PharmaSUG China 2015 - 30

Generating Define.xml Using SAS®
By Element-by-Element And Domain-by-Domian Mechanism

Lina Qin, Beijing, China

ABSTRACT

An element-by-element and domain-by-domain mechanism is introduced for generating define.xml using SAS®. Based on CDISC Define-XML Specification, each element in define.xml can be generated by applying a set of templates instead of writing a great deal of “put” statements. This will make programs more succinct and flexible. The define.xml file can be generated simply by combining, in proper order, all of the relevant elements. Moreover, as each element related to a certain domain can be separately created, it is possible to generate a single-domain define.xml by combining all relevant elements of that domain. This mechanism greatly facilitates generating and validating define.xml.

Keywords: Define-XML, define.xml, CRT-DDS, SDTM, CDISC, SAS

1 INTRODUCTION

Define.xml file is used to describe CDISC SDTM data sets for the purpose of submissions to FDA[1]. While the structure of define.xml is set by CDISC Define-XML Specification[1], its content is subject to specific clinical study. Many experienced SAS experts have explored different methods to generate define.xml using SAS[5][6][7][8][9].

This paper will introduce a new method to generate define.xml using SAS. The method has two features. First, each element in define.xml can be generated by applying a set of templates instead of writing a great deal of “put” statements. This will make programs more succinct and flexible. Second, the define.xml can be generated on a domain-by-domain basis, which means each domain can have its own separate define.xml file. This mechanism will help improve efficiency in the process of generating and validating define.xml.

As the primary idea of this method came from the structure of define.xml specified in Define-XML Specification, a brief introduction to the structure of define.xml is made in Section 2. Section 3 gives an overview of the method. Section 4 takes ItemGroupDef element as an example to illustrate how each element can be generated by applying templates. Section 5 shows how define.xml file is generated by combining all elements in certain orders. Section 6 shows how a single-domain define.xml is generated.

2 STRUCTURE OF DEFINE.XML

As specified in Define-XML Specification, the define.xml is composed of a series of element sections in certain order. (Note: an element section contains all elements of the same type.[2]) For the purpose of introducing this method, all element sections are categorized into nine parts which are shown in Figure 1.

Part one and part nine section usually each contains tens of XML code lines and is unrelated to SDTM data sets. So they can be easily generated by SAS programming.

Part six (CodeList), part seven (MethodDef) and part eight (def:CommentDef) each contains hundreds to thousands of lines, but they are also relatively easy to deal with for two reasons. First, they are simple in usage—only being referred to by other parts (elements) while containing no reference to others. Second, they are, to some extend, study-independent. For example, the CodeList element (controlled terminology) “CL.ACN” is identical across all studies. So a repository can be set up to prestore all such controlled terms, computational algorithms and comments. You may simply extract certain elements from the repository by referring to their unique OIDs.

Part two through part five are the most complicated elements to generate. Each of them contains hundreds to thousands of lines. They are study-specific, SDTM-related and referring to each other. Fortunately, Define-XML Specification has made detailed instructions on the structure and content of each element. For example, Define-XML Specification has specified the Name, XPath, Textual Value, Usage, Attributes and Child Elements of each element. These specifications enable establishing templates for all elements which can be used to generate elements line by line.

Figure 1 Element Sections Categorized in Nine Parts

```
①  <?xml version="1.0" encoding="UTF-8"?>
    <?xml-stylesheet type="text/xsl" href="../stylesheets/define2-0-0.xsl"?>
    <ODM>
      <Study>
        <GlobalVariables>
        <MetaDataVersion>
            <def:AnnotatedCRF>
            <def:SupplementalDoc>

②           <def:ValueListdef>
                <ItemRef>

③           <def:WhereClauseDef>
                <RangeCheck>

④           <ItemGroupDef>
                <Description>
                <ItemRef>
                <def:leaf>

⑤           <ItemDef>
                <Description>
                ...

⑥           <CodeList>
                <EnumeratedItem>
                ...
⑦           <MethodDef>
                <Description>
                ...
⑧           <def:CommentDef>
                <Description>
                ...
⑨           <def:leaf>
                <def:title>
        </MetaDataVersion>
      </Study>
    </ODM>
```

Note: Closing tabs are not shown here except for ODM, Study and MetaDataVersion.

3 METHOD OVERVIEW

The basic idea of this method is to generate define.xml by applying a series of templates. One template will be used for generating one type of element. All elements of the same type can be created using the same template. Each template bases on the specification of that element in Define-XML Specification.

Let's take ItemDef element as an example to illustrate how the template of ItemDef is set up. Three ItemDef elements (without child element and closing tab) in a real define.xml file[2] are listed below:

```
<ItemDef OID="IT.AE.AEACN" Name="AEACN" DataType="text" Length="30" SASFieldName="AEACN">
<ItemDef OID="IT.AE.AEENDY" Name="AEENDY" DataType="integer" Length="3" SASFieldName="AEENDY">
<ItemDef OID="IT.AE.AEENRF" Name="AEENRF" DataType="text" Length="5" SASFieldName="AEENRF">
```

We can see that these three ItemDef elements have the same pattern:

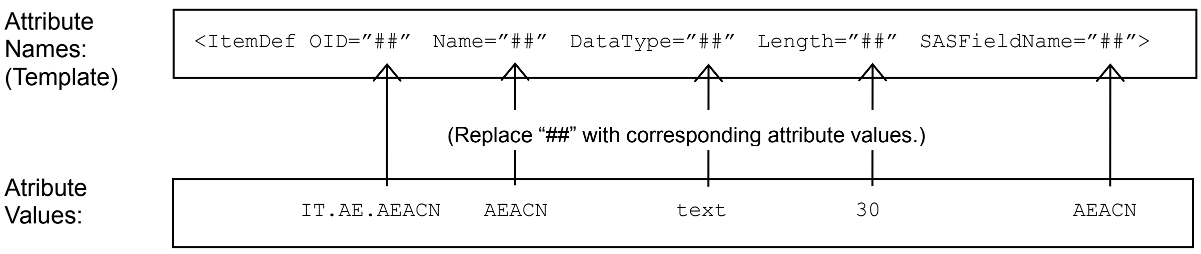
```
<ItemDef OID="##"  Name="##"  DataType="##"  Length="##"  SASFieldName="##">
```

As a matter of fact, based on its relevant specification, each element can be decomposed into the following two components which is illustrated in Figure 2:

- The name of element (e.g. "ItemDef") and the name of element attributes (e.g. "OID", "Name", etc.). This syntax is identical for all elements with the same type (e.g. all ItemDef elements) and thus can be used as a template.

