Paper AD10

Cost-Effective Solution to Automating the Creation of CDISC define.xml: Using SAS® Data _Null_ with Relational Data Model via SAS/AF® Application

Annie Guo, ICON Clinical Research, Redwood City, CA

ABSTRACT

CDISC define.xml is the Data Definition Document in XML format required for FDA electronic regulatory submission. It provides a description of the SAS data sets and the variables of each data set included in the submission. While most of the information in the document can be extracted from SAS Dictionary views, creating such a XML file may not be a typical task for a SAS programmer, and keeping it up to date with the constant changes to the study database is time consuming.

This paper presents a SAS/AF application that streamlines the data flow from the study database to define.xml. The backbone of the application is a relational data model with integrity constraints complying with the define.xml schema. The application serves as a user interface, enforces the data integrity constraints at the application level, and incorporates an external SAS program that consolidates the data in the relational database, and generates define.xml with DATA _NULL_ step and PUT statements. The result is a cost-effective tool that provides the benefits of a centralized database with easy maintenance, and automating the creation of define.xml for internal use as well as regulatory submission.

INTRODUCTION

The Food and Drug Administration is aggressively moving towards an electronic regulatory submissions environment. Each submission should be accompanied by a set of electronic common technical documents (eCTD). One of them is the Data Definition File, e.g., define.xml, which provides a table of contents for the data sets and a collection of data definition tables describing the variables of each data set in the submission.

The complete name for define.xml is Case Report Tabulation Data Definition Specification (CRT-DDS). It is developed by the Clinical Data Interchange Standards Consortium (CDISC) define.xml team. The specification of define.xml is based on the CDISC Operational Data Model (ODM), which is a specification of a standard XML schema.

As a SAS programmer to create define.xml, there are three challenges that we are faced with.

1. While most of the information about the study can be extracted from SAS Dictionary views, transferring the information to the define.xml format is not a typical task for a SAS programmer.

2. As the number of data sets and variables for the study increases, keeping the XML document up to date with the constant changes to the study database is time consuming.

3. Define.xml also requires external information such as the location and file name of the annotated CRF, which may exist outside the SAS system. The external information needs maintenance separately from the study database.

However, there are two factors to our advantage.

1. A XML file is text-based and follows a specific structure.

2. All sponsors and registered CDISC solution providers follow the same technical document for define.xml. That is the “Case Report Tabulation Data Definition Specification (define.xml)” released by the CDISC define.xml team. Most importantly FDA expects it to be in this format.

This paper presents an approach that streamlines the collection and maintenance of the information required for define.xml, and the creation of the XML document. In the following sections, first we talk about the structure of define.xml, and the parent-child relationships among the elements in the file. Then we propose a relational data model that fits the parent-child relationships and serves as a repository to store the information about the study. As

for the maintenance of the relational data model, a SAS/AF application is developed as the front-end of the data model, to enforce the integrity of the parent-child relationships. Finally, an external SAS program is incorporated into the SAS/AF application and generates the XML file automatically. That SAS program consolidates the data in the relational database, and utilizes DATA _NULL_ and PUT statements based on the specification of define.xml.

DEFINE.XML SCHEMA

What makes it possible to use DATA _NULL_ to generate define.xml is the fact that it is structured and text based. Think about how we use DATA _NULL_ to create a text file in CSV format. It is the same idea when it comes to a XML file. The texts in define.xml must comply with the syntax of the named objects in XML. For example, they should be surrounded by the open tag and the closing tag of a specific named object. Then the document along with its style sheet can display meaningful information to the reviewers.

ELEMENT

The define.xml schema contains the specification of named objects, called **ODM elements**. Examples of the required ODM elements are `ODM` for XML header, `Study` for the study in the submission, `MetaDataVersion` for the metadata version of the study, `ItemGroupDef` for each of the data sets in the submission, and `ItemDef` for each of the variables in the data sets. An ODM element must start with an open tag, and end with a closing tag, for example, `<ItemDef>` and `</ItemDef>`, which can be also in the form of `<ItemDef … />`.

ATTRIBUTE

An ODM element consists of one or more **attributes**. The texts of the attributes compose the data definition file displayed to the reviewers. Some attributes are required, for example, `OID`, the unique ID of an element. Some are optional, such as the `Purpose` of a data set for the element `ItemGroupDef`. The others are conditional, e.g., the `Length` of a character data type variable for the element `ItemDef`.

Figure 1 illustrates a basic layout of the elements in define.xml. The quoted texts in pink color are the values of the attributes. The unquoted ones in black color are free text, usually provided by the sponsor. All the other color texts and symbols are key words and case sensitive. Figures 2.1 and 2.2 are actual code in a define.xml document. Figures 3.1 and 3.2 illustrate what the code transforms to in a style sheet format when displayed to the reviewers.

Figure 1: Sample define.xml schema and eight sections of information

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="definel-0-0.xsl"?>
<ODM FileOID="PharmaABC04001" CreationDateTime="2006-09-04T12:21:20-08:00"  ...>
    <Study OID="PharmaABC04001">

        <GlobalVariables>
            <StudyName>PharmaABC04001</StudyName>
        </GlobalVariables>

        <MetaDataVersion OID="CDISC.SDTM.3.1.0" Name="Study PharmaABC04001, Data Definitions"...>

            <ItemGroupDef OID="PE" Name="PE" Purpose="Tabulation" ...>
                <ItemRef ItemOID="PE.PETESTCD" .../>
                <def:leaf ID="Location.PE" xlink:href="pe.xpt">
                  <def:title>crt/PharmaABC04001/pe.xpt</def:title>
                </def:leaf>
            </ItemGroupDef>

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

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

            <ItemDef OID="PE.PETESTCD" Name="PETESTCD" DataType="text"
                     def:ComoputationMethodOID="Comp.StudyDay">
                       <def:ValueListRef ValueListOID="AE.AESER.PETESTCD"/>
                <CodeListRef CodeListOID="YESNO"/>
            </ItemDef>

