PharmaSUG 2013 - Paper BB05

OpenCDISC: Beyond Point and Click

Frank DiIorio, CodeCrafters, Inc., Philadelphia PA

ABSTRACT

The OpenCDISC validation framework has rapidly gained popularity with programmers and statisticians who have to validate data compliance with ADaM, SDTM and other CDISC standards. The most common way to use the software is via the user interface. Its clean design and simplicity make it an appealing tool. Given that the software is open source, however, it makes sense to go "under the hood" and explore some of its other capabilities. This presentation discusses some ways Rho, Inc. has ventured beyond the user interface and has effectively utilized OpenCDISC. Some of the topics require a rudimentary knowledge of XML. Accordingly, we will briefly discuss the structure of the standards' configuration files and some tools that work well with these XML files.

INTRODUCTION

The migration from paper-based to electronic submissions to the FDA revolutionized the review process. The anticipated improvements in data quality and review time were affected, however, by a lack of data standardizatio. A new study meant a reviewer had to become familiar with dataset contents, naming conventions, and data types – all of which are essential for a thorough understanding of the study, all of which take time.

The introduction of CDISC data models – SDTM and, later, others such as ADaM and SEND – became a viable, though not perfect, solution to the lack of standardization. The FDA now accepts clinical and analysis data that conform to the CDISC models. This, in turn, has led to these models becoming the *de facto* standard for organizations dealing with the FDA.

The standards hold great promise. Similar data structures mean generalized tools can be built to construct and review model-compliant data. Also, studies using the models can be used in meta analyses. There is, though, a cost that comes with standards compliance. Workflow, in particular, data validation, becomes more complex.

Validation? Haven't datasets submitted to the FDA *always* been validated? Of course, but compliance with the CDISC models adds a new dimension to the validation process. In the past, pre-standards, the validation process was a matter of ensuring that the specification for creating a variable was followed. Now the validation has to ensure that the data was created correctly *and* that it complies with the appropriate CDISC standard. Dataset and variable names, attributes, and other metadata are among the items that need to be compliant.

The up side to the greater validation burden is that third-party software can be created to complement or replace in-house tools. One such tool was WebSDM, used since the mid-2000's by the FDA. WebSDM was a somewhat closed, non-extensible system, and its list of validation checks was not widely circulated. The alternative to WebSDM was for pharma companies and CROs to write their own validation suites. This was an arduous process, and one that became increasingly untenable as new standards were introduced and existing ones were revised.

Clearly, there was a gap to be filled. OpenCDISC and the SAS® Clinical Data Toolkit are but two of the solutions. The OpenCDISC Validator has become popular both within the FDA and the health sciences community. This paper focuses on several underexplored uses of the software. The fact that it doesn't discuss the Clinical Data Toolkit or products from other companies should in no way suggest that these applications are less capable than OpenCDISC.

OPENCDISC

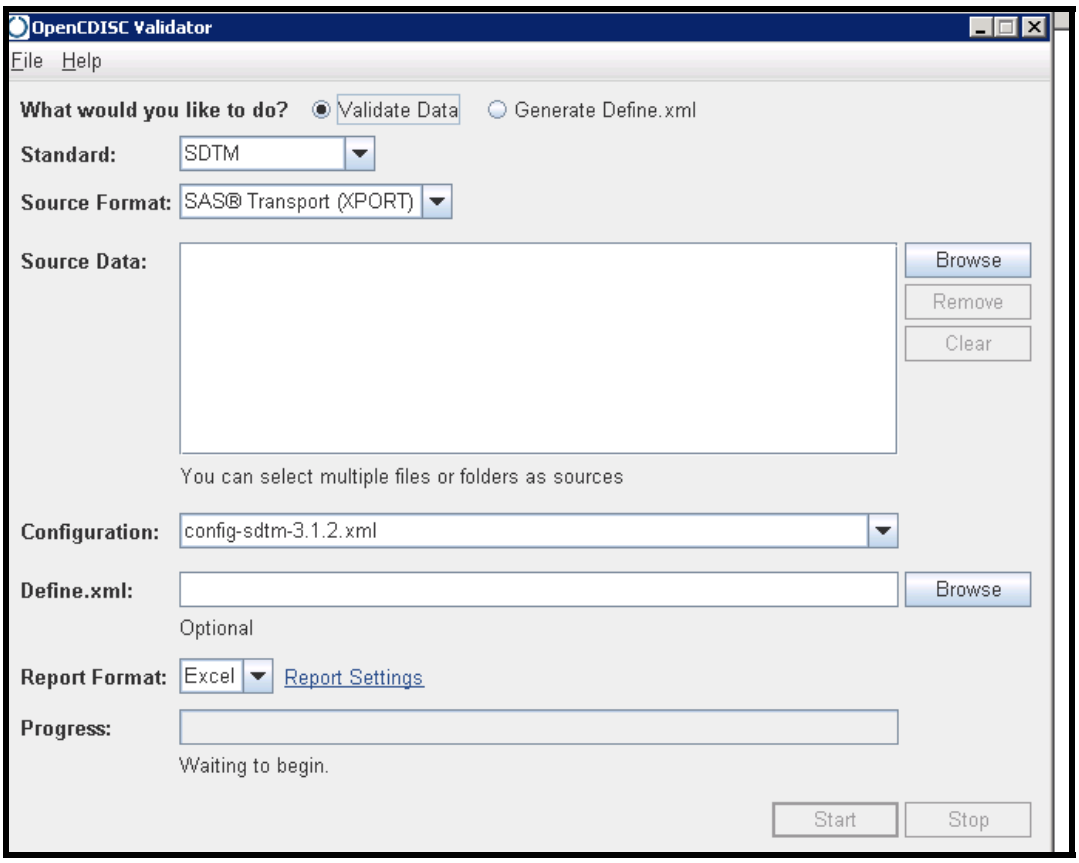
OpenCDISC.org was founded in 2008, its objective being an open source, collaborative environment for anyone working with CDISC standards. The first released product was the Validator, which ran checks exclusively on SDTM datasets. Subsequent versions have improved performance, added data models (SEND and ADaM), and have modified and added numerous checks. OpenCDISC Enterprise, a more full-featured, for-pay version of the software, was introduced in 2012.

