PhUSE 2016

Paper CD14

Out-of-the-box %defineXML – Just a Simple SAS Macro

Katja Glaß, Bayer Pharma AG, Berlin, Germany

ABSTRACT

Currently, there are multiple define.XML tools and processes available. But somehow all processes seem to require manual intervention. This paper will present a define.XML macro which will be available in the PhUSE Script Repository and which can create the final define.XML V2 for ADaM, SDTM or any other data format according to the define specifications, without the need for manual updates. This paper will describe various ways in which this macro can be used to implement an automated process for generating the define file. First of all, the define data needs to be collected. This can be done by reading metadata, study data, documents, annotated CRFs or anything else. After this data is collected, the %defineXML macro can be used to create the data display in the pre-specified define format.

INTRODUCTION

Quite often the define.xml must be updated manually. It is unclear which information is to be stored where and how in the final define structure. Editing the XML file itself is done far too often. This is quite error prone and complicated. It should not be an issue to store the define information which is typically available in the XML format. The %DefineXML macro was developed to overcome this issue so people can concentrate on the information and the %DefineXML macro takes care of how to display it.

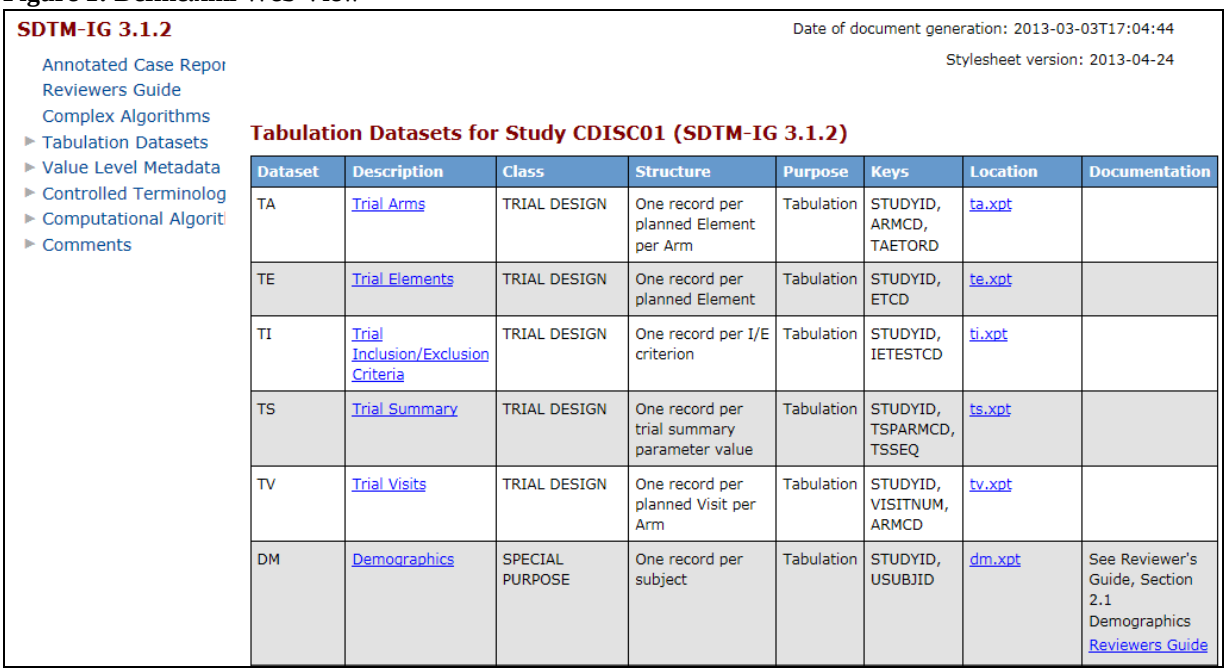
DEFINEXML – WHAT IS IT?

So what is the define.xml? According CDISC it is the following:

> *"Define-XML transmits metadata for SDTM, SEND and ADaM datasets; it is the metadata file sent with every study in each submission, which tells the FDA what datasets, variables, controlled terms, and other specified metadata were used."*[1]

Put simply, it is a format to submit metadata. Having a detailed look into the deliverable, there are two displays of the define.xml. It can be displayed in a browser with a nice style sheet (also provided by CDISC), but ultimately the define.xml is just plain text with descriptive tags.

