PharmaSUG 2013 - Paper AD19

Automating Validation of Define.xml using SAS®

Prafulla Girase, Biogen Idec, Cambridge, MA
Robert Agostinelli, Sunovion Pharmaceuticals, Marlborough, MA

ABSTRACT

There are various methods that can be used to validate the define.xml. A community version of OpenCDISC is a popular and efficient way to validate define.xml. However, since it may not cover everything, it is important to supplement it with other validation checks to ensure the accuracy, validity and completeness of your define.xml. To do this additional validation manually can be cumbersome and time consuming. Since the define.xml is a machine readable file, it is possible to automate many of these additional validation checks. In this paper, we discuss an automated SAS-based approach. One important technique that makes this possible is getting define.xml contents into SAS data sets using a XMLMap. Once the define.xml contents are transformed into SAS data sets, one can perform a variety of tasks or checks using the flexibility and power of SAS. We developed several custom validation checks in our macro. The checks are related to issues that are within the define.xml itself, and also issues related to inconsistencies between the define.xml and submission XPT files. The paper discusses a method for getting define.xml contents into SAS data sets, all the validation checks that can be performed, and also presents samples of validation output. While we have not provided the entire SAS code in this paper, important snippets of code necessary for someone to get started are provided.

INTRODUCTION

The purpose of this paper is to introduce a SAS based approach for validation of define.xml. The techniques presented in this paper are developed for define.xml prepared using define.xml specification version 1.0. The following figure gives a high level overview of the basic concept of this paper.

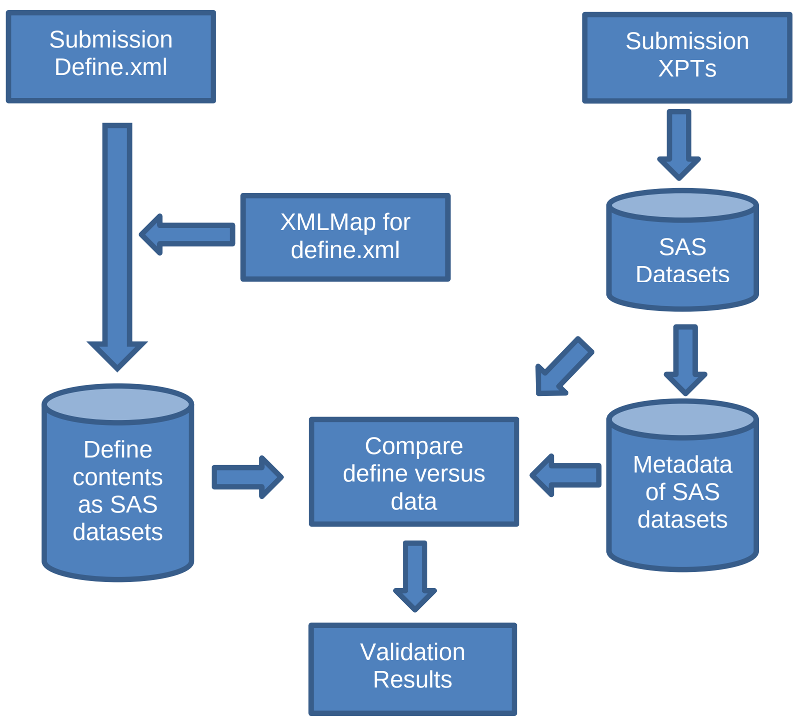

Figure 1. Overview of Basic Paper Concept

The XMLMap allows transformation of contents of define.xml into SAS data sets. The paper briefly explains how this transformation can be done. However, the SAS XMLMap file itself can't be provided. It can be created using SAS XML mapper. The technique of creating that map is beyond the scope of this paper. This technique is discussed in greater details in Lex Jansen's paper at SAS Global Forum 2010 [1]. Instead of providing the entire SAS code, important snippets of code are provided in the paper. Using the snippets of code, someone with a good understanding of SAS data step manipulation and SAS macros should be able to write the entire code. In any case, the code for this paper was written in UNIX operating environment. Therefore, developers would need to make some adjustments to the code based on their operating environment.