            <CodeList OID="YESNO" Name="YESNO" DataType=" "> </CodeList>

            <def:ValueListDef OID="AE.PETESTCD.PETESTCD"> </def:ValueListDef>

            <def:ComputationMethod OID="Comp.StudyDay">Exam Day - Screening Day + 1</def:ComputationMethod>

        </MetaDataVersion>
    </Study>
</ODM>
```

#1. XML Header, Study, MetaData Version

#2. Datasets

#3. Annotated CRF

#4. Supplemental Document

#5. Variables

#6. Code List

#7. Value List

#8. Computation Method

RELATIONSHIPS AMONG ELEMENTS

The ODM elements are associated with one another in a series of parent-child relationships. For example, `ItemGroupDef` is a child element of `MetaDataVersion`, and `ItemRef`, which holds a variable in the data set, is a child element of `ItemGroupDef`. A child element links to its parent element via the unique ID of the child element. For example, in Figure 1 the variable ID PE.PETESTCD is shared by the elements `ItemRef` in the Datasets section, and `ItemDef` in the Variables section. The purpose of the link is to group related information together, and/or insert a hypertext link between the parent element and the child element.

In Figure 2.1, the element `ItemGroupDef` defines the data set PE that contains a few child elements, i.e., variables. The detailed description of the mentioned variables is defined separately in the element `ItemDef` sections in Figure 2.2. The variables in the Datasets section and in the Variable sections are linked via the unique variable IDs. The variable IDs are the values of the attribute `ItemOID` in the Figure 2.1, and the values of the attribute `OID` in Figure 2.2. Because of the unique variable IDs, the table of contents for the data set PE in Figure 3.1 contains the hypertext link, Physical Examinations – PE, which points to the data definition table in Figure 3.2 for the variables in the data set PE. Similarly, in Figure 3.2 the format YESNO, the variable name PETESTCD, and the computation method StudyDay, which are in purple color, are hypertext links that point to the details in Figures 3.3, 3.4 and 3.5, respectively.

Figure 2.1: Table of Contents for PE data set in define.xml

```
<!-- ************************************************************ -->
<!-- ItemGroupDef (domain) information Section                    -->
<!-- ************************************************************ -->
<ItemGroupDef OID="PE"
  Name="PE"
  Repeating="Yes"
  IsReferenceData="No"
  Purpose="Tabulation"
  def:Label="Physical Examinations - PE"
  def:Structure="One record per subject per exam per visit"
  def:DomainKeys="STUDYID, USUBJID, VISITNUM, PETEST"
  def:Class="Findings"
  def:ArchiveLocationID="Location.PE">
  <!-- ********************************************************** -->
  <!-- Each variable is listed here for this ItemGroupDef (domain)  -->
  <!-- ********************************************************** -->
  <ItemRef ItemOID="STUDYID"     OrderNumber="01"  Mandatory="Yes/No"  Role="Identifier"/>
  <ItemRef ItemOID="DOMAIN"      OrderNumber="02"  Mandatory="Yes/No"  Role="Identifier"/>
  <ItemRef ItemOID="USUBJID"     OrderNumber="03"  Mandatory="Yes/No"  Role="Identifier"/>
  <ItemRef ItemOID="PE.PETESTCD" OrderNumber="07"  Mandatory="Yes/No"  Role="Synonym Qualifier"/>
  <ItemRef ItemOID="PE.PEBLFL"   OrderNumber="14"  Mandatory="Yes/No"  Role="Record Qualifier"/>
  <ItemRef ItemOID="VISITNUM"    OrderNumber="15"  Mandatory="Yes/No"  Role="Timing"/>
  <ItemRef ItemOID="PE.PEDY"     OrderNumber="17"  Mandatory="Yes/No"  Role="Timing"/>
  <!-- ********************************************************** -->
  <!-- def:leaf details for hypertext linking the dataset         -->
  <!-- ********************************************************** -->
  <def:leaf ID="Location.PE" xlink:href="pe.xpt">
    <def:title>crt/PharmaABC04001/pe.xpt</def:title>
  </def:leaf>
</ItemGroupDef>
```

Figure 2.2: Data Definition Table for the variables in PE data set in define.xml