Figure 1: Define.xml Web View



SDTM-IG 3.1.2

Date of document generation: 2013-03-03T17:04:44

Stylesheet version: 2013-04-24

- Annotated Case Repor
- Reviewers Guide
- Complex Algorithms
- Tabulation Datasets
- Value Level Metadata
- Controlled Terminolog
- Computational Algorit
- Comments

Tabulation Datasets for Study CDISC01 (SDTM-IG 3.1.2)

Dataset	Description	Class	Structure	Purpose	Keys	Location	Documentation
TA	Trial Arms	TRIAL DESIGN	One record per planned Element per Arm	Tabulation	STUDYID, ARMCD, TAETORD	ta.xpt	
TE	Trial Elements	TRIAL DESIGN	One record per planned Element	Tabulation	STUDYID, ETCD	te.xpt	
TI	Trial Inclusion/Exclusion Criteria	TRIAL DESIGN	One record per I/E criterion	Tabulation	STUDYID, IETESTCD	ti.xpt	
TS	Trial Summary	TRIAL DESIGN	One record per trial summary parameter value	Tabulation	STUDYID, TSPARMCD, TSSEQ	ts.xpt	
TV	Trial Visits	TRIAL DESIGN	One record per planned Visit per Arm	Tabulation	STUDYID, VISITNUM, ARMCD	tv.xpt	
DM	Demographics	SPECIAL PURPOSE	One record per subject	Tabulation	STUDYID, USUBJID	dm.xpt	See Reviewer's Guide, Section 2.1 Demographics Reviewers Guide

Figure 2: Define.xml Text View

```
<Study OID="cdisc01">
  <GlobalVariables>
    <StudyName>CDISC01</StudyName>
    <StudyDescription>CDISC Test Study</StudyDescription>
    <ProtocolName>CDISC01</ProtocolName>
  </GlobalVariables>
  <MetaDataVersion OID="MDV.CDISC01.SDTMIG.3.1.2.SDTM.1.2"
    Name="Study CDISC01, Data Definitions"
    Description="Study CDISC01, Data Definitions"
    def:DefineVersion="2.0.0"
    def:StandardName="SDTM-IG"
    def:StandardVersion="3.1.2">

    <!-- ******************** -->
    <!-- Supporting Documents -->
    <!-- ******************** -->

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

    <def:SupplementalDoc>
      <def:DocumentRef leafID="LF.ReviewersGuide"/>
      <def:DocumentRef leafID="LF.ComplexAlgorithms"/>
    </def:SupplementalDoc>
```

The official CDISC Define.XML specifications can be downloaded via the member page and these describe the XML tags and specify how the XML should be structured[2].

One issue with the define.xml is that when looking at the define.xml web display it is quite clear what information is required, but to really get it done the XML with its structure and tags must be understood. The %defineXML macro generates the define.xml file “out of the box”, so there is no need to understand the structure and tags, but of course the define information must be collected first.

DEFINEXML – CONTENT

The process for generating the define.xml can be very complex and is a time consuming task. The process consists of two parts, the major part being the information collection. Once the information is all together the second part is technically just the transfer into XML format. The second part in particular should not be problematic, but it often is. As tools are often available which generate just parts of the define.xml, people often take the partial XML output and edit it manually afterwards to enter the missing information, which is a laborious task.

A SAS macro may not be able to collect all information for any kind of environment and metadata structure, but a macro could be created and used to create the define structure from pre-collected information straight out of the box.

DEFINEXML GENERATION BY MACRO – WHY AND HOW

There are multiple tools available to generate the Define.XML, which is to be expected as each and every submission should deliver this. Apart from tools which provide metadata repositories and which probably also support define.xml generation, there is the Pinnacle21-community tool available (formerly called OpenCDISC Validator), which is commonly used as it is free and offers good functionality.[3] So why create a new tool and process?

Metadata repository tools in particular are limited in how they provide and modify additional metadata and not all details are visible to the customers. The Pinnacle21-community tool is very open and a lot of additional information can be entered via an exported Excel file which can be reimported following amendment. A major disadvantage of this tool is the user interaction which is optimized for using its built-in Graphical User Interface. With respect to traceability, everything should ideally be created in a fully automated or traceable process, in which case this tool might not be the right one. The Command Line Interface (CLI) is very useful, but not sufficient, especially as the change interaction takes place via Excel files.

A SAS macro provides full flexibility, full traceability and full transparency. Furthermore additional requirements like including Results Level Metadata for Analysis Results can easily be added by updating the macro. It does not require any licenses, but of course needs quality checks, documentation and validation unless you validate the define.xml itself and not the process by which it was created.

The SAS programming language is used, as in general the define.xml developers are data management programmers or statistical analysts with SAS expertise. Furthermore, as the data and probably the metadata too are also available as SAS datasets it makes sense to use this programming language.