PICTORIAL OVERVIEW OF FLOW WITHIN THE MACRO

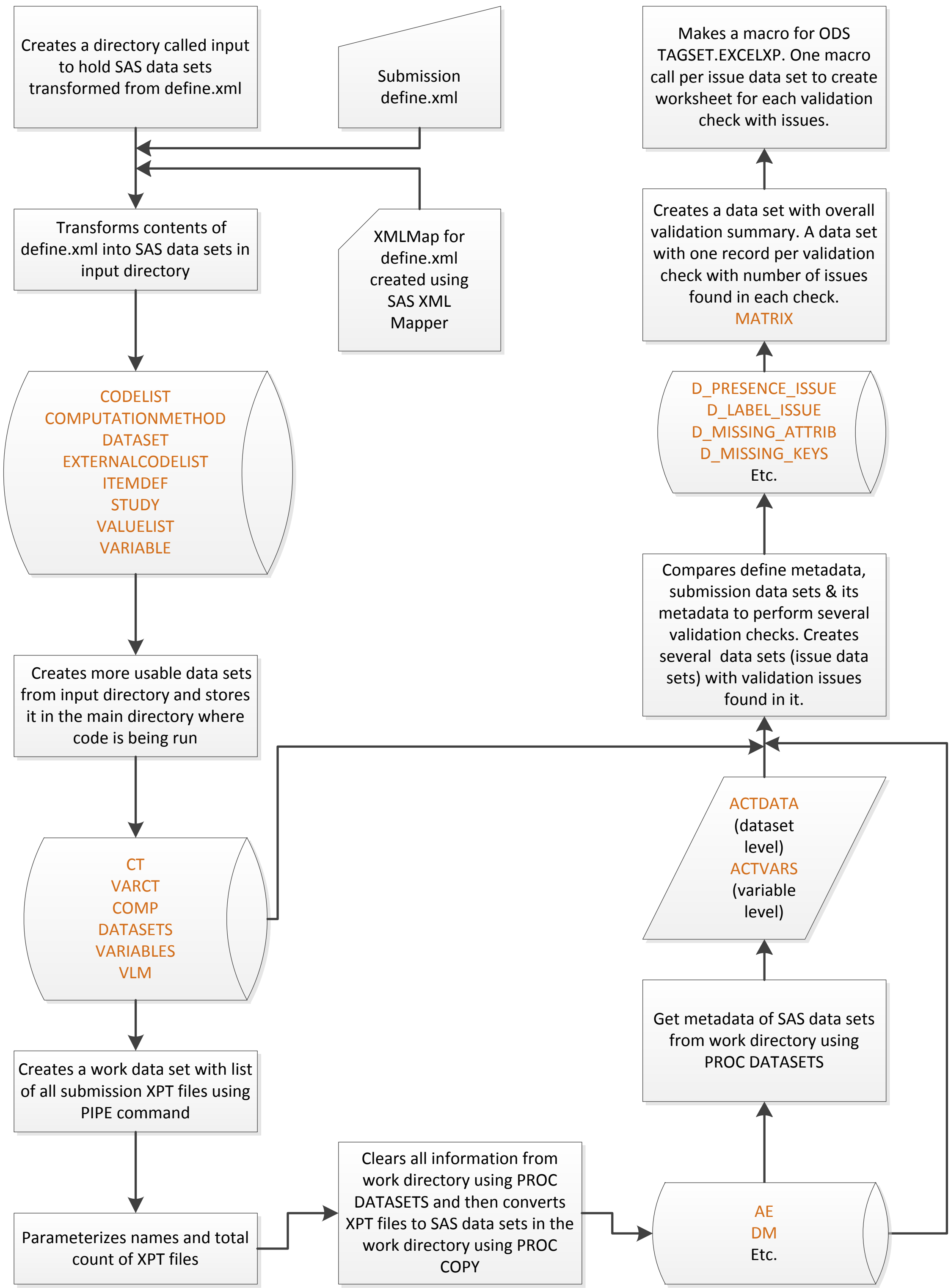

Figure 2. Flow Within the Macro

MACRO INPUT PARAMETERS AND MACRO CALL

The Figure 2 above gives a pictorial overview of the flow within our macro. We call this macro as %checkdefine and it requires 3 input parameters as listed in Table 1 below:

Input Parameter	*Description*
defxptpath	Path where define.xml to be validated and XPT files are located
xmlmappath	Path where XMLMap of define.xml is located
study_std	Name of the study and study standard (This becomes a part of the name of validation output file)

Table 1. Macro Input Parameters

Below is a sample macro call:

```
%let defxptpath = %str(/biostats/submission/tabulations/sdtm) ;
%let xmlmappath = %str(/biostats/submission/adhoc/map) ;
%let study_std = 102xy102_sdtm ;

%checkdefine ;
```

The remainder of the paper goes into further details of important steps withing this macro.