- The value of each attribute (e.g. "IT.AE.AEACN" for "OID", "AEACN" for "Name", etc.). In contrast to the name of attribute, the value of attribute is different for each different element, even though they are of the same type. For example, for the same attribute, different ItemDef elements have different attribute values.

Figure 2 Decomposition of ItemDef Element


Attribute Names: (Template)
<ItemDef OID="##" Name="##" DataType="##" Length="##" SASFieldName="##">
(Replace "##" with corresponding attribute values.)
Atribute Values:
IT.AE.AEACN
AEACN
text
30
AEACN

Note that all attribute values are double-quoted. Such kind of format enables automatically searching a template for the position where an attribute value shall locate (e.g. "##") and then replacing it with certain source data.

Section 4 will take ItemGroupDef element as an example to illustrate how each element can be generated by applying templates.

4 GENERATING ITEMGROUPDEF BY APPLYING TEMPLATES

In order to generate define.xml file, each element in define.xml shall be generated first. Here I will take ItemGroupDef element (domain level metadata) and the relevant ItemRef element as an example to illustrate how to generate elements by applying templates and relevant source data sets.

The names and descriptions of SAS data sets used in this section are listed in Table 1.

Table 1 Names and Descriptions of SAS Data Sets

Name of Data Set	Feature	Description
ItemGroupDef_Tmp	Template	Template of ItemGroupDef element.
ItemGroupDef_S	Source	Contains values of element attributes for all domains.
ItemGroupDef_T	Source	Transposed version of ItemGroupDef_S.
ItemGroupDef_AE	Result	Completed ItemGroupDef element for AE domain.
ItemRef_Tmp	Template	Template of ItemRef element.
ItemRef_S	Source	Contains values of element attributes for all variables in all domains.
ItemRef_T	Source	Transposed version of ItemRef_S.
ItemRef_AE	Result	Completed ItemRef element(s) (within ItemGroupDef) for AE domain.

The steps of generating ItemGroupDef and relevant ItemRef element are listed below:

Step 1. Create ItemGroupDef template and ItemRef template.
Step 2. Create two source data sets, each of which contains attribute values for ItemGrouDef and ItemRef respectively.
Step 3. Transpose the two source data sets of Step 2.
Step 4. Perform basic validations on the transposed data sets of Step 3.
Step 5. Merge templates with corresponding transposed source data sets (select a single domain for each merging), then replace the pseudo attribute values in templates with the real attribute values in source data sets.
Step 6. Check and remove blank-valued attributes from the merged data sets and thus complete the ItemGroupDef and ItemRef elements respectively.
Step 7. Insert completed ItemRef element(s) into the completed ItemGroupDef at a predefined position.

Each step will be further explained in this section.

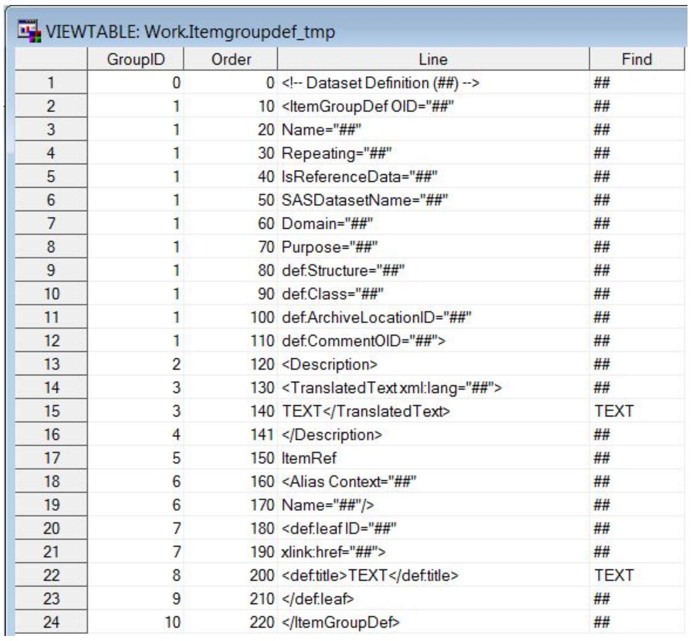
4.1 CREATING ITEMGROUPDEF AND ITEMREF TEMPLATES (STEP 1)

Based on the specifications of Define-XML Specification and following the formatting of define2-0-0-example-sdtm.xml [2] , the ItemGroupDef template is created as shown in Figure 3.

GroupID will be used in transposing process, Order will be used in merging process, and Find will be used to replace pseudo attribute values ("##") or textual values ("TEXT") with real attribute values or textual values in source data sets. Note the record with "150" in Order and "ItemRef" in Line. This is where the ItemRef elements will be inserted in.

Similarly, the ItemRef template is created as shown in Figure 4.

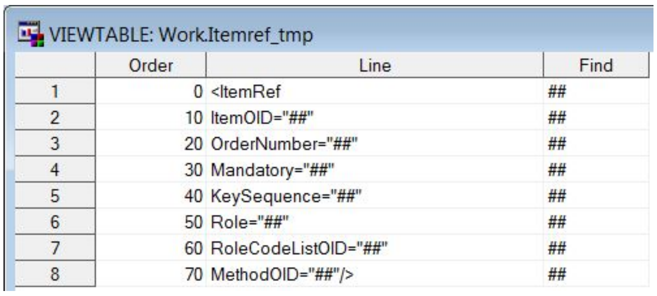
Figure 3 ItemGroupDef Template (ItemGroupDef_Tmp)



VIEWTABLE: Work.Itemgroupdef_tmp

	GroupID	Order	Line	Find
1	0	0	<!-- Dataset Definition (##) -->	##
2	1	10	<ItemGroupDef OID="##"	##
3	1	20	Name="##"	##
4	1	30	Repeating="##"	##
5	1	40	IsReferenceData="##"	##
6	1	50	SASDatasetName="##"	##
7	1	60	Domain="##"	##
8	1	70	Purpose="##"	##
9	1	80	def:Structure="##"	##
10	1	90	def:Class="##"	##
11	1	100	def:ArchiveLocationID="##"	##
12	1	110	def:CommentOID="##">	##
13	2	120	<Description>	##
14	3	130	<TranslatedText xml:lang="##">	##
15	3	140	TEXT</TranslatedText>	TEXT
16	4	141	</Description>	##
17	5	150	ItemRef	##
18	6	160	<Alias Context="##"	##
19	6	170	Name="##"/>	##
20	7	180	<def:leaf ID="##"	##
21	7	190	xlink:href="##">	##
22	8	200	<def:title>TEXT</def:title>	TEXT
23	9	210	</def:leaf>	##
24	10	220	</ItemGroupDef>	##