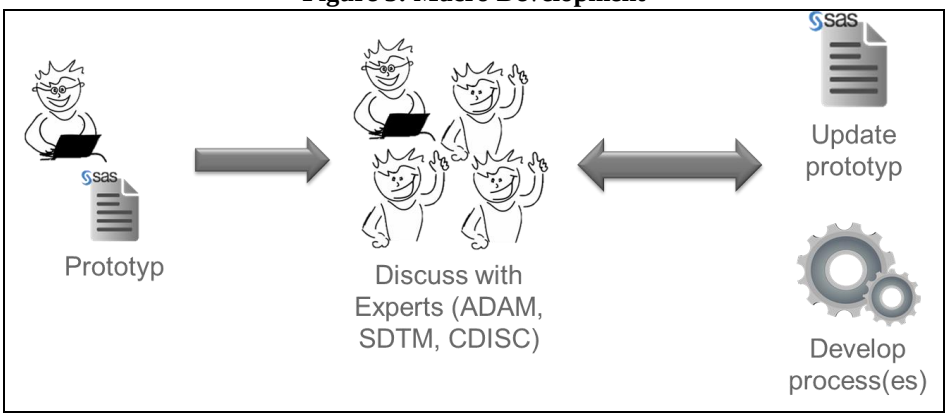
DO IT YOURSELF

To provide complete flexibility it was decided to create a %defineXML SAS macro. The only requirement for the macro is to “create a submission ready define.xml file”. Searching for define.xml papers[4], there is one excellent paper which describes a macro to generate the define.xml[5]. In 2016 another paper on this topic was published and is recommended to gain a better understanding of how the define.xml is constructed[6].

The macro was developed based on the source and content from the aforementioned paperand adapted to be runnable on custom collected data and to support both ADaM and SDTM. As it is quite generic it could also be used for other data formats, but the testing was done mainly on ADaM and SDTM. During the process of testing it became apparent quite early that the difficulty with creating the final define.xml is not so much achieving a correctly structured XML file, which is quite easy by using and adapting the source, but how and where to collect the information. So keeping the information collection separate from the planned macro makes the final macro creation quite straightforward.

After creating a simple prototype, the possible processes and first requirements were discussed with a team comprising an SDTM programmer, an ADaM programmer and a CDISC expert. Short and long term processes were worked out.

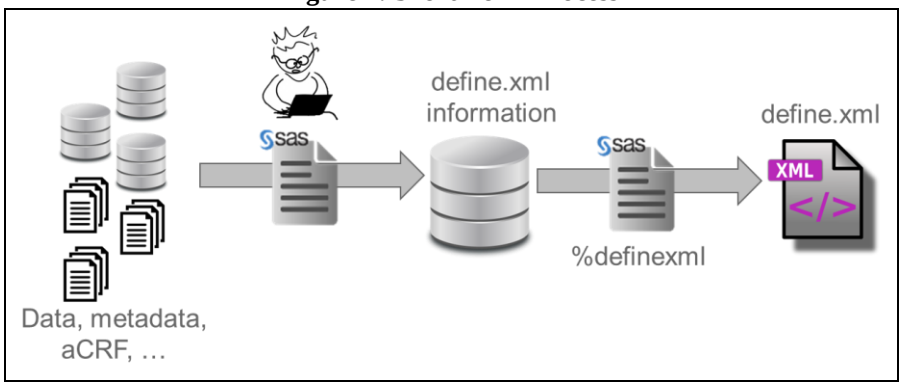
Figure 3: Macro Development



SHORT TERM PROCESS

For quick results, a short term process can be implemented, and this can already be used to create a submission ready define.xml. The data standards programmer is responsible for collecting and delivering the information in the form of SAS datasets for the macro. The macro is called and the resulting define.xml is checked with the Pinnacle21-community tool. Structure checks are run as well as content checks. In case issues arise or additional information needs to be added to the macro, the macro is updated and the process is repeated until a final define.xml is created.

Figure 4: Short Term Process



LONG TERM PROCESS

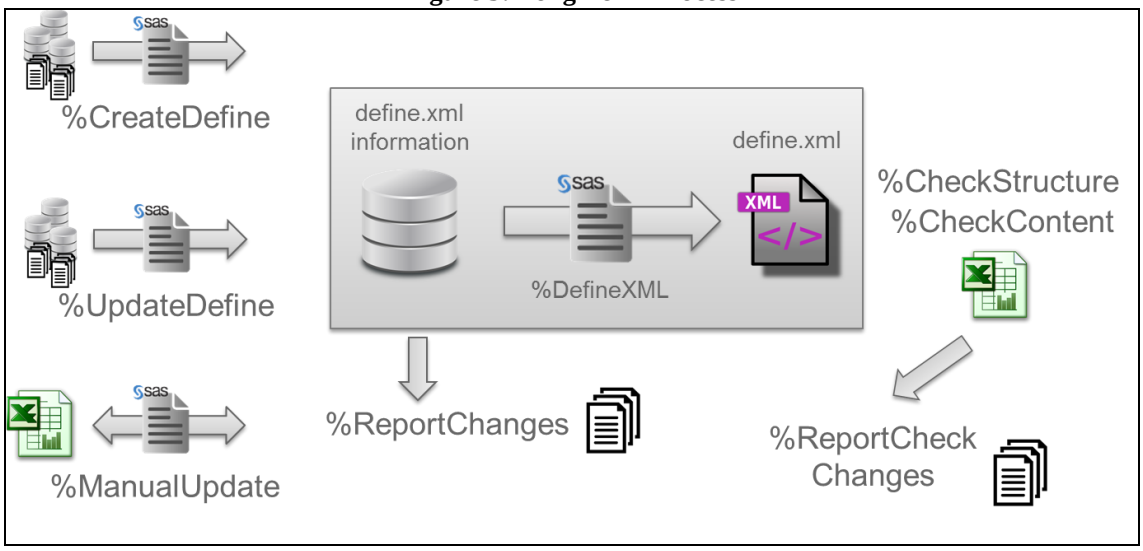
The short term approach can then be successively enhanced to create an optimal final process with full traceability and automation support. With sufficient real study experience it should be possible to collect most information completely automatically. Nevertheless the flexibility to update items manually should always remain available. The process should support continuous changes which become necessary as a study progresses.