It is easy to see why the Validator has become popular within the pharma community: it is easy to use; its configuration files can be repurposed for tasks other than validation, and the original "Community" version is free. Most important, it has been embraced by the FDA: when looking for validation rules in the FDA web site, you're directed from FDA web site (http://www.fda.gov/forindustry/datastandards/studydatastandards/default.htm) to OpenCDISC.org. Repeat after me: "if the FDA likes it, we like it."

THE VALIDATOR INTERFACE

Let's look at the typical way to run the Validator, via the GUI (shown in **Figure 1**, below). Usage is straightforward: specify the standard, the datasets to be validated, the configuration (validation rules) file, and other options that control the type and amount of diagnostic output. Once the options are selected, click "Start" to begin the Java-based processing.

Figure 1: Validator Interface



The GUI is simple to use and gives the user a good deal of flexibility. Repeated usage, however, surfaces some issues:

- **Location:** The BAT file that runs the Validator is located in the OpenCDISC root directory. Unless you copy the file to each study directory or create a desktop shortcut, you have to navigate to the root directory each time you use the program.
- **Persistence:** Selection of model, directory, etc. are not saved once you exit the GUI.
- **Checks:** This is not a GUI issue *per se*, but is important nonetheless. The standard, out-of-the-box checks are thorough, but may require modification (some might need to be turned off, study requirements may raise the need for additional checks, etc.)
- **Output:** The diagnostic output, typically an XLS file, is written to the same directory (`\reports`, below the OpenCDISC root directory) regardless of study.
- **Resource consumption:** The Validator runs in the foreground. This can degrade other applications' performance.

With the above in mind, we can develop a wish list of another way to run the Validator:

- **Location:** Be able to run the application from any directory, thus removing the need for navigation to the OpenCDISC directory.
- **Persistence:** Have the ability to preserve settings (model, datasets, configuration file, etc.).
- **Customized checks:** Be able to define a subset of existing checks and/or write new ones.

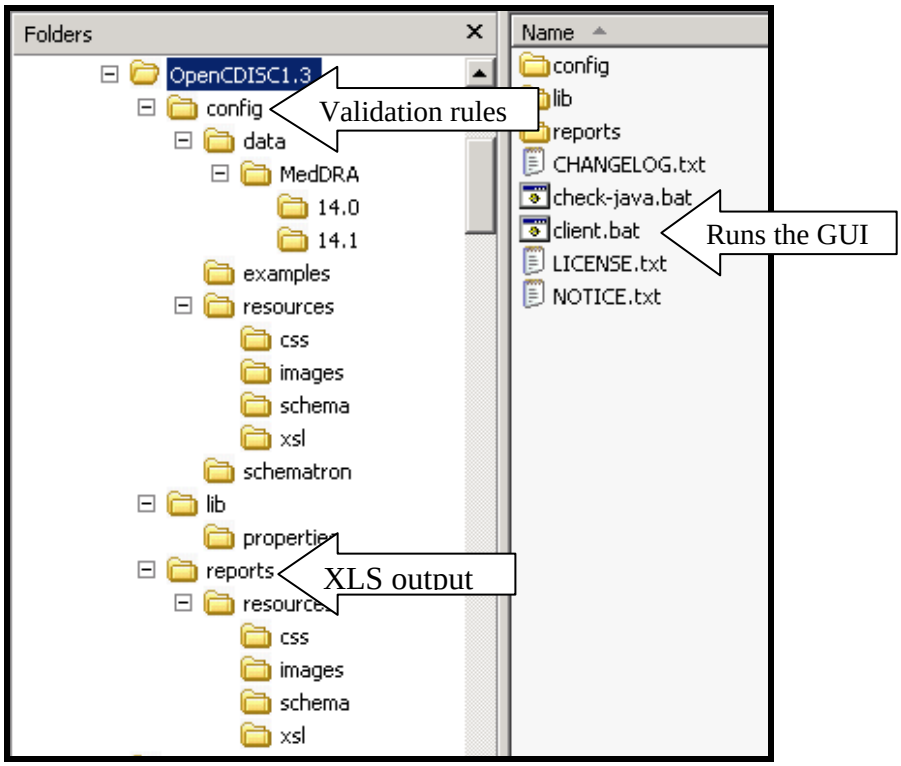
- **Output:** Be able to direct Validator output to a less-general, study-specific directory.
- **Performance:** Be able to batch/remote submit a program that runs the Validator.

