefs(ItemOID)
ItemDefs
*PK OID: CHAR(128)
* Name: CHAR(128)
* DataType: CHAR(8)
Length: NUMBER(8,2)
SignificantDigits: NUMBER(8,2)
SASFieldName: CHAR(8)
SDSVarName: CHAR(8)
Origin: CHAR(2000)
Comment: CHAR(2000)
FK CodeListRef: CHAR(128)
Label: CHAR(2000)
DisplayFormat: CHAR(2000)
FK ComputationMethodOID: CHAR(128)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_ItemDefs_CodeLists(CodeListRef)
+ FK_ItemDefs_ComputationMethods(ComputationMethodOID)
+ FK_ItemDefs_MetaDataVersion(FK_MetaDataVersion)
+ PK_ItemDefs(OID)
ItemRangeCheckValues
CheckValue: CHAR(512)
*FK FK_ItemRangeChecks: CHAR(128)
+ FK_ItemRangeChec_ItemRangeChec(FK_ItemRangeChecks)
CodeLists
*PK OID: CHAR(128)
* Name: CHAR(128)
* DataType: CHAR(7)
SASFormatName: CHAR(8)
*FK FK_MetaDataVersion: CHAR(128)
+ FK_CodeLists_MetaDataVersion(FK_MetaDataVersion)
+ PK_CodeLists(OID)
ItemQuestionTranslatedText
TranslatedText: CHAR(2000)
lang: CHAR(17)
*FK FK_ItemDefs: CHAR(128)
+ FK_ItemQuestionTransl_ItemDefs(FK_ItemDefs)
ItemAliases
* Context: CHAR(2000)
* Name: CHAR(2000)
FK FK_ItemDefs: CHAR(128)
+ FK_ItemAliases_ItemDefs(FK_ItemDefs)
ExternalCodeLists
Dictionary: CHAR(2000)
Version: CHAR(2000)
*FK FK_CodeLists: CHAR(128)
+ FK_ExternalCodeLists_CodeLists(FK_CodeLists)
CodeListItems
*PK OID: CHAR(128)
* CodedValue: CHAR(512)
*FK FK_CodeLists: CHAR(128)
Rank: NUMBER(8,2)
+ FK_CodeListItems_CodeLists(FK_CodeLists)
+ PK_CodeListItems(OID)
CLItemDecodeTranslatedText
TranslatedText: CHAR(2000)
lang: CHAR(17)
*FK FK_CodeListItems: CHAR(128)
+ FK_CLItemDecodeT_CodeListItems(FK_CodeListItems)
ItemRole
Name: CHAR(2000)
*FK FK_ItemDefs: CHAR(128)
+ FK_ItemRole_ItemDefs(FK_ItemDefs)
ItemQuestionExternal
Dictionary: CHAR(2000)
Version: CHAR(2000)
Code: CHAR(2000)
*FK FK_ItemDefs: CHAR(128)
+ FK_ItemQuestionExtern_ItemDefs(FK_ItemDefs)