To support this functionality the define.xml information package in the form of the SAS datasets could be encapsulated by macros. With sufficient standards also in place all information collection can be automated by macros as well, which would ideally mean having macros available for:

- Creating initial define information through metadata, data, aCRF reading tool, …
- Updating initial data according to any changes required (data, metadata, aCRFs, …)
- User interaction
  - Export to Excel for manual update
  - Import and incorporate into information datasets
  - On updates, ensure manual updates are kept (according to rules), e.g. database updates for example do not change manually provided CRF page numberings
- Creating an audit trail (if wanted, might not be needed for the define.xml, but is possible)
- Creating automatic reports
  - Run pinnacle21-community tool for checks
  - Provide reports on differences in the information datasets
  - Provide reports on differences in the check findings
- Reading in external define.XML for further modification

There are many possibilities for implementation, but to stay focused and get quick wins, the process should ideally be implemented not all at once, but piece-by-piece, “agile-style”, so as to benefit from the process as soon as possible.

Figure 5: Long Term Process



MACRO DEVELOPMENT

It was relatively easy to implement the creation of the define.xml with the XML tags. The source provided by the nice paper found through an internet search was used as the basis. Additional features were implemented, especially for ADaM. The file creation was adapted as it is much more appropriate to create just one file by using the “MOD” option to add to a file sequentially instead of creating multiple files to be concatenated afterwards. Tests with real study data brought additional functionalities to be implemented. Finally the macro is available to create the define.xml out of the box when you have all required information collected.

INTERFACE

The define.xml information is provided to the macro through eight SAS datasets:

- general information (header)
- dataset information (datasets)
- variable information (variables)
- value level information including “WhereClause” definitions (valuelevel)
- code list and controlled terminology information (codes)
- computational method information (methods)
- comments (comment)
- documents (documents)

To enable support for a minimal approach not all datasets have to be provided. Apart from the HEADER, DATASETS and VARIABLES information datasets, all other input datasets are optional. Whereas most variables of the interface datasets are straightforward, there are some with hidden features and restrictions. Detailed documentation on how the information datasets are transferred to the xml tags with respect to the Define-XML-2-0-Specifications is available in the PhUSE script repository[7].

DATASETS INFORMATION EXAMPLE

In the define.xml web view there is a dedicated datasets table available.

Figure 6: Dataset Table - Web View

Tabulation Datasets for Study CDISC01 (SDTM-IG 3.1.2)

Dataset	Description	Class	Structure	Purpose	Keys	Location	Documentation
TA	Trial Arms	TRIAL DESIGN	One record per planned Element per Arm	Tabulation	STUDYID, ARMCD, TAETORD	ta.xpt	
TE	Trial Elements	TRIAL DESIGN	One record per planned Element	Tabulation	STUDYID, ETCD	te.xpt	

Most content is filled through the DATASETS information dataset. Only the "Keys" are filled from the VARIABLES information dataset. Some content which is present in the define.xml is not even visible in the web view, for example the "Repeating" or "IsReferenceData" attributes. The following table displays which variable of the dataset information is used where:

Variable Name	Variable Label	Usage in Display Table Column
DOMAIN	Domain	Dataset
DESCRIPTION	Description	Description
CLASS	Class	Class
STRUCTURE	Structure	Structure
PURPOSE	Purpose	Purpose
ARCHIVELOCATION	Archive Location	Location
REPEATING	Repeating	Not displayed, just an XML attribute
ISREFERENCEDATA	Is Referenced Data	Not displayed, just an XML attribute
COMMENT	Comment OID	Documentation contains the description of the corresponding comment OID in the INDAT_COMMENTS dataset
ORDID	Order for Domain Display	Not displayed, used to order the datasets for the display. Explanation: The dataset with the lowest order number is displayed as the first item in the table.

WHERE CLAUSE HANDLING

The "WhereClause Definition" is handled in a special way by the macro. The complete definition of the WhereClause is not done through a separate sheet with all single options, but through one identifier using special rules. The identifier is made up of multiple parts. The first part is "WC" indicating that this is a WhereClause. The second part describes the domain or dataset. Then two parts always follow the variable and the variable value. The final XML elements are created from this construct. This type of WhereClause handling only supports the equals operator, which should be sufficient for most studies, especially where the information collection is automated.