CREATING A DIRECTORY FOR STORING DEFINE.XML CONTENTS

While one can easily manually create a directory, we are letting SAS create it using the following macro [2]. If the directory already exists, then macro will not try to create it.

```
%macro chk_dir(dir=) ;
   %local rc fileref ;
   %let rc = %sysfunc(filename(fileref,&dir)) ;
   %if %sysfunc(fexist(&fileref))  %then
      %put NOTE: The directory "&dir" exists ;
   %else
     %do ;
         %sysexec mkdir   &dir ;
         %put %sysfunc(sysmsg()) The directory has been created. ;
   %end ;
   %let rc=%sysfunc(filename(fileref)) ;
%mend chk_dir ;

%chk_dir(dir=./input) ;
```

TRANSFORMATION OF DEFINE.XML CONTENTS INTO SAS DATA SETS

This transformation is possible using XMLMap and XML engine of SAS. The first step here is to create XMLMap for define.xml using SAS XML Mapper. The XMLMap tells the SAS XML engine how to map the content in the hierarchical define.xml file to rows and columns in the rectangular SAS tables [1]. Once we have XMLMap, the following code can transform contents of define.xml into SAS data sets [1]. The parameter DEFXPTPATH holds path of define.xml location and XMLMAPPATH holds path of XMLMap location.

```
filename  define "&defxptpath./define.xml";
filename  SXLEMAP "&xmlmappath./define2sas.map";
libname   define xml xmlmap=SXLEMAP access=READONLY;
libname   indata  "./input";

proc datasets library=define;
 copy out=indata;
run;
```

Using the XMLMap we have, above step creates data sets as shown in Part A of Figure 3. Then we do some manipulation to create data sets shows in Part B of Figure 3. The purpose of manipulation is to make the data sets more useful. The data sets that may be created when you use your version of XMLMap, may have a different structure and/or names. The key is to understand the structure and primary key of data sets created using XMLMap. Once you understand it, then you can use your own programming style to create data sets that you can work with.