Figure 4 ItemRef Template (ItemRef_Tmp)



VIEWTABLE: Work.Itemref_tmp

	Order	Line	Find
1	0	<ItemRef	##
2	10	ItemOID="##"	##
3	20	OrderNumber="##"	##
4	30	Mandatory="##"	##
5	40	KeySequence="##"	##
6	50	Role="##"	##
7	60	RoleCodeListOID="##"	##
8	70	MethodOID="##"/>	##

4.2 CREATING SOURCE DATA SETS (STEP 2 AND STEP 3)

The attribute values of ItemGroupDef and ItemRef are prestored in corresponding source data sets (ItemGroupDef_S and ItemRef_S) shown in Figure 5 and Figure 6 respectively[1].

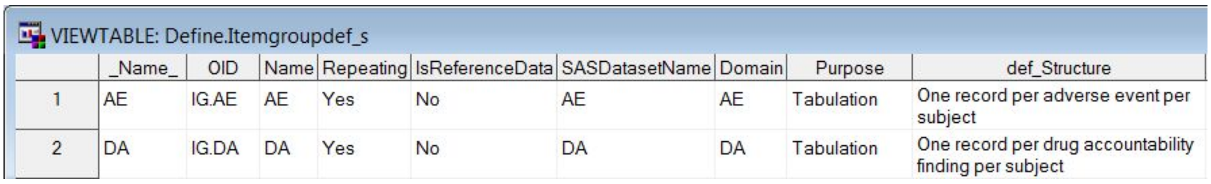
In both source data sets, each variable corresponds to one attribute of the element. All attributes, including required, conditional and optional ones, are included in both data sets. That is why some variable names are extraordinary long.

ItemGroupDef_S contains 21 variables. Except "_Name_", the other 20 variables correspond to the ItemGroupDef attributes or child element attributes. "_Name_" will be used as the name of transposed variables in ItemGroupDef_T. ItemRef_S contains 8 variables. Except "Domain", the other 7 variables correspond to the ItemRef attributes. "Domain" will be used to identify which group of records (within one domain) shall be selected in the transposing process.

In practice, the attribute values come from different sources. For example, in ItemGroupDef_S, most values come from SDTM-IG specifications[3]. As a matter of fact, values in this data set are study-independent and identical across all studies. In ItemRef_S, some values, like "ItemOID", come from SDTM data set; some come from SDTM-IG, like "Mandaroty"; and some are assigned, like "MethodOID". In fact, when "ItemOID" are set, the following variables can be automatically set.

[1] The sample data (attribute values) in both data sets are extracted from CDISC define2-0-0-example-sdtm.xml[2]. Only AE (Adverse Events) and DA (Drug Accountability) are selected here as examples. In practice, source data of all domains can be included in the two data sets.

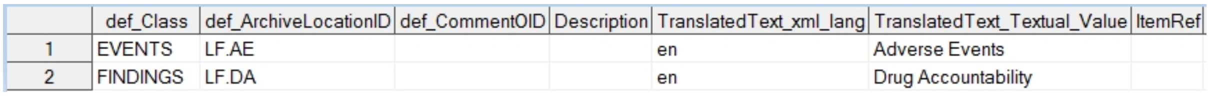
Figure 5 ItemGroupDef Source Data Set (ItemGroupDef_S, with all variables and partial records)



VIEWTABLE: Define.Itemgroupdef_s

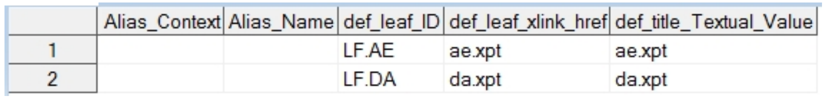
	Name	OID	Name	Repeating	IsReferenceData	SASDatasetName	Domain	Purpose	def_Structure
1	AE	IG.AE	AE	Yes	No	AE	AE	Tabulation	One record per adverse event per subject
2	DA	IG.DA	DA	Yes	No	DA	DA	Tabulation	One record per drug accountability finding per subject

(continued)



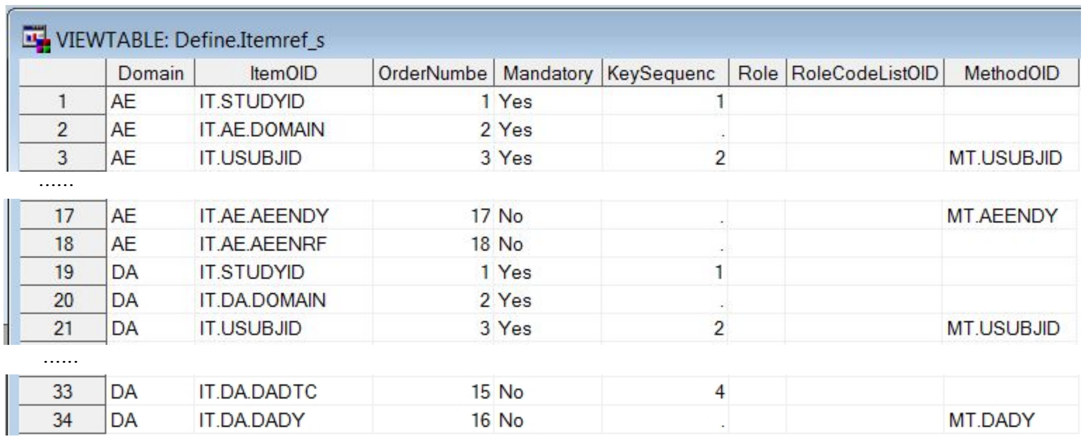
	def_Class	def_ArchiveLocationID	def_CommentOID	Description	TranslatedText_xml_lang	TranslatedText_Textual_Value	ItemRef
1	EVENTS	LF.AE			en	Adverse Events	
2	FINDINGS	LF.DA			en	Drug Accountability	