```
<!-- ************************************************************ -->
<!-- The details of each variable is here for all domains         -->
<!-- ************************************************************ -->
<ItemDef OID="DOMAIN"   Name="DOMAIN"      DataType="char"    Length="2"  Origin="Derived"
         Comment="Two-character..."      def:Label="Domain Abbreviation">
</ItemDef>
<ItemDef OID="PE.PEBLFL"   Name="PEBLFL"   DataType="char"    Length="1"  Origin="CRF or Derived"
         Comment="Indicator used..."     def:Label="Baseline Flag">
  <CodeListRef CodeListOID="YESNO"/>
</ItemDef>
<ItemDef OID="PE.PEDY"     Name="PEDY"     DataType="num" Length="8"  Origin="Derived"
         Comment="Study day of..."       def:Label="Study Day of Examination"
         def:ComputationMethodOID="StudyDay">
</ItemDef>
<ItemDef OID="PE.PETESTCD" Name="PETESTCD" DataType="char"    Length="40" Origin="CRF"
         Comment="Topic variable for..." def:Label="Body System Examined">
  <def:ValueListRef ValueListOID="PRETESTCD"/>
</ItemDef>
<ItemDef OID="STUDYID"  Name="STUDYID"     DataType="char"    Length="6"  Origin="CRF"
         Comment="Unique identifier..."  def:Label="Study Identifier">
</ItemDef>
<ItemDef OID="USUBJID"  Name="USUBJID"     DataType="char"    Length="7"  Origin="Sponsor Defined"
         Comment="Unique subject..."     def:Label="Unique Subject Identifier">
</ItemDef>
<ItemDef OID="VISITNUM" Name="VISITNUM"    DataType="num" Length="8"  Origin="CRF or Derived"
         Comment="Clinical encounter."   def:Label="Visit Number">
</ItemDef>
```

Figure 3.1: Table of Contents for PE data set displayed in define.xml style sheet

Datasets for Study PharmaABC04001					
Dataset	**Description**	**Structure**	**Purpose**	**Keys**	**Location**
PE	Physical Examinations - PE	Findings - One record per subject per exam per visit	Tabulation	STUDYID, USUBJID, VISITNUM, PETEST	crt/PharmaABC04001/pe.xpt

Figure 3.2: Data Definition Table for the variables in PE data set displayed in define.xml style sheet

Physical Examinations - PE Dataset (PE)						
Variable	**Label**	**Type**	**Controlled Terms or Format**	**Origin**	**Role**	**Comment**
STUDYID	Study Identifier	text		CRF	Identifier	Unique identifier for a study within the submission.
DOMAIN	Domain Abbreviation	text		Derived	Identifier	Two-character abbreviation for the domain most relevant to the observation.
USUBJID	Unique Subject Identifier	text		Sponsor Defined	Identifier	Unique subject identifier within the submission.
PETESTCD	Body System Examined Short Name	text		CRF or Derived	Synonym Qualifier	Topic variable for PE. Short name for the value in PETEST, which can be used as a column name when converting the dataset from a vertical format to a horizontal format. The short names can be up to 8 characters. Examples: CV, GI. For subject-level exam, value should be 'ALL'.
PEBLFL	Baseline Flag	text	YESNO	CRF or Derived	Record Qualifier	Indicator used to identify a baseline value.
VISITNUM	Visit Number	integer		CRF or Derived	Timing	1. Clinical encounter number. 2. Numeric version of VISIT, used for sorting.
PEDY	Study Day of Examination	integer		Derived	Timing	1. Study day of physical exam, measured as integer days. 2. Algorithm for calculations must be relative to the sponsor-defined RFSTDTC variable in Demographics. This formula should be consistent across the submission. Computational Algorithm: StudyDay