APPENDIX 3

Example SAS code to create the source_values data set

```sas
proc sql;
create table srcmdata.source_values(label='Source Value Metadata') (
   SASref char(8) format=$8. label='SASreferences sourcedata libref',
   table char(32) format=$32. label='Table Name',
   column char(32) format=$32. label='Column Name',
   value char(32) format=$32.  label='Column Value',
   label char(200) format=$200. label='Column Description',
   order num format=8. label='Column Order',
   type char(1) format=$1. label='Column Type',
   length num format=8. label='Column Length',
   displayformat char(32) format=$32. label='Display Format',
   xmldatatype char(8) format=$8. label='XML Data Type',
   xmlcodelist char(32) format=$32. label='SAS Format/XML Codelist',
   core char(10) format=$10. label='Column Required or Optional',
   origin char(40) format=$40. label='Column Origin',
   role char(128) format=$128. label='Column Role',
   term char(80) format=$80. label='Controlled Term or Format in Standard',
   algorithm char(2000) format=$2000. label='Computational Algorithm or Method',
   qualifiers char(200) format=$200. label='Column qualifiers (space delimited)',
   standard char(20) format=$20. label='Name of Standard',
   standardversion char(20) format=$20. label='Version of Standard',
   standardref char(200) format=$200. label='Associated reference(s) in Standard',
   comment char(1000) format=$1000. label='Comment'
   );
  insert into  srcmdata.source_values
    values("SRCDATA","EG","EGTESTCD","PRI","PR Interval",1,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","Just a comment")
    values("SRCDATA","EG","EGTESTCD","QRSI","QRS Interval",2,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","EG","EGTESTCD","QTI","QT Interval",3,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","EG","EGTESTCD","QTCB","QTcB - Bazett's Correction Formula",4,
           "",.,"","","","","","","",
           "QTcB = QT interval / square root of (60 / heart rate)","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","EG","EGTESTCD","QTCF","QTcF - Fridericia's Correction Formula",5,
           "",.,"","","","","","","",
           "QTcF = QT interval / cubic root of (60 / heart rate)","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","IE","IETESTCD","INCL02","Acceptable chest X-Ray",1,"",.,"","","","","","","",
           "","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","IE","IETESTCD","INCL10","Systolic BP > 180",2,"",.,"","","","","","","",
           "","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","LB","LBTESTCD","CALCIUM","Calcium",1,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","LB","LBTESTCD","CHLORIDE","Chloride",2,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","LB","LBTESTCD","POTASS","Potassium",3,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","LB","LBTESTCD","SODIUM","Sodium",4,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","PE","PETESTCD","CARDIO","Cardiovascular",1,"",.,"","","","","","","",
           "","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","PE","PETESTCD","ENT","Ear/Nose/Throat",2,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","PE","PETESTCD","RESP","Respiratory",3,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","PE","PETESTCD","SKIN","Skin",4,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","SC","SCTESTCD","INITIALS","Initials",1,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","SC","SCTESTCD","RACEOTH","Race, Other",2,"",.,"","","","","","","",
           "","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","SUPPAE","QNAM","AECONIA",
           "Interaction between add'l and trial meds",1,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","SUPPAE","QNAM","AETRTEM","Treatment emergent",2,"",.,"","","","","","","",
           "","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","TI","IETESTCD","EXCL01",
           "Systolic BP > 180",4,"",.,"","","","","","","","","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","TI","IETESTCD","EXCL02",
           "Diastolic BP > 120",3,"",.,"","","","","","","","","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","TI","IETESTCD","INCL01",
           "Age between 18 and 70",2,"",.,"","","","","","","","","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","TI","IETESTCD","INCL02",
           "Acceptable chest X-Ray",1,"",.,"","","","","","","","","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","VS","VSTESTCD","HEIGHT","Height",1,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","VS","VSTESTCD","WEIGHT","Weight",2,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","VS","VSTESTCD","DIABP",
           "Diastolic Blood Pressure",3,"",.,"","","","","","","","","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","VS","VSTESTCD","FRMSIZE",
           "Frame Size",4,"",.,"","","","","","","","","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","VS","VSTESTCD","HR","Heart Rate",5,"",.,"","","","","","","","","",
           "CDISC-SDTM","3.1.2","","")
    values("SRCDATA","VS","VSTESTCD","PULSEPR",
           "Pulse Pressure",6,"",.,"","","","","","","","","","CDISC-SDTM","3.1.2","","")
    values("SRCDATA","VS","VSTESTCD","SYSBP",
           "Systolic Blood Pressure",7,"",.,"","","","","","","","","","CDISC-SDTM","3.1.2","","")
    ;
  quit;

  data srcmdata.source_values(label='Source Value Metadata');
   set srcmdata.source_values;

   select (value);
      when ('INCL02', 'INCL10', 'INCL01', 'INCL02', 'EXCL01', 'EXCL02',
            'AETRTEM', 'AECONIA') xmlcodelist="NY";
      when ('FRMSIZE') xmlcodelist="SIZE";
      otherwise;
   end;

   select (value);
      when ('FRMSIZE', 'INCL02', 'INCL01', 'INCL10', 'INCL02', 'EXCL01', 'EXCL02',
            'AETRTEM', 'AECONIA', 'INITIALS', 'RACEOTH') do; xmldatatype='text';  type='C'; end;
      when ('CALCIUM', 'POTASS') do; xmldatatype='float'; type='N'; end;
      when ('QTCB', 'QTCF', 'HEIGHT', 'WEIGHT') do; xmldatatype='float'; type='N'; end;
      otherwise do; xmldatatype='integer'; type='N'; end;
   end;
   select (value);
     when ('INCL02','INCL10', 'INCL01', 'INCL02', 'EXCL01', 'EXCL02', 'AETRTEM', 'AECONIA')
length=2;
     when ('INITIALS') length=3;
     when ('RACEOTH') length=10;
     when ('FRMSIZE') length=6;
     when ('QTCB', 'QTCF') length=5;
     otherwise length=8;
   end;
   select (value);
      when ('POTASS') displayformat='8.2';
      when ('CALCIUM') displayformat='8.2';
      when ('QTCB', 'QTCF') displayformat='5.1';
      when ('HEIGHT') displayformat='3.0';
      when ('WEIGHT') displayformat='5.1';
      otherwise;
   end;

   select (table);
     when ('EG') Origin="eDT";
     when ('LB') Origin="eDT";
     when ('VS') Origin="CRF Page 11";
     when ('IE') Origin="CRF Page 5";
     when ('PE') Origin="CRF Page 10";
     when ('SC') Origin="CRF Page 6";
     when ('SUPPAE') Origin="CRF Page 17";
     when ('TI') Origin="CRF Page 14";
     otherwise;
   end;

   if xmldatatype='text' then qualifiers='UPPERCASE';
   core = 'Perm';
  run;
```