(continued)



	Alias_Context	Alias_Name	def_leaf_ID	def_leaf_xlink_href	def_title_Textual_Value
1			LF.AE	ae.xpt	ae.xpt
2			LF.DA	da.xpt	da.xpt

Figure 6 ItemRef Source Data Set (ItemRef_S, with all variables and partial records)



VIEWTABLE: Define.Itemref_s

	Domain	ItemOID	OrderNumbe	Mandatory	KeySequenc	Role	RoleCodeListOID	MethodOID
1	AE	IT.STUDYID	1	Yes	1			
2	AE	IT.AE.DOMAIN	2	Yes	.			
3	AE	IT.USUBJID	3	Yes	2			MT.USUBJID
......								
17	AE	IT.AE.AEENDY	17	No	.			MT.AEENDY
18	AE	IT.AE.AEENRF	18	No	.			
19	DA	IT.STUDYID	1	Yes	1			
20	DA	IT.DA.DOMAIN	2	Yes	.			
21	DA	IT.USUBJID	3	Yes	2			MT.USUBJID
......								
33	DA	IT.DA.DADTC	15	No	4			
34	DA	IT.DA.DADY	16	No	.			MT.DADY

Before proceeding to next steps, both ItemGroupDef_S and ItemRef_S shall be transposed into ItemGroupDef_T and ItemRef_T (Step 3). The relevant SAS program is presented in Appendix 1 Part I and Part IV.

4.3 VALIDATION CHECKS (STEP 4)

For the purpose of generating ItemGroupDef element, a basic validation on whether or not a certain required or conditional attribute having been assigned a value in ItemGroupDef_T will be performed. If no value assigned to a required attribute, an "Error" message will be put into the log like this :

```
    Error: Required attribute is missing for AE="OID".
```

If no value assigned to a conditional attribute, a "Warning" will be put like this:

```
    Warning: Conditional attribute is missing for DA="Domain".
```

The SAS program is presented in Appendix 1 Part II.

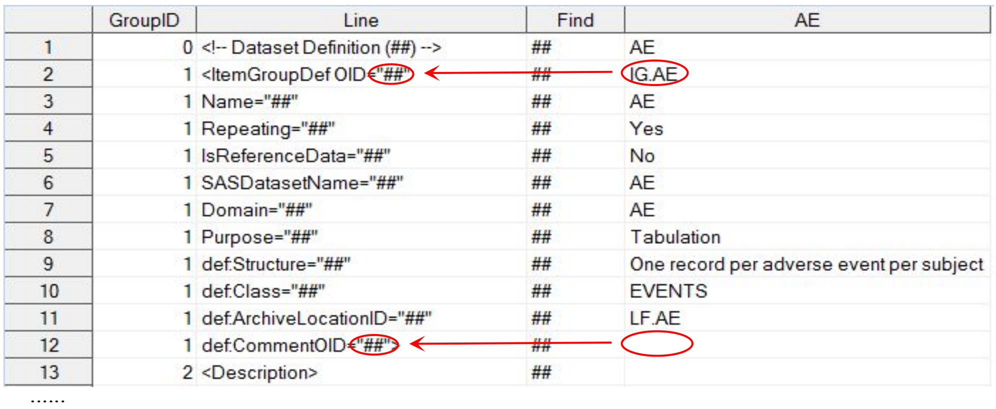
Similar validation checks can be performed on ItemRef element (SAS code not supplied).

4.4 MERGING, REPLACING AND REMOVING (STEP 5 AND STEP 6)

First, merge the template of ItemGroupDef (ItemGroupDef_Tmp) with the transposed source data set (ItemGroupDef_T) by Order variable. Note that only one domain shall be selected from ItemGroupDef_T for each merging, and the ItemGroupDef element will be created for that particular domain. Here AE is selected as an example and the merged data set is shown in Figure 7.

Next, replace the pseudo values in Line ("##" or "TEXT") with corresponding real values in AE. For example, replace "##" in "<ItemGroupDef OID="##"" (second record of Line) with "IG.AE" (second record of AE).

Figure 7 Merged ItemGroupDef Data Set for AE (Partial)



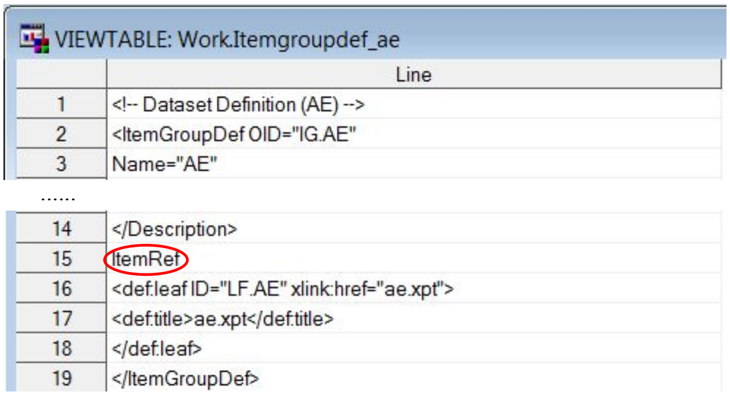
	GroupID	Line	Find	AE
1	0	<!-- Dataset Definition (##) -->	##	AE
2	1	<ItemGroupDef OID="##"	##	IG.AE
3	1	Name="##"	##	AE
4	1	Repeating="##"	##	Yes
5	1	IsReferenceData="##"	##	No
6	1	SASDatasetName="##"	##	AE
7	1	Domain="##"	##	AE
8	1	Purpose="##"	##	Tabulation
9	1	def:Structure="##"	##	One record per adverse event per subject
10	1	def:Class="##"	##	EVENTS
11	1	def:ArchiveLocationID="##"	##	LF.AE
12	1	def:CommentOID="##"	##	
13	2	<Description>	##	

......

Then, remove all blank-valued attributes in Line (Note that Line has been populated with real values after the above replacing process), e.g., “def:CommentOID=” ”” (twelfth record of Line). An attribute with blank value means it is either conditional or optional and has not been assigned a value. So, the attribute shall be removed from the final ItemGroupDef element.

After some additional minor manipulations, the ItemGroupDef_AE will be what is shown in Figure 8.