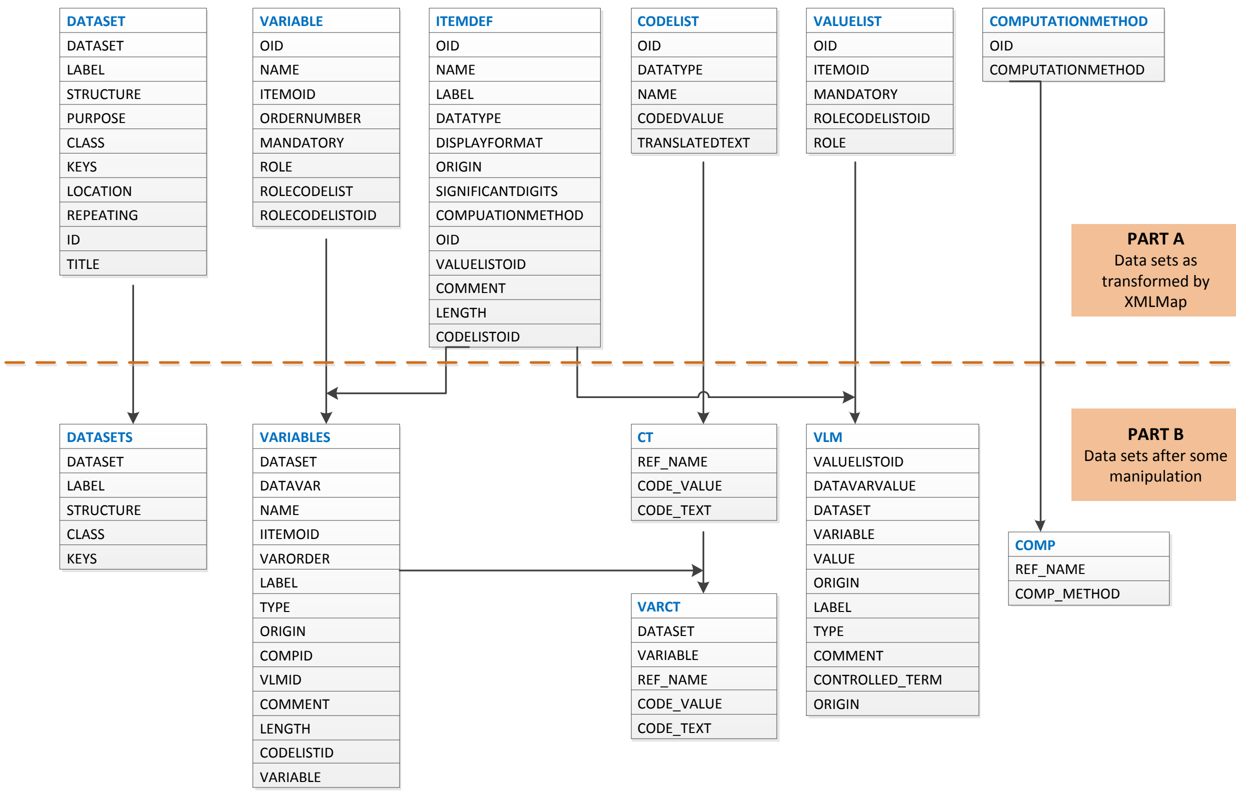

Figure 3. Transformation of Define.xml Contents into SAS Data Sets

CONVERSION OF SUBMISSION XPT FILES TO SAS DATA SETS

The XPT files (SAS XPORT Transport File Format) are what is submitted to the regulatory agency (not SAS data sets) as per FDA's Study Data Specifications [3]. The metadata in define.xml should support metadata associated with these files. Therefore, we use these XPT files (instead of SAS data sets that were used to create the XPT files) to perform the validation. However, the XPT files need to be converted to SAS data sets to be able to get the metadata associated with it. First, we create macro parameters to hold names and count of XPT files using the PIPE command and PROC SQL as shown in the code below.

```
filename dirlist pipe "ls &defxptpath. *.xpt";
data dirlist ;
   infile dirlist  lrecl=200 truncover;
   length line $ 200 ;
   input line $ char200. ;
   if upcase(scan(line,-1,.)) ne "XPT" then delete ;
   length dname $20;
   dname=scan(line,-2,.);
   keep dname line ;
run;

proc sql noprint;
  select count(distinct dname) into :totfiles from dirlist ;
quit ;
proc sql noprint;
  select distinct dname into :dsn1 - :dsn%left(&totfiles.) from dirlist ;
quit ;
```

Before we do the conversion of XPT files, we are clearing all the information present in WORK directory as shown in the code below.

```
proc datasets library=work kill ;
quit ;
```

```
run ;
```

Now the conversion of XPT files to SAS data sets in WORK directory will ensure that any metadata stored in the work directory is associated with submission data sets. Using the macro parameters created above, we create the following macro to do the conversion of XPT files using PROC COPY.

```
%macro xpt2sas ;
%local i ;
%do i=1 %to &totfiles. ;
   libname xportin xport "&defxptpath./&&dsn&i...xpt" ;
   proc copy in=xportin out=work;
   run;
%end ;
%mend xpt2sas ;

%xpt2sas ;
```

READING METADATA OF SUBMISSION DATA SETS

We read the metadata associated with submission data sets from the WORK directory using the CONTENTS statement of PROC DATASETS as shown in the code below.

```
proc datasets lib=work;
  contents data=_all_ directory out=work.metadata (keep=libname mem: name type
length varnum label)  noprint;
run ;
```

From the data set METADATA, we create a data set level metadata file called ACTDATA and a variable level metadata file called ACTVARS.

PERFORMING VALIDATION CHECKS

Now that we have define.xml metadata (data sets in Part B of Figure 3), submission data sets and its metadata (ACTVARS and ACTDATA) available as SAS data sets, we have the power of SAS to perform several validation checks. In our macro, we perform validation checks as shown in Table 2 below. You are not limited to the checks listed here. Once you have understood the structure and contents of all data sets created for this validation, you can develop as many checks as possible. The process that you use for creating define.xml will also drive what types of checks are more important than the others. The advantage of this approach is that you have the ability and flexibility to add/delete the checks based upon your validation needs.