APPENDIX 4

Example SAS code to create the data sets that are related to Value Level metadata

```
*Lookup OID for the SDTM 3.1.2 standard in MetaDataVersion;
proc sql noprint;
  select OID into :mdv_oid from srcdata.MetaDataVersion
  where name="CDISC-SDTM 3.1.2";
quit;

%let ds1=_srcval1;
%let ds2=_srcval2;
%let ds3=_srcval3;
%let ds4=_srcval4;
%let ds5=_srcval5;
%let ds6=_srcval6;

%* Create data set with table, column and ItemDef/@OID *;
proc sql;
  create table &ds1 as
  select a.*, b.Name as column
  from
    (select a.Name as table, b.ItemOiD as FK_ItemDefs
     from srcdata.ItemGroupDefs a left join srcdata.ItemGroupDefItemRefs b
     on (a.oid = b.fk_itemgroupdefs) and (a.fk_metadataversion = "&mdv_oid")) a
   left join srcdata.ItemDefs b
   on (a.FK_ItemDefs =b.OID) and (b.fk_metadataversion = "&mdv_oid");
quit;

%* Create fk_metadataversion and ValueListDef/@OID *;
data &ds2;
length fk_metadataversion ValueListOID ItemOID ComputationMethodOID $128 Mandatory $3;
  set sampdata.source_values;
  fk_metadataversion="&mdv_oid";
  ValueListOID=cats("VL.", table, ".", column);
  ItemOID=catx(".", "IT", table, column, value);
  if not missing(algorithm) then ComputationMethodOID=catx(".", "CM", table, column, value);
  Mandatory = ifc(upcase(core)='REQ','Yes','No', 'No');
run;

%* join to get ItemOID and the CodeList OID *;
proc sql;
  create table &ds3 as
  select a.*, b.OID as CodeListRef from
    (select a.*, b.FK_ItemDefs
     from &ds2 a left join &ds1 b
     on (a.table = b.table) and (a.column = b.column)) a
  left join
     srcdata.CodeLists b
  on (a.xmlcodelist =b.Name) and (b.fk_metadataversion = "&mdv_oid")
  order by ItemOID, order;
quit;

%*****************************************************************************;
%* Create output data sets                                                   *;
%*****************************************************************************;
proc sort data=&ds3 out=&ds4(keep=ValueListOID fk_metadataversion) nodupkey;
by ValueListOID;
run;

data srcdata.ValueLists;
  set srcdata.ValueLists &ds4(rename=(ValueListOID=OID));
run;
%*****************************************************************************;

proc sort data=&ds3 out=&ds5(keep=ValueListOID FK_ItemDefs) nodupkey;
by ValueListOID FK_ItemDefs;
run;

data srcdata.ItemValueListRefs;
  set srcdata.ItemValueListRefs &ds5;
run;
%*****************************************************************************;

data srcdata.ValueListItemRefs
       (keep=ItemOID OrderNumber Mandatory KeySequence
             ImputationMethodOID Role RoleCodeListOID FK_ValueLists);
  set srcdata.ValueListItemRefs
      &ds3(rename=(ValueListOID=FK_ValueLists order=OrderNumber));
run;

%*****************************************************************************;

%* add to existing ItemDefs;
data srcdata.ItemDefs;
  set srcdata.ItemDefs
      &ds3(drop=SASRef FK_ItemDefs ValueListOID xmlcodelist Mandatory SASref algorithm table column
order type core role term qualifiers standard standardversion standardref
           rename=(value=Name ItemOID=OID xmldatatype=DataType));
run;

%*****************************************************************************;

proc sort data=&ds3 out=&ds6(keep=fk_metadataversion ComputationMethodOID algorithm where=(not
missing(algorithm)));
by ComputationMethodOID;
run;

%* add to existing ComputationMethods;
data srcdata.ComputationMethods;
 set srcdata.ComputationMethods &ds6(rename=(ComputationMethodOID=OID algorithm=Method));
run;
```