Figure 8 Nearly-Completed ItemGroupDef Element of AE (Partial)



VIEWTABLE: Work.Itemgroupdef_ae

	Line
1	<!-- Dataset Definition (AE) -->
2	<ItemGroupDef OID="IG.AE"
3	Name="AE"
......	
14	</Description>
15	ItemRef
16	<def:leaf ID="LF.AE" xlink:href="ae.xpt">
17	<def:title>ae.xpt</def:title>
18	</def:leaf>
19	</ItemGroupDef>

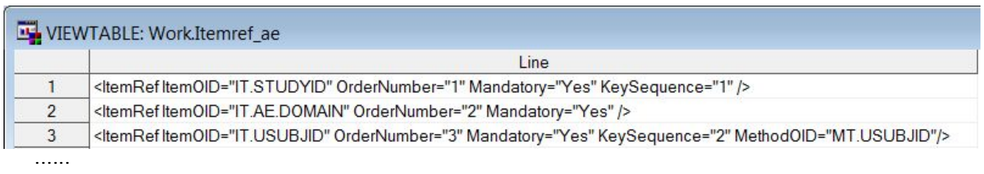
But this is not the completed version of ItemGroupDef_AE. Note the fifteenth record of “ItemRef” in Line. It shall be replaced with relevant ItemRef_AE elements for constructing final ItemGroupDef_AE.

The SAS program is presented in Appendix 1 Part III.

The process of generating ItemRef elements for AE is similar to that of ItemGroupDef element. After the processes of transposing, merging, replacing and removing, the ItemRef_AE is generated as shown in Figure 9.

The SAS program is presented in Appendix 1 Part IV and Part V.

Figure 9 Completed ItemRef Element of AE (Partial)



VIEWTABLE: Work.Itemref_ae

	Line
1	<ItemRef ItemOID="IT.STUDYID" OrderNumber="1" Mandatory="Yes" KeySequence="1" />
2	<ItemRef ItemOID="IT.AE.DOMAIN" OrderNumber="2" Mandatory="Yes" />
3	<ItemRef ItemOID="IT.USUBJID" OrderNumber="3" Mandatory="Yes" KeySequence="2" MethodOID="MT.USUBJID"/>

......

4.5 INSERTING (STEP 7)

After we have got nearly-completed ItemGrouDef_AE and ItemRef_AE, the last step to generate final ItemGroupDef for AE is to insert ItemRef_AE into ItemGroupDef_AE at the predefined position, i.e. “ItemRef” in Line. The SAS program is presented in Appendix 1 Part VI.

In this section, three small macros have been developed to generate ItemGroupDef element for AE domain. In practice, a more general macro can be easily created to further automate this process. Furthermore, by combing all ItemGroupDef elements of SDTM domains, the whole ItemGroupDef section in define.xml can be generated .

5 GENERATING DEFINE.XML BY COMBINING ALL ELEMENT SECTIONS

Similar to generating ItemGroupDef section, all other element sections of define.xml can be generated by applying relevant templates. Then the final define.xml file can be generated simply by combining all these element sections.

Suppose the data sets, each of which contains one part (element sections) listed in Figure 1, are respectively named as “Beginning” (part one), “Element1” through “Element7” (part two throuth part eight) and “Ending” (part nine). Use the following program to combine all of them and output to the define.xml file:

```
%let path=your-path;
filename xmlout “&path.\define.xml” lrecl=2000;
data _null_;
    set Beginning Element1-Element7 Ending;  /* Must in this order.*/
    file xmlout;
    put Line;
run;
filename xmlout clear;
```

Finally we get the define.xml file!

Please note that each data sets must be combined in order. As a matter of fact, “in correct order” is explicitly required or recommended by Define-XML Specification to ensure the normal performance of define.xml or for the purpose of regulatory submission[1].

There are three levels of order in define.xml that shall be followed:

- Order of element level —— within an element.
  For example, within the ItemGroupDef element of AE, the order of variables (i.e. ItemRef elements) shall follow the SDTM-IG variable ordering requirements[1][3].

- Order of section level —— within an element section.
  For example, within the ItemGroupDef section, the order of domains (i.e. ItemGroupDef elements) shall follow the SDTM-MSG ordering recommendations[1][4].

- Order of file level —— within define.xml
  The order of all element sections in define.xml file shall follow the global elements ordering of Define-XML Specification[1]. The correct order has been illustrated in Figure 1.

6 GENERATING SINGLE-DOMAIN DEFINE.XML

Technically speaking, based on the structure of define.xml, to generate an XML file for a single domain is fairly easy. I still take AE domain as an example to illustrate this. Suppose all element sections for AE domain have been generated, each in one data set, i.e. Beginning, ElementAE1 through ElementAE7 and Ending data sets, the following program shows how to generate define.xml for AE domain:

```
%let path=your-path;
filename xmlout “&path.\define_AE.xml” lrecl=2000;
data _null_;
    set Beginning ElementAE1-ElementAE7 Ending;  /* Must in this order.*/
    file xmlout;
    put Line;
run;
filename xmlout clear;
```

With the stylesheet of define2-0-0.xsl[2], the content of define_AE.xml can be displayed in web browser as shown in Appendix 2.

With similar methods, the define.xml file for each single domain, e.g. define_DA.xml, define_EX.xml, etc., can also be generated. The more interesting issue might be why we need to create such single-domain define.xml files. The answer is that it can greatly facilitate the process of creating and validating define.xml in the following ways: (1) Mistakes and problems can be found earlier, since the processes of preparing SDTM data sets, generating single-domain define.xml files and validating them can be executed “interleavingly” instead of sequentially; (2) Tasks can be fairly assigned to team members; and (3) Effectiveness of validation can be guaranteed.

The last but important thing to mention is about the structure of SDTM submission package. Check carefully that the define.xml file and its stylesheet, SDTM data sets (Xport files) and relevant PDF documents have been placed in correct paths, otherwise the define.xml won’t work as what you have expected.

7 CONCLUSION

As each element of define.xml can be separately generated by applying templates, the define.xml file can be generated on the element-by-element and domain-by-domain basis. This mechanism will make programs more succinct and flexible. The define.xml file can be generated simply by combining all relevant elements in proper order, so do the single-domain define.xml files. This method can significantly improve the efficiency in generating and validating define.xml. The indentation of codes in define.xml is not covered in the program. Although this will incur difficulty in directly reading XML codes, the display of define.xml will not be influenced.