Figure 3.3: Data Definition Table for code list displayed in define.xml style sheet

Controlled Terminology (Code Lists) Section	
Code Value	**Code Text**
YESNO, Reference Name (YESNO)	
N	No
Y	Yes

Figure 3.4: Data Definition Table for value list displayed in define.xml style sheet

Value Level Metadata							
Source Variable	**Value**	**Label**	**Type**	**Controlled Terms or Format**	**Origin**	**Role**	**Comment**
PETESTCD	Abdomen	Abdomen Exam	text		CRF		
PETESTCD	Skin	Skin Exam	text		CRF		

Figure 3.5: Data Definition Table for computation method displayed in define.xml style sheet

Computational Algorithms Section	
Reference Name	**Computation Method**
StudyDay	Examination Date - Screening Day + 1

RELATIONAL DATA MODEL

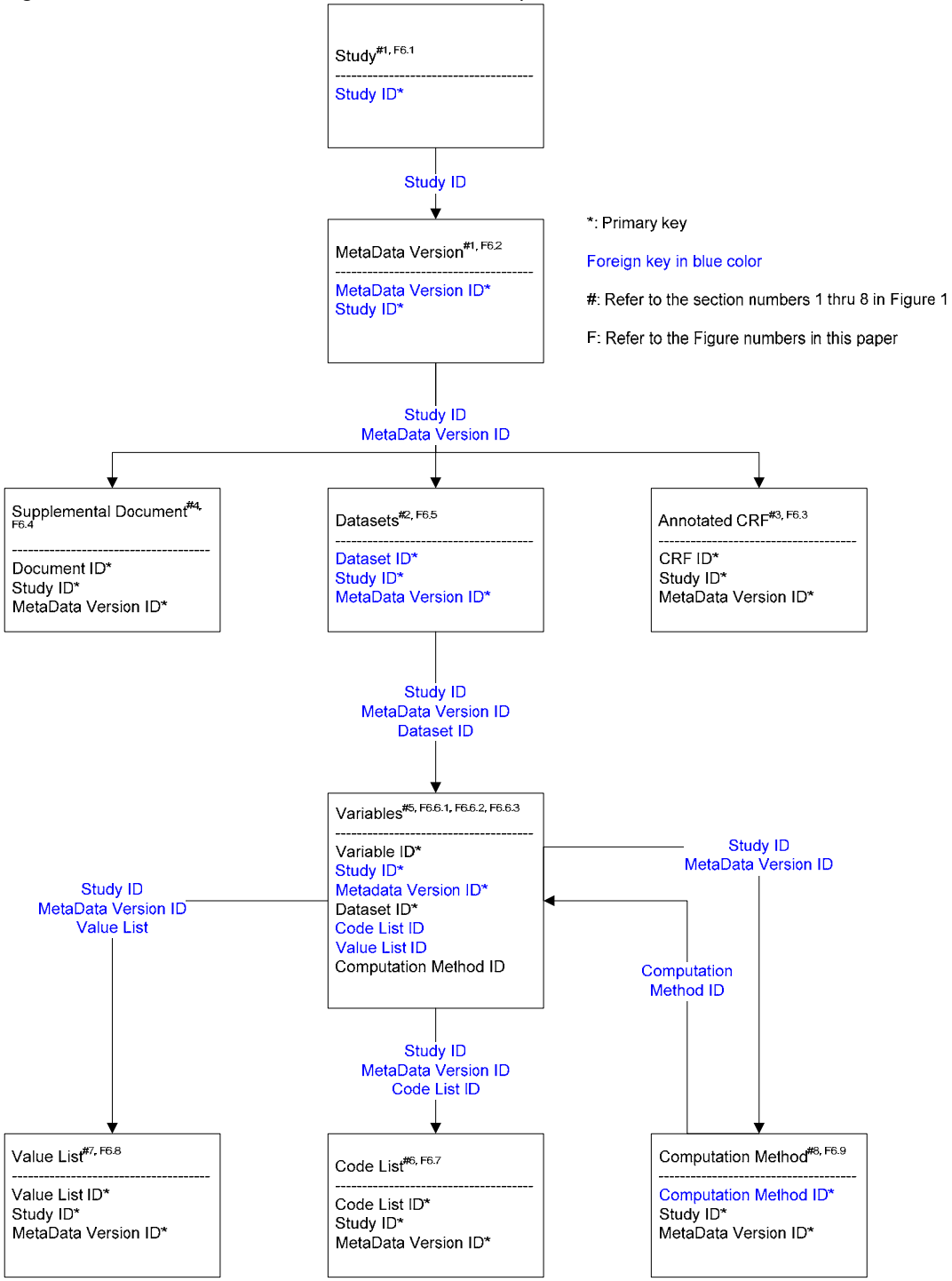
Traditionally SAS data sets are flat-structured and each may contain a set of repeated header variables. Figure 4 illustrates an example, where the descriptive columns, e.g., Study Description, Metadata Version Description, and Data set Description, repeat up to as many times as the total number of variables in the study. Such a data structure may be convenient for data analysis, but can be difficult to maintain as the number of variables in the study increases. For example, if a variable within a data set is accidentally added twice into this file, it would violate the requirement of unique element IDs for define.xml.

Alternatively a normalized data structure consists of a group of data sets that relate to one another by predefined integrity constraints. They are organized to reduce and even eliminate data redundancy so that there is no repeated header information in multiple data sets.

Figure 4: Flat data structure

Study ID	Study Description	MetaData Version ID	MetaData Version Description	Dataset ID	Dataset Description	Variable ID	Variable Description
ABC	ABC Desc	3.1.0	SDTM 3.1.0	AE	Adverse Events	SUBNUM	Subject Number
ABC	ABC Desc	3.1.0	SDTM 3.1.0	AE	Adverse Events	VISITNUM	Visit Number
ABC	ABC Desc	3.1.0	SDTM 3.1.0	CM	Medication	SUBNUM	Subject Number
ABC	ABC Desc	3.1.0	SDTM 3.1.0	CM	Medication	VISITNUM	Visit Number

Figure 5: Relational Data Model based on define.xml specification



PRIMARY KEY

In the case of define.xml, the first integrity constraint is the primary key; each ODM element must have an ID that is not null and is unique among all the elements in this define.xml. For example, the `OID` of a data set name in the Datasets section `ItemGroupDef` in Figure 2.1 is the primary key.

FOREIGN KEY

Next is the foreign key constraint; if an element in one section in define.xml plays as a key element in another section, that element is a foreign key. For example, the relationships between the element `ItemRef` in the Datasets section in Figure 2.1, and the element `ItemDef` in the Variables section in Figure 2.2 is maintained by the same variable IDs in the two sections, e.g., `ItemOID` and `OID`, respectively. The interesting thing about the variable ID is the fact that it is part of the primary key for the Variables section as well as the foreign key to the Datasets section.