For each validation check, we output a SAS data set which can be used later to produce output in Excel. The following SAS code is an example of validation check for check #1. The LIBNAME OUTDATA is where all define.xml metadata is stored. The data set DATASETS from that location is being compared with submission data set metadata ACTDATA. It produces a data set called D_PRESENCE_ISSUE with all issues related to this validation check.

```
proc sort data=outdata.datasets out=datasets_sorted ;
  by dataset ;
run ;
data d_presence_issue (keep=message dataset)
     d_both ;
     attrib message length=$50 ;
     merge datasets_sorted (in=a keep=dataset label keys rename=(label=deflabel))
           actdata (in=b keep=dataset label rename=(label=actlabel)) ;
     by dataset ;
     if a & NOT b then do ;
        message="Dataset in define, not in actual data" ;
        output d_presence_issue;
     end ;
     if b & NOT a then do ;
        message="Dataset in actual data, not in define" ;
        output d_presence_issue;
     end ;
     if a & b then do ;
        output d_both ;
```

```
        end ;
    run ;
```

CHECK #	*CHECK RULE*
Check #1	Data Presence Issues
Check #2	Dataset Label Mismatches Between Actual Data & Define
Check #3	Dataset Label Missing in Actual Data or Define
Check #4	(Within Define.xml) one or more dataset attributes (structure, class, or keys) are missing
Check #5	Keys have variable/variables that is not found in the data
Check #6	Domain:XX Keys listed in the define do not represent uniqueness in the data
Check #7	Variable Presence Issues Between Actual Data & Define
Check #8	Variable label mismatch between actual data and define
Check #9	Variable type mismatch between actual data and define
Check #10	Variable length mismatch between actual data and define
Check #11	Variable order mismatch between actual data and define
Check #12	(Within Define.xml) Controlled Terminology reference presence issues
Check #13	Controlled terminology value is present in the data but not in define
Check #14	(Within Define.xml) Value Level Metadata reference presence issues
Check #15	Value Level Metadata (VLM) value is present in the data but not in define (or vice versa)
Check #16	(Within Define.xml) Computational Method reference presence issues

Table 2. Validation Checks Being Performed in the Program

PRODUCING VALIDATION OUTPUT

We discussed earlier that a data set is produced for each validation check. We also create a data set that has summary of validation results. We call this data set as MATRIX and it has one record per validation check with two variables in it. One variable holds description of what the check is and the other variable has count of how many issues were found for that check. We create this validation summary data set based on individual validation check data sets. After creating this data set, we have all data sets we need to create an Excel output of validation results. We use ODS TAGSETS.EXCELXP to produce output in Excel. In the main ODS TAGSETS.EXCELXP statement, we use option of INDEX='yes' so we can get a table of contents with hyperlinks for each worksheet in the workbook (shown in the code below).

```
ods tagsets.ExcelXP file="&study._&standard._chkdefine_&sysdate9..xls"
                                  style=sansPrinter options(index='yes');
```

We put ODS TAGSETS.EXCELXP statement required for every worksheet into a macro and then execute macro for every data set. Below is an example with macro call for check #1.

```
     %macro exceltab(chknum=, indat=, header=, sheet=, vars=, col=) ;
      ods tagsets.ExcelXP options(embedded_titles='yes'
                                  autofilter='all'
                                  orientation='landscape'
                                  zoom='85'
                                  scale='85'
                                  autofit_height='yes'
                                  print_footer=' '
                                  sheet_interval='none'
                                  sheet_name="Check&chknum.: &sheet."
                                  row_repeat='1-3'
                                  frozen_headers='3'
                                  absolute_column_width="&col."
```

```
                              );

        title1 j=L "Check #&chknum. &header." ;
        title2 ;
        proc print data=&indat. width=min noobs;
        var &vars. ;
        run ;
%mend exceltab ;
%exceltab(chknum=1, indat=d_presence_issue, header=Data Presence Issues,
          sheet=Data Presence Issues, vars=message dataset, col=%str(40,10)) ;
```

For validation summary worksheet, we use PROC REPORT instead of macro above (macro uses PROC PRINT) to get colors based on whether issue is present or not. The Display 1 below shows how validation summary is displayed in Excel. We generate colors using style attribute in PROC REPORT [4]. The issues presented here are artificially created by modifying the define.xml or submission data being used for this paper.