REFERENCE

[1] CDISC Define-XML Team. “CDISC Define-XML Specification Version 2.0”. March 5, 2013. <http://www.cdisc.org/define-xml>.
[2] CDISC XML Technologies Team. “Define2-0-0-example-sdtm.xml”. March 3, 2013. <http://www.cdisc.org/define-xml>.
[3] CDISC Submission Data Standards Team. 2013. “Study Data Tabulation Model Implementation Guide: Human Clinical Trials Version 3.2”. <http://www.cdisc.org/sdtm>.
[4] CDISC SDS Metadata Team. “Study Data Tabulation Model Metadata Submission Guidelines (SDTM-MSG)”. December 31, 2011. <http://www.cdisc.org/sdtm>.
[5] Adams, John H. 2011. “Creating a define.xml file for ADaM and SDTM”. PharmaSUG 2011 - Paper AD14. <http://www.pharmasug.org/proceedings/2011/AD/PharmaSUG-2011-AD14.pdf>.
[6] Annapareddy, Uma Sarath & Kottam, Sandeep & Tripuraneni, Sree Lakshmi K. “GENERATING Define.xml”. NESUG 2011. <http://www.nesug.org/Proceedings/nesug11/ph/ph08.pdf>.
[7] Chen, Qinghua (Kathy) & Lenihan, James. (2014). “Creating Define.xml v2 Using SAS® for FDA Submissions”. PharmaSUG 2014 – Paper AD02. <http://www.pharmasug.org/proceedings/2014/AD/PharmaSUG-2014-AD02.pdf>.
[8] Jansen, Lex. 2014. “Creating Define-XML version 2 with the SAS® Clinical Standards Toolkit 1.6”. PharmaSUG 2014 - Paper DS23-SAS. <http://www.lexjansen.com/pharmasug/2014/DS/PharmaSUG-2014-DS23-SAS.pdf>.
[9] Molter, Michael. 2009. “A SAS® Programmer’s Guide to Generating Define.xml”. SAS Global Forum 2009, Paper 163-2009. <http://support.sas.com/resources/papers/proceedings09/163-2009.pdf>.

CONTACT INFORMATION

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

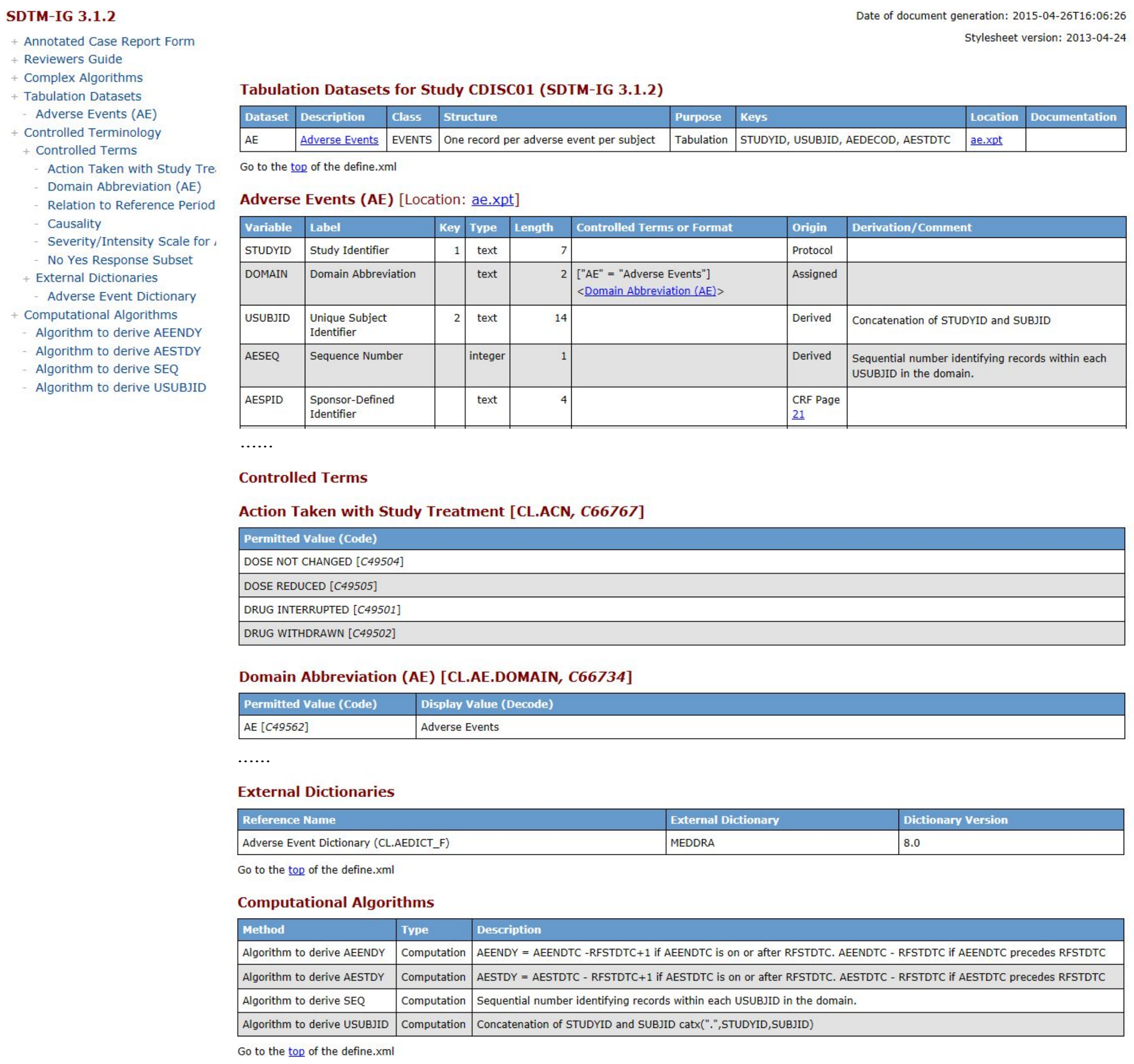
Lina Qin
Independent Consultant
Beijing, China
linach6@hotmail.com