Figure 7: WhereClause Definition - Web View

Value Level Metadata - LB [LBORRES]

Variable	Where	Type	Length / Display Format
LBORRES	LBTESTCD EQ GLUC AND LBCAT EQ URINALYSIS AND LBSPEC EQ URINE AND LBMETHOD EQ DIPSTICK	text	200

The above Where column is created by using the following OID:
IT.LB.LBTESTCD.GLUC.LBCAT.URINALYSIS.LBSPEC.URINE.LBMETHOD.DIPSTICK

So for each pair the equal check is put into the define.xml file with the conform and the final complex xml code is constructed:

```
<def:WhereClauseDef
     OID="WC.LB.LBTESTCD.GLUC.LBCAT.URINALYSIS.LBSPEC.URINE.LBMETHOD.DIPSTICK">
  <RangeCheck SoftHard="Soft" def:ItemOID="IT.LB.LBTESTCD" Comparator="EQ">
      <CheckValue>GLUC</CheckValue>
  </RangeCheck>
  <RangeCheck SoftHard="Soft" def:ItemOID="IT.LB.LBCAT" Comparator="EQ">
      <CheckValue>URINALYSIS</CheckValue>
  </RangeCheck>
  <RangeCheck SoftHard="Soft" def:ItemOID="IT.LB.LBSPEC" Comparator="EQ">
```

```
      <CheckValue>URINE</CheckValue>
    </RangeCheck>
    <RangeCheck SoftHard="Soft" def:ItemOID="IT.LB.LBMETHOD" Comparator="EQ">
      <CheckValue>DIPSTICK</CheckValue>
    </RangeCheck>
  </def:WhereClauseDef>
```

%DEFINEXML IN PHUSE SCRIPT REPOSITORY

The macro, some documentation and an example are available as open source in the PhUSE Script repository[8] straight out of the box. It can simply be downloaded and it runs in SAS. The development was performed in SAS 9.2, but other versions might be supported as well.

The following files are available:

File	Description
src/definexml.sas	Contains the %definexml macro which is called by the user
src/_definexml_attributes_for.sas	Macro to create expected ATTRIB statements for input datasets
src/example_adam.sas	Contains an example call for the %definexml macro creating an ADAM define.xml
src/example_adam.xlsx	Information datasets in Excel format for easier review
results/example_adam.xml	Output of the example call
results/define2-0-0.xsl	Style Sheet file which can be used[9]
docs/DefineXML – Macro notes.docx	Technical macro notes, especially how the datasets are transferred to the xml parts

DEFINEXML.SAS

The macro itself is available as a SAS file. The source does not require any sub macros and can be included and used by any other SAS program. It was developed and tested in SAS 9.2 on a Unix system but likely supports other operating systems and newer SAS versions as well.

The macro is well structured and commented, so for a SAS programmer the macro is easy to read. The most important point is that the user can see how the information is transformed into the XML schema. The following figure displays an extract from the macro.

Figure 8: %DefineXML Macro Extract

```
DATA _NULL_;
    SET _valuelevel END=eof;
    BY valuelistoid;
    FILE output MOD LRECL=2000  ENCODING="UTF-8";

    IF _N_ = 1
    THEN DO;
        PUT @5 "<!-- ****************************************** -->" /
            @5 "<!-- VALUE LEVEL LIST DEFINITION INFORMATION  ** -->" /
            @5 "<!-- ****************************************** -->";
    END;
    IF FIRST.valuelistoid
       THEN PUT @5 '<def:ValueListDef OID="' valuelistoid +(-1) '">';

    PUT @7 '<ItemRef ItemOID="' itemoid +(-1) '"' /
        @16 'OrderNumber="' valueorder +(-1) '"' /
        @16 'Mandatory="' mandatory +(-1) '"' ;
    IF computationmethodoid NE ''
        THEN PUT @16 'MethodOID="' computationmethodoid +(-1) '"';
```

_DEFINEXML_ATTRIBUTES_FOR.SAS

This SAS support macro can be used to create the ATTRIB statements for the required input datasets. This creates a dataset with the correct formats, labels and variable formats which are useful for the %DefineXML macro.

Figure 9: %_DefineXML_Attributes_For Macro Extract

```
%IF %UPCASE(&data.) = HEADER
%THEN %DO;
    ATTRIB FILEOID          LENGTH = $200  FORMAT = $200.  LABEL = 'File Object ID' ;
    ATTRIB STUDYOID         LENGTH = $200  FORMAT = $200.  LABEL = 'Study Object ID' ;
    ATTRIB STUDYNAME        LENGTH = $200  FORMAT = $200.  LABEL = 'Study Name' ;
    ATTRIB STUDYDESCRIPTION LENGTH = $2000 FORMAT = $2000. LABEL = 'Study Description' ;
    ATTRIB PROTOCOLNAME     LENGTH = $200  FORMAT = $200.  LABEL = 'Protocol Name' ;
    ATTRIB STANDARD         LENGTH = $200  FORMAT = $200.  LABEL = 'Standard' ;
    ATTRIB VERSION          LENGTH = $200  FORMAT = $200.  LABEL = 'Standard Version' ;
    ATTRIB SCHEMALOCATION   LENGTH = $200  FORMAT = $200.  LABEL = 'Schema Location' ;
    ATTRIB STYLESHEET       LENGTH = $200  FORMAT = $200.  LABEL = 'Stylesheet' ;
    ATTRIB ORIGINATOR       LENGTH = $200  FORMAT = $200.  LABEL = 'Originator' ;
%END;
```