Summary of Validation Results

VALIDATION CHECK	ISSUES
Check #1 Data Presence Issues	3
Check #2 Dataset Label Mismatches Between Actual Data & Define	8
Check #3 Dataset Label Missing in Actual Data or Define	0
Check #4 (Within Define.xml) one or more dataset attributes (structure, class, or keys) are missing	3
Check #5 Keys have variable/variables that is not found in the data	0
Check #6.01 Domain:DS Keys listed in the define do not represent uniqueness in the data	3
Check #6.02 Domain:FA Keys listed in the define do not represent uniqueness in the data	1
Check #7 Variable Presence Issues Between Actual Data & Define	6
Check #8 Variable label mismatch between actual data and define	2
Check #9 Variable type mismatch between actual data and define	0
Check #10 Variable length mismatch between actual data and define	17
Check #11 Variable order mismatch between actual data and define	12
Check #12 (Within Define.xml) Controlled Terminology reference presence issues	12
Check #13 Controlled terminology value is present in the data but not in define	0
Check #14 (Within Define.xml) Value Level Metadata reference presence issues	2
Check #15 Value Level Metadata (VLM) value is present in the data but not in define (or vice versa)	45
Check #16 (Within Define.xml) Computational Method reference presence issues	3

Display 1. Validation Summary Result Worksheet

The Displays 2 and 3 below shows how the validation output is displayed in Excel for check #1 & check #2 respectively. The check #1 looks into data set presence issues (i.e. data set is being submitted as XPT but not listed in the define.xml or vice versa).

Check #1 Data Presence Issues

MESSAGE	DATASET
Dataset in actual data, not in define	AE
Dataset in actual data, not in define	DV
Dataset in actual data, not in define	SUPPDV

Display 2. Validation Result for Check #1

The check #3 compares internal labels associated with data sets and compares it with data set labels listed in the description column of define.xml. If there is a mismatch, then issue is listed in the validation output for this check.

Check #2 Dataset Label Mismatches Between Actual Data & Define			
MESSAGE	DATASET	DEFLABEL	ACTLABEL
Dataset label mismatch between actual data and define	FA	Findings About	Findings About Events or Interventions
Dataset label mismatch between actual data and define	LB	Laboratory Tests	Laboratory Tests Results
Dataset label mismatch between actual data and define	PC	PK Concentrations	Pharmacokinetic Concentrations
Dataset label mismatch between actual data and define	PE	Physical Exam	Physical Examination
Dataset label mismatch between actual data and define	PP	PK Parameters	Pharmacokinetic Parameters
Dataset label mismatch between actual data and define	SUPPLB	Supplemental Qualifiers LB	Supplemental Qualifiers for LB
Dataset label mismatch between actual data and define	SUPPPC	Supplemental Qualifiers PC	Supplemental Qualifiers for PC
Dataset label mismatch between actual data and define	SUPPPP	Supplemental Qualifiers PP	Supplemental Qualifiers for PP

Display 3. Validation Result for Check #2

CONCLUSION

This approach of automated supplemental validation of define.xml could be very helpful to improve efficiency and to create a define.xml that is complete, accurate, and valid. Additionally, it helped us to validate our home grown define.xml generating software during the development phase. We were able to identify some issues that could have been missed if we had just relied on OpenCDISC or manual checking. Some of the validation checks presented in this paper overlap with the checks performed by OpenCDISC. However, one could make their list of validation checks mutually exclusive to OpenCDISC checks.

We developed this code for validation of define.xml created using define.xml specification version 1.0. The approach and concept presented in this paper should work for define.xml prepared using version 2.0 of specifications. However, a new XMLMap will need to be created and the SAS code will need to be modified based on the structure and contents of define.xml metadata.

REFERENCES

1. Jansen, Lex. 2010. Understanding the define.xml File and Converting It to a Relational Database. SAS Global Forum 2010. 157-2010.

2. www.sas.com. How Can I Check If a Directory Already Exists and If Not, Create it? Customer Support – Hints and Tips. Available at: http://www.sas.com/offices/europe/uk/support/sas-hints-tips/ht1_mar04.html

3. FDA. Study Data Specifications Version 2.0. Page 4.

4. Thaer Baroud, John Senner, Paul Johnson. 2006. Incorporating PROC FORMAT, PROC REPORT, and ODS Style Definitions to Track Sources of Records with Poor Data. SUGI 2006. 129-31

CONTACT INFORMATION

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

Prafulla S Girase
Senior Analyst III, Statistical Submissions Management
Biogen Idec Inc.
14 Cambridge Center
Cambridge MA 02142
Phone: 617-679-2651
Email: prafulla.girase@biogenidec.com

Robert Agostinelli
Associate Director, Statistical Programming
Sunovion Pharmaceuticals Inc.
84 Waterford Drive
Marlborough MA 01752
Phone: 508-357-7690
Email: robert.agostinelli@sunovion.com