APPENDIX 1: PROGRAM FOR GENERATING ITEMGROUPDEF AND ITEMREF

```
/* Part I ---------------------------------------------------------------------- *
 * Transpose source data set ItemGroupDef_S into ItemGroupDef_T.                 *
 * The variable "_Name_" in ItemGroupDef_S contains domain names that will be    *
 * used as name of transposed variables in ItemGroupDef_T.                       *
 * ----------------------------------------------------------------------------- */
proc transpose data=ItemGroupDef_S
               out=ItemGroupDef_T (where=(_Name_ ne "_Name_"));
    var _all_;
run;
/* Add variable "Order" to merge with template data set ItemGroupDef_Tmp later. */
/* Multiplying Order by 10 so that it has the same value with corresponding variable in
ItemGroupDef_Tmp. */
data ItemGroupDef_T;
    length Order 3.;
    set ItemGroupDef_T;
    label _Name_=;
    rename _Name_=Attribute;
    Order=_N_*10;
run;

/* Part II --------------------------------------------------------------------- *
 * Perform basic validations on ItemGroupDef_T.  That is, if a required attribute  *
 * kept in &Req_IG is missing, an error message will be put into log; if a conditional*
 * attribute kept in &Cond_IG is missing, a warning message will be put into log.   *
 * -----------------------------------------------------------------------------*/
%macro Validation_IG(ItemGroupDef_T);
    %local Req_IG Cond_IG;
    %let Req_IG = ("OID" "Name" "Repeating" "Purpose" "def_Structure"
                  "Translatedtext_Textual_Value" "def_leaf_ID" "def_leaf_xlink_href"
                  "def_title_Textual_Value");
    %let Cond_IG= ("Domain" "SASDatasetName" "def_Class" "def_ArchiveLocationID");
/* Retrieve the number of character variables in ItemGroupDef_T.  "Nvar" equals the number of
variables, each for one domain contained in ItemGroupDef_T, plus one ("Attribute").*/
data _null_;
    set &ItemGroupDef_T.;
    array Var{*} _character_;
    call symput("Nvar",dim(Var));
run;
/* Perform validation check sequentially on each domain.*/
data _null_;
    set &ItemGroupDef_T.;
    array Var{*} _character_;
    %do i=2 %to &Nvar.;
        if (Attribute in &Req_IG.) and missing(Var{&i.}) then
            put 'Err' 'or: Required attribute is missing for ' Var{&i.}= +(-2)
                         '"' Attribute +(-1) '".';
        else if (Attribute in &Cond_IG.) and missing(Var{&i.}) then
            put 'Warn' 'ing: Conditional attribute is missing for ' Var{&i.}= +(-2)
                          '"' Attribute +(-1) '".';
    %end;
run;
%mend Validation_IG;
%Validation_IG(ItemGroupDef_T);

/*Part III --------------------------------------------------------------------- *
 * Gernerate ItemGroupDef element for a certain selected domain by merging        *
 * ItemGroupDef_T(DsIn) with ItemGroupDef_Tmp(DsTmp).                             *
 * Domain - A certain domain selected for generating ItemGroupDef and contained in ItemGroupDef_T.*
 * DsIn - ItemGroupDef_T that contains real attribute values for each domain.           *
 * DsTmp - Template data set ItemGroupDef_Tmp.                                          *
 * DsOut - Output data set for the selected domain which contains ItemGroupDef but not ItemRefs.*
 * ----------------------------------------------------------------------------- */
%macro ItemGroupDef(Domain=, DsIn=, DsTmp=, DsOut= );
/* Merge DsTmp with the selected domain in DsIn that contains real attribute values of that domain.*/
data &DsOut. (drop=Order);
    merge &DsTmp. &DsIn.(keep=Order &Domain.);
    by Order;
run;
/* Replace pseudo attribute values in "Find" with real values in "&Domain". */
data &DsOut. (keep=GroupID Line);
    set &DsOut.;
    if _N_=1 then &Domain.="&Domain.";
    Line=tranwrd(Line,strip(Find),strip(&Domain.));
run;
/* Perform some minor manipulations in order to keep in line with standard Define-XML format. */
proc transpose data=&DsOut. out=&DsOut.(drop=_name_) prefix=Line;
    by GroupID;
    var Line;
run;
data &DsOut.;
    set &DsOut.;
    select (GroupID);
        when (3) do;
            Line1=cats(Line1,Line2);
            Line2="";
            end;
        when (6,7) do;
            Line1=catx(" ",Line1,Line2);
            Line2="";
            end;
        otherwise;
    end;
run;
proc transpose data=&DsOut. out=&DsOut.(where=(not missing(Col1)) keep=GroupID Col1);
    by GroupId;
    var Line:;
run;
/* Remove blank-valued attribute(s). */
data &DsOut.;
    set &DsOut.(rename=(Col1=Line));
    by GroupID;
    select (GroupID);
        when (1) do;
            if not Last.GroupID and findw(Line,"""")>0 then delete;
            else if Last.GroupID and findw(Line,"""","" >")>0 then delete;
            end;
        otherwise do;
            if findw(Line,"""")>0 then delete;
            end;
    end;
run;
/* Add missing ">" for GroupID=1 whenever necessary. */
data &DsOut. (keep=Line);
    set &DsOut.;
    by GroupID;
    if GroupID=1 and Last.GroupID then do;
        if find(Line,">","t",-1)=0 then Line=cats(Line,">");
    end;
run;
%mend ItemGroupDef;
%ItemGroupDef(Domain=AE, DsIn=ItemGroupDef_T,
              DsTmp=ItemGroupDef_Tmp, DsOut=ItemGroupDef_AE);

/* Part IV ------------------------------------------------------------------- *
 * Transpose source data set ItemRef_S into ItemRef_T.                         *
 * -------------------------------------------------------------------------- */
proc transpose data=ItemRef_S out=ItemRef_T prefix=Var;
    by Domain;
    var ItemOID OrderNumber Mandatory KeySequence Role RoleCodeListOID MethodOID;
run;
/* Retrieve the number of 'Var's in ItemRef_T, which equals to the number of variables of a certain
domain that has the most number of variables among all domains. */
data _null_;
    set ItemRef_T;
    array Var{*} Var:;
    call symput("Nvar",dim(Var));
run;
/* Add variable "Order" in order to merge with template data set ItemRef_Tmp later. */
/* For each domain in ItemRef_T, "Order" cycles from 10 to 70 by 10.*/
data ItemRef_T (drop=i);
    length Domain $8. Order 3.;
    do i=10 to 70 by 10;
        set ItemRef_T;
        Order=i;
        output;
    end;
    label _Name_=;
    rename _Name_=Attribute;
run;

/* Part V ---------------------------------------------------------------------- *
 * Generate ItemRef elements, one for each variable of the selected domain.       *
 * Domain - The Domain contained in ItemRef_T and selected for generating ItemRef elements.  *
 * DsIn - ItemRef_T that contains real attribute values of all variables in all domains.     *
 * DsTmp - Template data set ItemRef_Tmp.                                                    *
 * DsOut - Output data set which contains ItemRef elements of the selected domain.           *
 * ---------------------------------------------------------------------------- */
%macro ItemRefs(Domain=, DsIn=, DsTmp=, DsOut= );
/* Merge DsTmp with all variables of a selected domain of DsIn. */
data &DsOut.;
    merge &DsTmp. &DsIn.(where=(Domain="&Domain."));
    by Order;
    keep Line Find Var:;
run;
/* Replace pseudo attribute values in "Find" with real values in each 'Var' variable respectively.*/
data &DsOut.(drop=Line keep=Line:);
    set &DsOut.;
    %do i=1 %to &Nvar.;
        Line&i.=tranwrd(Line,strip(Find),strip(Var&i.));
    %end;
run;
/* Transpose &DsOut into the structure of 'one record per variable of the domain'. */
proc transpose data=&DsOut. out=&DsOut.(keep=Attrib:) prefix=Attrib;
    var _All_;
run;
/* Check out blank-valued attribute(s), then delete it(them).*/
data &DsOut.;
    set &DsOut.;
    %do i=2 %to 7;
        if findw(Attrib&i.,"""")>0 then Attrib&i.="";
    %end;
    if findw(Attrib8,"""")>0 then Attrib8="/>";
run;
data &DsOut. (keep=Line);
    length Line $2048.;
    set &DsOut.;
    Line=catx("", of Attrib:);
    if Line="<ItemRef />" then delete;
run;
%mend ItemRefs;
%ItemRefs(Domain=AE, DsIn=ItemRef_T, DsTmp=ItemRef_Tmp, DsOut=ItemRef_AE);

/* Part VI --------------------------------------------------------------------- *
 * Insert ItemRefs into ItemGroupDef to generate completed ItemGroupDef element.   *
 * ---------------------------------------------------------------------------- */
%macro Insert(Main=, Insert= );
/* Retrieve number of records (i.e. number of ItemRefs) in data set &Insert. */
%local dset Nobs;
%let dset=&Insert.;
%let dsid=%sysfunc(open(&dset.));
%if &dsid %then %do;
    %let Nobs=%sysfunc(attrn(&dsid,Nobs));
    %let rc=%sysfunc(close(&dsid.));
    %end;
```

```
%else
    %put Open for data set &dset failed - %sysfunc(sysmsg());
/* Inserting ItemRefs into ItemGroupDef. */
data &Main. (drop=i);
    length Line $2048;
    if Line="ItemRef" then do i=1 to &Nobs.;
        set &Insert.;
        output;
        end;
    set &Main.;
    if Line="ItemRef" then delete;
    output;
run;
%mend Insert;
%Insert(Main=ItemGroupDef_AE, Insert=ItemRef_AE);

/* End of program.-----------------------------------------------------------------*/
```