EXAMPLE_ADAM.SAS

The example program generates define.xml data from a dummy study and calls the %DefineXML macro. Some DATALINES are longer than 255 characters, so this program should not be included by another program, as otherwise the text affected will be cropped. This program uses the %_definexml_attributes_for submacro to create the final information datasets.

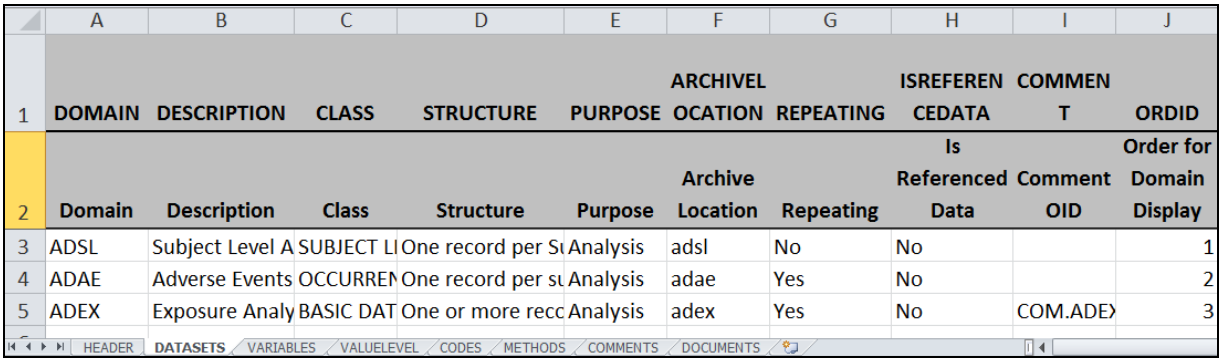
The following source is used for example to create the HEADER input dataset.

```
DATA header;
   %_definexml_attributes_for(header);
   fileoid          = "Study12345-defineXML-2.0-ADAM";
   studyoid         = "Study12345";
   studyname        = "Study 12345";
   studydescription = "This is the study description, this study
                       is a dummy study used as example for the define.xml generation.";
   protocolname     = "Protocol 98765/12345";
   standard         = "ADaM-IG";
   version          = "1.0";
   schemalocation   = "http://www.cdisc.org/ns/odm/v1.3";
   stylesheet       = "define2-0-0.xsl";
   originator       = "Company XY";
   OUTPUT;
RUN;
```

EXAMPLE_ADAM.XLSX

The corresponding example_adam.xlsx file displays the input datasets used for the %DefineXML macro in Excel format. This file provides a nice overview of which information can be stored in the defineXML and which should be provided.