Figure 5 illustrates such a relational data model that can accommodate multiple studies for multiple submissions. It consists of primary key and foreign key constraints. They are organized according to the define.xml schema. There are nine tables in the data model to store the eight sections of information shown in Figure 1. At the top level is the Study table with one record per study ID, which is the primary key, and contains the general information about the study. Each study can have one or more metadata versions. So for each study as a foreign key in the Study table, there are one or more metadata versions in the MetaDataVersion table. Given the combination of a study and a metadata version, which in turn serves as the foreign key, there are multiple data sets, one or more CRF files, and one or more supplemental document files. They correspond to the Datasets table, AnnotatedCRF table, and SupplementalDocument table, respectively, where the combination of the study ID, metadata version ID, and the data set ID, CRF ID, or document ID is the primary key within each of the three tables. Given a data set, there are a number of variables, and the information about the variables resides in the Variables table. A variable may have none or one SAS format, e.g., controlled terminology or code list, none or one value list, and none or one computation method. They are stored in the CodeList table, ValueList table, and ComputationMethod table, respectively.

SAS/AF APPLICATION

The integrity constraints in the relational data model require maintenance. This is where the SAS/AF application comes to the picture. It utilizes SAS/AF objects such as Form Viewer, Table Viewer and SAS Data Set Model, and constructs nine screens as in Figures 6.1 thru 6.9 that interact with the nine tables in Figure 5.

There are three mechanisms implemented in the application to enforce the integrity constraints from the relational data model. They utilize the Add buttons, macro variables, and reverse engineering as in Figure 7.

PRIMARY KEY CONSTRAINTS – ADD BUTTON

First of all, the application makes use of buttons, labeled Add, that allow us to add unique values on the primary keys in certain tables. They are the tables MetaDataVersion, AnnotatedCRF, SupplementalDocument, and ComputationMethod as in Figure 7. For example, in the MetaData Version screen in Figure 6.2 we click on the Add button to add a new metadata version ID for the study. There is a algorithm in the background to check for the uniqueness of the newly entered value. If it already exists in the table MetaDataVersion for the study, the application brings up that existing row. If it is a new one, the application accepts it and displays the new row.

FOREIGN KEY CONSTRAINTS – MACRO VARIABLE

Secondly, the values on the foreign keys are retained in macro variables as we go from one screen to the next. For example, in Figure 6.1, the selected study ID is set to a macro variable. The macro variable constructs a WHERE clause and filters the data in the table MetaDataVersion. So the MetaData Version screen displays only the rows for the selected study ID from the table MetaDataVersion. Similarly, in the MetaData Version screen, given a study and a metadata version, we click on the button CRF, Document, or Datasets, and view or edit the information about the annotated CRF (Figure 6.3), the supplemental document (Figure 6.4), or the data sets (Figure 6.5), respectively. Further from the Datasets table to the Variables table, we click on the button Variables in the Datasets screen, and the detailed description of the variables for the data set is