APPENDIX 2: DEFINE_AE.XML (PARTIAL)



SDTM-IG 3.1.2

+ Annotated Case Report Form
+ Reviewers Guide
+ Complex Algorithms
+ Tabulation Datasets
  - Adverse Events (AE)
+ Controlled Terminology
  + Controlled Terms
    - Action Taken with Study Tre
    - Domain Abbreviation (AE)
    - Relation to Reference Period
    - Causality
    - Severity/Intensity Scale for
    - No Yes Response Subset
  + External Dictionaries
    - Adverse Event Dictionary
+ Computational Algorithms
  - Algorithm to derive AEENDY
  - Algorithm to derive AESTDY
  - Algorithm to derive SEQ
  - Algorithm to derive USUBJID

Date of document generation: 2015-04-26T16:06:26

Stylesheet version: 2013-04-24

Tabulation Datasets for Study CDISC01 (SDTM-IG 3.1.2)

Dataset	Description	Class	Structure	Purpose	Keys	Location	Documentation
AE	Adverse Events	EVENTS	One record per adverse event per subject	Tabulation	STUDYID, USUBJID, AEDECOD, AESTDTC	ae.xpt	

Go to the top of the define.xml

Adverse Events (AE) [Location: ae.xpt]

Variable	Label	Key	Type	Length	Controlled Terms or Format	Origin	Derivation/Comment
STUDYID	Study Identifier	1	text	7		Protocol	
DOMAIN	Domain Abbreviation		text	2	["AE" = "Adverse Events"] <Domain Abbreviation (AE)>	Assigned	
USUBJID	Unique Subject Identifier	2	text	14		Derived	Concatenation of STUDYID and SUBJID
AESEQ	Sequence Number		integer	1		Derived	Sequential number identifying records within each USUBJID in the domain.
AESPID	Sponsor-Defined Identifier		text	4		CRF Page 21	

……

Controlled Terms

Action Taken with Study Treatment [CL.ACN, *C66767*]

Permitted Value (Code)
DOSE NOT CHANGED [*C49504*]
DOSE REDUCED [*C49505*]
DRUG INTERRUPTED [*C49501*]
DRUG WITHDRAWN [*C49502*]

Domain Abbreviation (AE) [CL.AE.DOMAIN, *C66734*]

Permitted Value (Code)	Display Value (Decode)
AE [*C49562*]	Adverse Events

……

External Dictionaries

Reference Name	External Dictionary	Dictionary Version
Adverse Event Dictionary (CL.AEDICT_F)	MEDDRA	8.0

Go to the top of the define.xml

Computational Algorithms

Method	Type	Description
Algorithm to derive AEENDY	Computation	AEENDY = AEENDTC -RFSTDTC+1 if AEENDTC is on or after RFSTDTC. AEENDTC - RFSTDTC if AEENDTC precedes RFSTDTC
Algorithm to derive AESTDY	Computation	AESTDY = AESTDTC - RFSTDTC+1 if AESTDTC is on or after RFSTDTC. AESTDTC - RFSTDTC if AESTDTC precedes RFSTDTC
Algorithm to derive SEQ	Computation	Sequential number identifying records within each USUBJID in the domain.
Algorithm to derive USUBJID	Computation	Concatenation of STUDYID and SUBJID catx(".",STUDYID,SUBJID)

Go to the top of the define.xml