Figure 10: Example Input Datasets in Excel Format



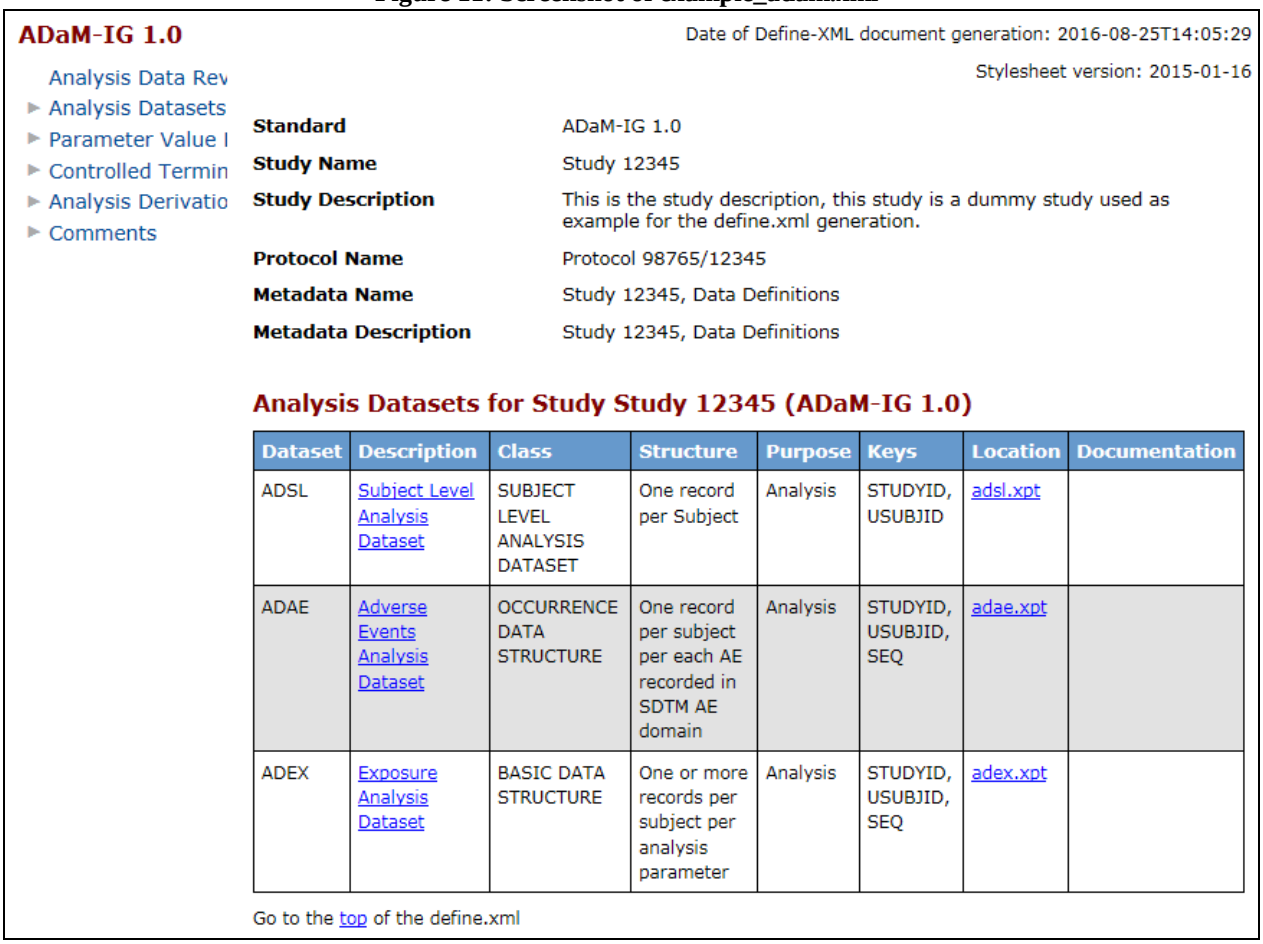
	A	B	C	D	E	F	G	H	I	J
1	DOMAIN	DESCRIPTION	CLASS	STRUCTURE	PURPOSE	ARCHIVELOCATION	REPEATING	ISREFERENCEDATA	COMMENT	ORDID
2	Domain	Description	Class	Structure	Purpose	Archive Location	Repeating	Is Referenced Data	Comment OID	Order for Domain Display
3	ADSL	Subject Level A	SUBJECT LI	One record per Su	Analysis	adsl	No	No		1
4	ADAE	Adverse Events	OCCURREN	One record per su	Analysis	adae	Yes	No		2
5	ADEX	Exposure Analy	BASIC DAT	One or more recc	Analysis	adex	Yes	No	COM.ADEX	3

HEADER | DATASETS | VARIABLES | VALUELEVEL | CODES | METHODS | COMMENTS | DOCUMENTS

EXAMPLE_ADAM.XML AND DEFINE2-0-0.XSL

The resulting output file from the example_adam.sas file is stored as example_adam.xml. This file can be displayed with a text viewer or as a web view together with the style sheet file.

Figure 11: Screenshot of example_adam.xml



ADaM-IG 1.0

Date of Define-XML document generation: 2016-08-25T14:05:29

Stylesheet version: 2015-01-16

- Analysis Data Rev
- Analysis Datasets
- Parameter Value l
- Controlled Termin
- Analysis Derivatio
- Comments

Standard	ADaM-IG 1.0
Study Name	Study 12345
Study Description	This is the study description, this study is a dummy study used as example for the define.xml generation.
Protocol Name	Protocol 98765/12345
Metadata Name	Study 12345, Data Definitions
Metadata Description	Study 12345, Data Definitions

Analysis Datasets for Study Study 12345 (ADaM-IG 1.0)

Dataset	Description	Class	Structure	Purpose	Keys	Location	Documentation
ADSL	Subject Level Analysis Dataset	SUBJECT LEVEL ANALYSIS DATASET	One record per Subject	Analysis	STUDYID, USUBJID	adsl.xpt	
ADAE	Adverse Events Analysis Dataset	OCCURRENCE DATA STRUCTURE	One record per subject per each AE recorded in SDTM AE domain	Analysis	STUDYID, USUBJID, SEQ	adae.xpt	
ADEX	Exposure Analysis Dataset	BASIC DATA STRUCTURE	One or more records per subject per analysis parameter	Analysis	STUDYID, USUBJID, SEQ	adex.xpt	

Go to the top of the define.xml

DEFINEXML – MACRO NOTES.DOCX

Documentation is provided through the "DefineXML – Macro Notes" file. Next to a disclaimer and MIT License information, the technical translation of the dataset to the corresponding XML tags from the Define-XML-2.0 specifications is documented.