APPENDIX 5

Example SAS code to create the source_documents data set

```
proc sql;
  create table srcmdata.source_documents (
     SASref       char(8)    format=$8.    label='SASreferences sourcedata libref',
     doctype      char(10)   format=$10.   label='Document Type (CRF/DOC)',
     DocumentRef  char(2000) format=$2000. label='The referenced Annotated CRF document',
     href         char(512)  format=$512.
       label='The pathname and filename of the target dataset relative to the define.xml',
     title        char(2000) format=$2000.
       label='Meaningful description, label, or location of the document leaf'
     );
   /*                  doctype DocumentRef href                  title */
  insert into  srcmdata.source_documents
    values("SRCDATA", "CRF",   "",         "blankcrf.pdf",       "Annotated Case Report Forms")
    values("SRCDATA", "CRF",   "",         "blankcrf_add.pdf",   "Annotated Case Report Forms
(Addendum)")
    values("SRCDATA", "DOC",   "",         "reviewersguide.pdf", "Reviewers Guide")
    ;
quit;
```

APPENDIX 6

Example SAS code to create the data sets that are related to Document metadata (Annotated CRF and Supplemental Data Definitions)

```
*Lookup OID for the SDTM 3.1.2 standard in MetaDataVersion;
proc sql noprint;
  select OID into :mdv_oid from srcdata.MetaDataVersion
  where name="CDISC-SDTM 3.1.2";
quit;

proc sort data=sampdata.source_documents out=sourcedocs;
by SASRef doctype href;
run;

data sourcedocs;
  retain _cstcounter;
  length fk_metadataversion $128 leafid $64 _cstcounter 8;
  set sourcedocs;
  by SASRef doctype href;
  fk_metadataversion=strip("&mdv_oid");
  if first.doctype then do;
    _cstcounter=1;
    select (doctype);
      when ('CRF') leafid='lf.blankcrf';
      when ('DOC') leafid='lf.supplementaldoc';
      otherwise;
    end;
  end;
  else do;
    _cstcounter=sum(_cstcounter,1);
    if upcase(doctype) = 'CRF'
      then leafid=cats('lf.blankcrf', put(_cstcounter, best.));
    if upcase(doctype) = 'DOC'
      then leafid=cats('lf.supplementaldoc', put(_cstcounter, best.));
  end;
run;

%******************************************************************************;
%* Create output data sets                                                    *;
%******************************************************************************;
data srcdata.AnnotatedCRFs(keep=documentref leafid fk_metadataversion);
 set srcdata.AnnotatedCRFs
     sourcedocs(where=(upcase(doctype)="CRF"));
run;

data srcdata.SupplementalDocs(keep=documentref leafid fk_metadataversion);
 set srcdata.SupplementalDocs
     sourcedocs(where=(upcase(doctype)="DOC"));
run;

data srcdata.MDVLeaf(keep=id href fk_metadataversion);
 set srcdata.MDVLeaf
     sourcedocs(rename=(leafid=id) where=(upcase(doctype) in ("CRF" "DOC")));
run;

data srcdata.MDVLeafTitles(keep=title fk_mdvleaf);
 set srcdata.MDVLeafTitles
     sourcedocs(rename=(leafid=fk_mdvleaf) where=(upcase(doctype) in ("CRF" "DOC")));
run;
```