displayed as in Figures 6.6.1, 6.6.2 and 6.6.3. These three screens are the Variables screen for three variables, one with the controlled terminology `YESNO`, another with the value list PETESTCD, and the other with the computation method `StudyDay`. From Figure 6.6.1, when we click on the button CodeList in the Variables screen, the Code List screen in Figure 6.7 is displayed with the information about the controlled terminology `YESNO` in the table CodeList. Similarly, from Figure 6.6.2 or 6.6.3, when we click on the button ValueList or Method in the Variables screen, the Value List screen (Figure 6.8) or the Computation Method screen (Figure 6.9), respectively, is displayed with the definition of the value list PETESTCD in the table ValueList, or the computation mthod `StudyDay` in the ComputationMethod table.

The values of the foreign keys must not change. So the SAS/AF objects that display the values are set to not

editable. In the front end is the objects in gray background color, for example, Study OID on the top of the screen in Figure 6.2, alerting us that it is not editable.

One step further to enforce the parent-child relationships is in the MetaData Version screen. We must save a new MetaData Version ID by clicking the Save button, prior to proceeding or reverse-engineering the information about the study database into the system. Secondly, the application does not allow us to delete a saved value on any primary key in any table, because doing so may violate the integrity constraints. For example, in the MetaData Version screen, the application does not allow the deletion of a saved metadata version ID. If one must be deleted, the application administrator takes care of it by first deleting the relevant rows in the lowest level child table, moving up one by one, and finally deleting the row in the MetaDataVersion table.

BOTH PRIMARY KEY AND FOREIGN KEY CONSTRAINTS – REVERSE ENGINEERING

The third mechanism is the reverse engineering of the study database by importing the information from SAS views, SAS format catalog, and sponsor-provided SDTM files to the tables Datasets, Variables, CodeList and ValueList. This is implemented in the MetaData Version screen in Figure 6.2. When we click on the button Import Dataset/Variable, an external SAS program runs. The SAS program calls in the macro variables for the selected study ID and metadata version ID. At the same time it reads in the information about the study database from SAS dictionary views Tables and Columns, SAS format library, and SDTM specification files. Then it outputs the information, including the values on Study ID and MetaData VersionID, to the mentioned tables above.

To ensure the uniqueness of the variable names, with the exception of a handful of unique identifier such as study ID and subject ID, the application makes use of two-level variable name ID, e.g., DomainName.VariableName.

Figure 6.1: Study screen

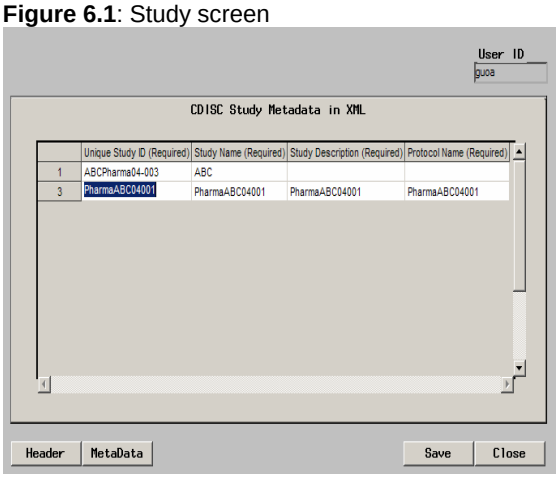


Figure 6.2: MetaData Version screen

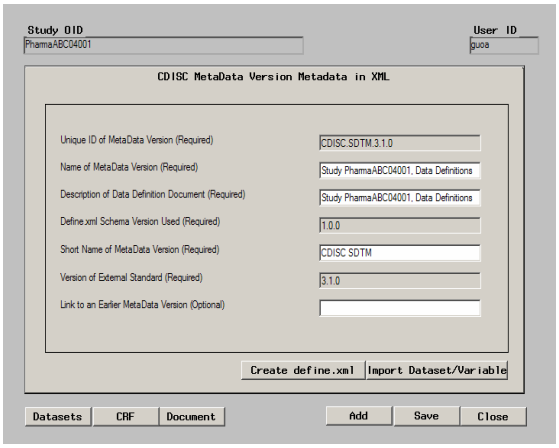


Figure 6.3: Annotated CRF screen

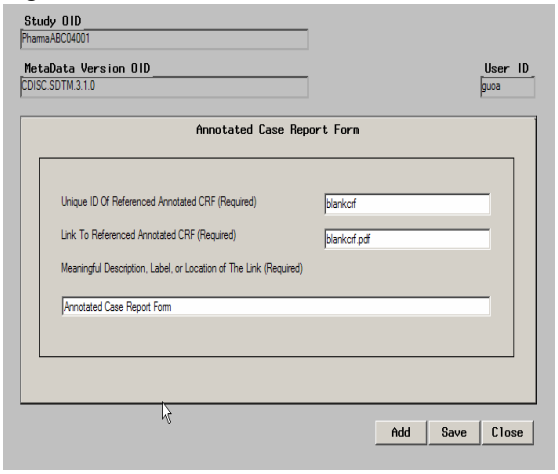


Figure 6.4: Supplemental Document screen

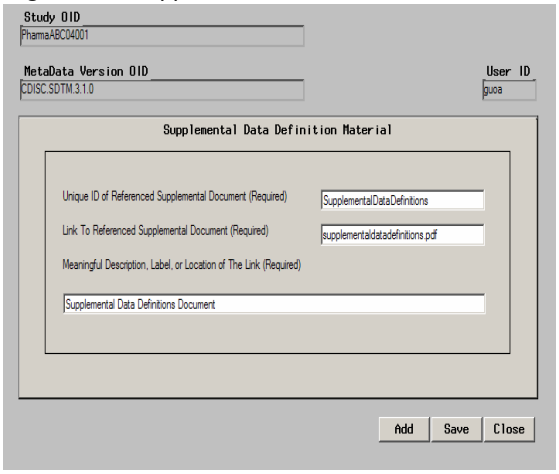


Figure 6.5: Datasets screen

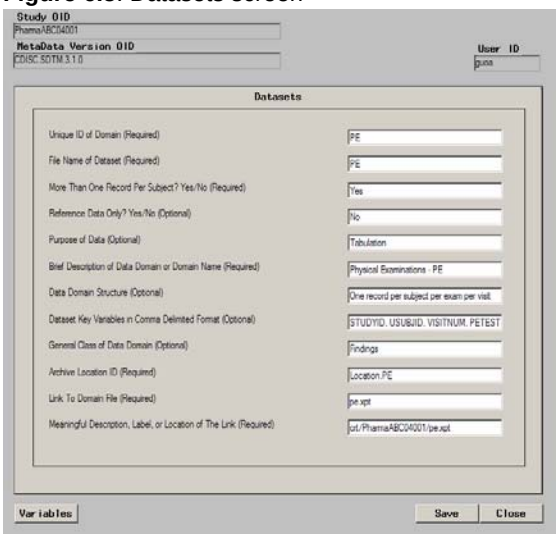


Figure 6.6.1: Variable screen with YESNO code list

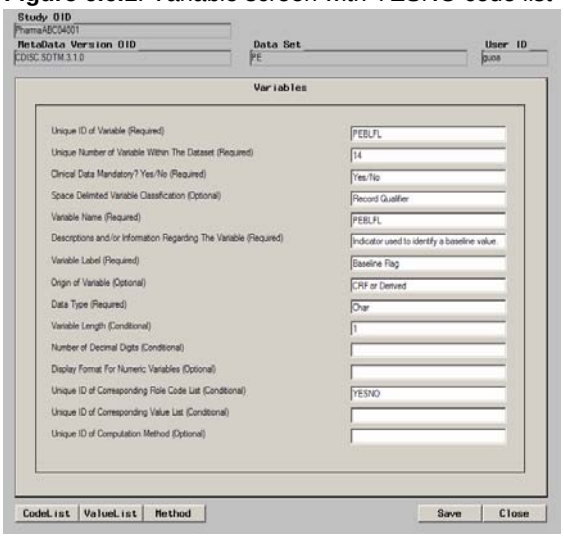