Figure 12: Extract from DefineXML - Macro Notes

The folloing XML Attributes are set for the MethodDef definition:

XML Attribute	Origin
OID	ComputationMethodoID
Name	ComputationMethodName
Type	ComputationMethodType
Description TranslatedText	Description
def:DocumentRef leafID	PDFLeaf (if provided)
def:PDFPageRef PageRefs	PDFPage (if provided)
Type	"PhysicalRef"
def:PDFPageRef PageRefs	PDFDestination (if provided and PDFPage not)
Type	"NamedDestination"

USABILITY

This macro only translates pre-collected define information to the structured xml format. The information collection part is highly dependent upon the environment and will be company or even study specific. As the separation has already been done at that point, the generic %DefineXML macro will be useful for any company which needs to create a define.xml file.

Open source programs have issues with respect to validation. But if the define.xml is validated as a file, then the process which created the file, even if this involved a non-validated open source macro, is not relevant.

COLLABORATION

A major advantage of the PhUSE Script Repository is the option to work collaboratively together. The macro, documentation and examples are available for further enhancement. Even though a lot of files are provided, additional things could be done.

Possible Enhancement	Description
Results Level Metadata Support	Currently Results Level Metadata is not included in the macro. A new input information dataset "results" might be added providing the interface for this type of metadata and translating the information into the corresponding XML format.
Example(s) for SDTM	There is currently just an example for ADaM provided. An SDTM example would be valuable.
Example(s) for ADAM	There is one ADaM example, which might be changed and enhanced.
Guidance Document / How to Document	There could be additional documentation about how to fill the various fields. Descriptions can be found in the define-xml-2-0 specifications, but additional guidance would be helpful for the users.
New generic information collection macro	Process wise a new macro to collect the define.xml information and store it in the information datasets would be useful. As this should be environment independent, simple processes like reading input datasets, e.g. with PROC CONTENTS, could be implemented.
New read define.xml macro	To have a macro available which reads a define.xml file and allows the data to be updated in SAS or via an Excel import/export would be useful.

CONCLUSION

The define.xml creation process should be focused on the content, and the people involved should not have to struggle with the XML format, which is just a different machine readable presentation of information. The %DefineXML macro provides an out of the box solution to generate an XML structured file from pre-filled information datasets.

As this paper points out, anyone can just download and use this tool. Basic documentation is available. Of course much more could be done and if the community takes this tool up, then quite a lot of surrounding functionality can be added. As the source is available, full transparency is provided. The process is implemented in a SAS macro. Additional functionality could easily be implemented by SAS programmers which are typically the ADaM and SDTM experts out there in the various companies.

REFERENCES

[1] CDISC – Define-XML definition
http://www.cdisc.org/define-xml

[2] CDISC – Define-XML-2-0 ReleasePackage20140424 including define specifications
O:\Wr_CDPT\Apps+Tools\SDTM\Define-XML-2-0_ReleasePackage20140424

[3] Pinnacle21 community tool
https://www.pinnacle21.net/

[4] SAS Proceedings search page
http://www.lexjansen.com/

[5] Creating Define.xml v2 Using SAS® for FDA Submissions, PharmaSUG 2014 – Paper AD02, Qinghua (Kathy) Chen and James Lenihan
http://www.lexjansen.com/pharmasug/2014/AD/PharmaSUG-2014-AD02.pdf

[6] A SAS® Macro Tool to Automate Generation of Define.xml V2.0 from SDTM Specification for FDA Submission, PharmaSUG2016 - Paper SS08, Min Chen and Xiangchen (Bob) Cui
http://www.lexjansen.com/pharmasug/2016/SS/PharmaSUG-2016-SS08.pdf

[7] Technical defineXML macro notes on PhUSE Script Repository
https://github.com/phuse-org/phuse-scripts/tree/master/lang/SAS/datahandle/defineXML/docs

[8] DefineXML macro and related files location on PhUSE Script Repository
https://github.com/phuse-org/phuse-scripts/tree/master/lang/SAS/datahandle/defineXML

[9] Style sheet produced by the CDISC community, used style sheet file is from 2015-01-16
http://wiki.cdisc.org/display/PUB/Stylesheet+Library

RECOMMENDED READING

1) Creating Define.xml v2 Using SAS® for FDA Submissions, PharmaSUG 2014 – Paper AD02, Qinghua (Kathy) Chen and James Lenihan
http://www.lexjansen.com/pharmasug/2014/AD/PharmaSUG-2014-AD02.pdf

2) Define_xml_2_0 release package including specifications and examples (require CDISC membership)
http://www.cdisc.org/system/files/members/standard/define_xml_2_0_releasepackage20140424.zip

CONTACT INFORMATION

Contact the author at:

Author:	Katja Glaß
Company:	Bayer Pharma AG
Address:	Sellerstraße 32, 13353 Berlin, Germany
Email:	Katja.Glass@Bayer.com