Figure 6.6.2: Variables screen with PETESTCD value list

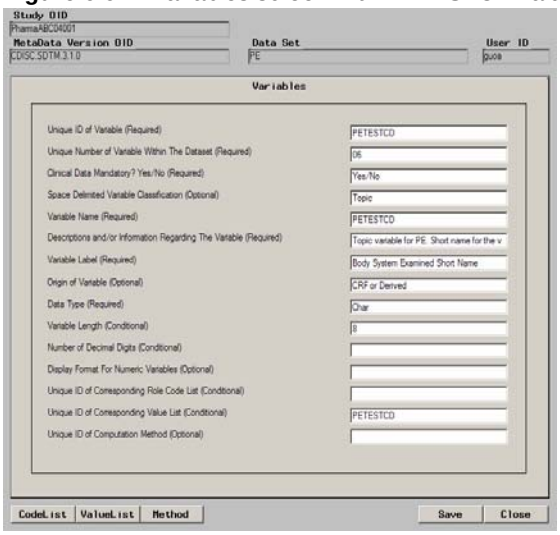


Figure 6.6.3: Variables screen with StudyDay method

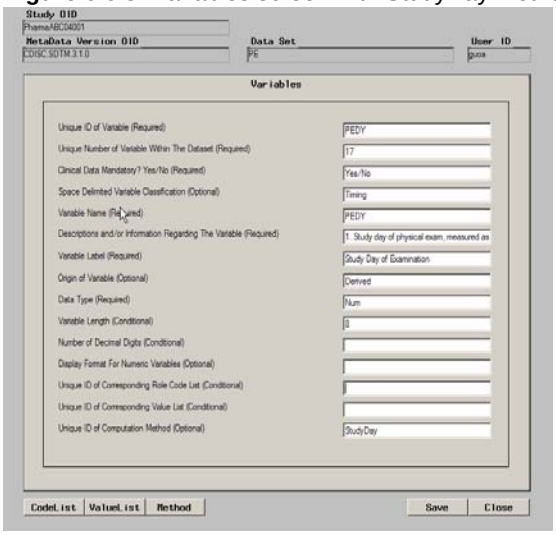


Figure 6.7: Code List screen

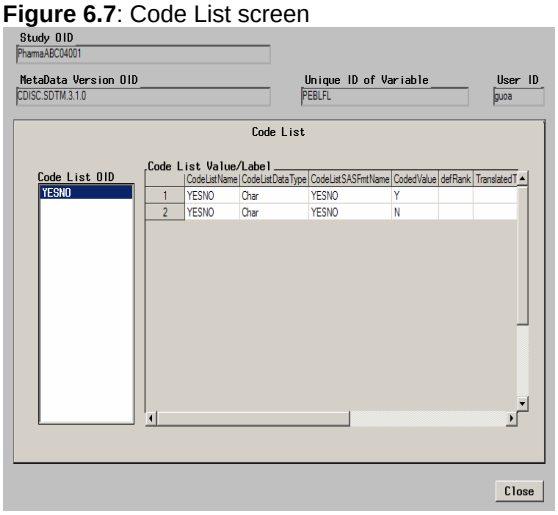


Figure 6.8: Value List screen

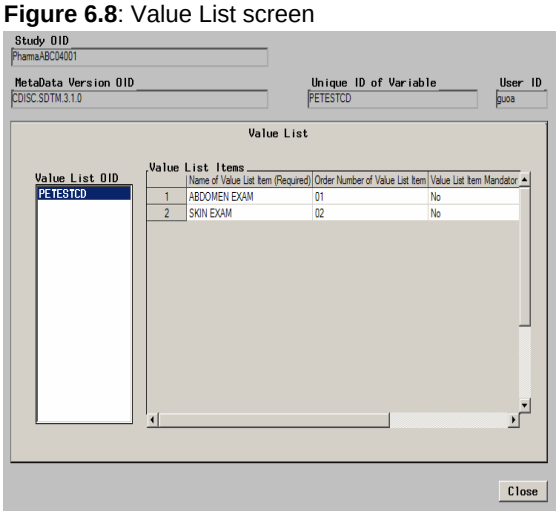


Figure 6.9: Computation Method screen

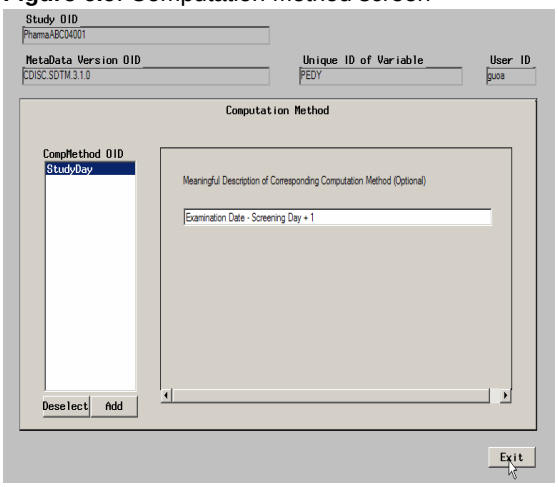
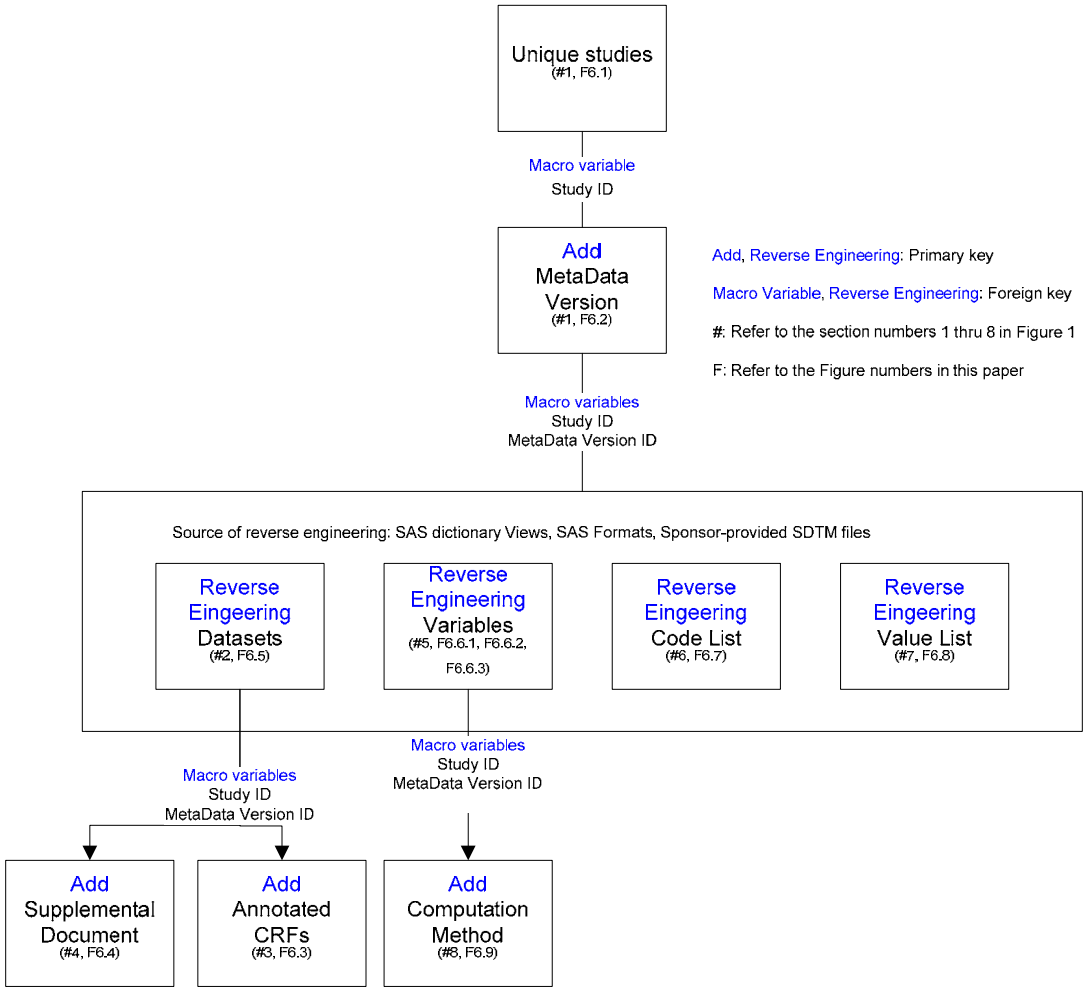


Figure 7: Enforcement of integrity constraints



DATA _NULL_ STEP

The foundation of the DATA _NULL_ step lies upon the consolidation of the data sets from the relational model according to the integrity constraints. There are different ways to organizing the data. A straightforward approach is to create eight data sets that correspond to the eight sections of information in the define.xml schema in Figure 1. Each of the eight data sets contains only the rows for the given study ID and metadata version as illustrated in Figure 7. The Study table and the MetadataVersion table are merged by the study ID which is the foreign key between the two tables. The Datasets table merges with the Variables table to include all the variables of each data set, because the Datasets section in define.xml (`ItemGroupDef` in Figure 2.1) needs to reference to the variables in each data set. The tables CodeList, ValueList and ComputationMethod include only those code lists, value lists, and computation methods, respectively, that are associated with one or more variables in the given study and metadata version.

SET STATEMENT

The next step is to stack up the eight data sets using SET statement, and produce a output data set named final. The output data set contains the new variable section. The value on the variable section corresponds to the eight section number in define.xml schema in Figure 1. For example, the section number for the Datasets table is 2. The purpose of the section numbers is to retain the order of the information in the final data set, which in turn generates the eight sections in define.xml in the desired order. It is worth noting that, except for the XML header part, the order of sections 2 thru 8 in define.xml is interchangeable; the define.xml style sheet recognizes the sections by the key words, for example, `ItemGroupDef`, and displays the contents properly.

PUT STATEMENTS

The construction of the define.xml document utilizes the PUT statement in DATA _NULL_ step. It reads in the data set final and generates the text file by following the define.xml schema. Figure 8 is a sample code that formats the Variables section in Figure 1, e.g., the detailed description of all the variables in the study. The output is as in Figure 2.2, and when viewed in define.xml style sheet as in Figure 3.2.

There are a few things to note when generating define.xml. First of all, some special characters that may exist in variable labels, data set labels, or comment fields can break the compilation of define.xml. They are &, ‘, “, < and >. Those special characters must be changed to `&`, `&apos;`, `&quot;`, `<`, and `>`, respectively. Secondly, white space is insignificant in define.xml document.

Figure 7: Processes from SAS data sets to define.xml

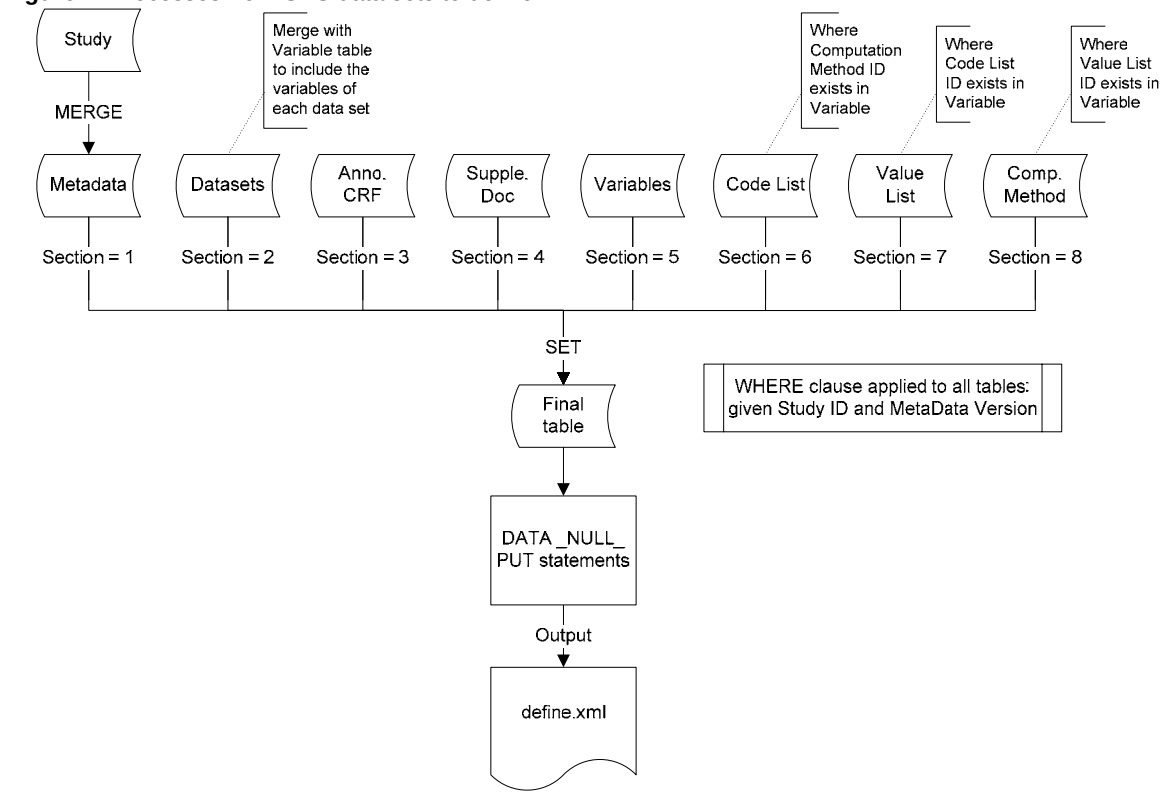


Figure 8: Data _Null_ step and Put statements to create the Variables section in define.xml

```
filename outfile "define.xml";
data _null_;
    set final;
    file outfile notitles lrecl=2000;
    by section Study MetaData datasets crf doc variables codelist valulist method;
    ...
    if (section=5) then do;
    ********************************************;
    ******** VARIABLES DEF              **********;
    ********************************************;
        if (first.section) then put
        '      <!-- ********************************************************** -->'/
        '      <!-- The details of each variable is here for all domains      -->'/
        '      <!-- ********************************************************** -->';
        put
        '      <ItemDef OID="' ItemOID +(-1) '"'/
        '        Name="' ItemDefName +(-1) '"'/
        '        DataType="' ItemDefDataType +(-1) '"'/
        '        Length="' ItemDefLength +(-1) '"'/
        '        Origin="' ItemDefOrigin +(-1) '"'/
        '        Comment="' ItemDefComment +(-1) '"';
        if (ItemDefDisplayFormat ne '') then put
        '        def:DisplayFormat="' ItemDefDisplayFormat +(-1) '"';
        if (CompMethodOID ne '') then put
        '        def:ComputationMethodOID="' compmethodoid +(-1) '"';
        put
        '        def:Label="' ItemDefLabel +(-1) '">';
        if (ValueListOID ne '') then put
        '        <def:ValueListRef ValueListOID="' valuelistoid +(-1) '"/>';
        if (CodeListOID ne '') then put
        '        <CodeListRef CodeListOID="' codelistoid +(-1) '"/>';
        put
        '      </ItemDef>';
    end;
    ...
run;
```

CONCLUSION

This paper presents a system that centralizes the information about the database of multiple studies. It can be used by SAS programmers as well as clinical staff to navigate to the detail of the database. With external SAS programs incorporated into the SAS/AF application, creating define.xml or any database documents required for regulatory submission can be accomplished at the click of a button.

DISCUSSION

The SAS/AF application can be modified to also accomplish the forward engineering of the tables where currently reverse engineering takes place. That is, a button labeled Add will be added to the screens Datasets, Variables, Code List and Value List, along with the mechanism that checks for the uniqueness of the values on the primary keys in the corresponding tables Datasets, Variables, CodeList and ValueList. Once it is implemented, we click on the Add button, and enter the design of the study database, such as the data set names, variable names, format names and values. After the design is complete and saved to the relational data model, we run an external SAS program to read in the information, and create SAS programs that will in turn be executed and generate the study database in SAS.

REFERENCES

CDISC define.xml Team 2005. “Case Report Tabulation Data Definition Specification (define.xml)” Standards Version 1.0.0.

ACKNOWLEDGMENTS
The author would like to extend a special thanks to Robert Stemplinger for his review of this paper, as well as his ongoing support for progress, creativity and innovation.

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

Annie Guo
ICON Clinical Research
Redwood City, CA, USA
E-mail: guoa@iconus.com
Web: www.iconclinical.com