All of these goals can be attained by looking at capabilities of the Validator that lie beyond the simple, traditional point-and-click usage. We need to know how to create a DOS BAT file to run the Validator and how to customize OCD's configuration files. Note that while the paper discusses the construction of a BAT file, the underlying concept – using a scripting language to execute the Validator – is applicable to *any* operating system.

TOPIC 1: CREATING VALIDATOR BAT FILES

Typically, to run OCD, you have to navigate to the OCD root directory, run the Validator client BAT file, enter model, etc. Then, when validation is complete, you have to navigate to the reports directory. These file and directory locations are identified in **Figure 2**, below:

Figure 2: Directory Structure



TOOLS

A fairly straightforward approach to extending the GUI's functionality is to create a program, in this case, a SAS macro, that will generate and run a BAT file that controls the Validator's input, processing, and output. In order to accomplish this, we need to know several things:

- Location of OpenCDISC directories
- Location of input (XPT) and output (typically, XLS) files
- Syntax of DOS BAT files (or the scripting language of the host operating system). In particular, we need to know the syntax required to navigate to a directory and to execute the Validator
- Validator Command Line Interface (CLI) parameters. These are shown in **Table** 1, below. (The table's contents were created by navigating to the OpenCDISC root directory, then to `\lib`, then entering
  `java -jar validator-cli-1.3.jar –help`

Table 1: Command Line Client Parameters

```
------------------------------------------------------------------
                  OpenCDISC Validator Command Line Client
 ----------------------------------------------------------------
 The following parameters may be passed to the Validator. Note that
 certain parameters may be required.
 For additional help, or to submit suggestions, please visit our
 community at http://www.opencdisc.org/
 General Parameters
    -task               Validate|Generate (Validate)
    -type               SDTM|Define|Custom (SDTM)
 Source Data Parameters
    -source              <path>
    -source:type         SAS|Delimited (SAS)
    -source:delimiter    <delimiter> (,)
    -source:qualifier    <qualifier> (")
Configuration Parameters
    -config              <path>
    -config:define       <path>
    -config:codelists    <path>
 Report Parameters
    -report              <path>
    -report:type         Excel|HTML|CSV|XML (Excel)
    -report:cutoff       <#> (1000)
    -report:overwrite    yes|no
 Generation Parameters
    -output              <path>
    -output:overwrite    yes|no
```

MACRO PARAMETERS

A robust design for a macro that builds the Validator BAT file requires a modest number of parameters. Those that correspond with the CLI include:

- **model:** Data model (ADaM or SDTM)
- **ver:** IG version of the model (for example, 3.1.2 if model is SDTM)
- **XPT:** Directory or LIBNAME containing the XPT files to validate
- **data:** Blank-delimited list of files in XPT directory to validate. Special handling, demonstrated in Example 2, below, if XPT identifies a LIBNAME.
- **useDefine:** Use define.xml to control Controlled Terminology and other checks?
- **msgLim:** Report message limit
- **outFile:** Name of report dataset
- **outDir:** Directory of output dataset
- **OCdir:** OpenCDISC Validator root directory

Other possible parameters:

- **dateTime:** Add date-time to the output file name?
- **runBat:** Run the BAT file?

MACRO DESIGN

The design of the macro is straightforward.

- Check for valid values of data model, add date-time, run BAT file and other parameters. Verify that the XPT and output directories exist. If define.xml is, verify it is present in the XPT directory. Write messages to the SAS Log and terminate if any test fails.
- Obtain the location of the OpenCDISC Validator directory. Verify that the configuration file for the data model and IG version (e.g., STDM 3.1.2) exists. Write messages to the SAS Log and terminate if any test fails.
- Write the BAT file to the same directory as the output XLS.
- Optionally, run the BAT file.

The purpose of this macro, like *any* macro, is to simplify the user's life. Yet we can readily see that specification of all parameters could become tedious. There are directories (input, output, Validator) and model-IG version to specify. Some of these can, of course, be handled by choosing sensible default values (`model=sdtm`, `ver=3.1.2`). Others, due to their non-generic, study-specific nature can't have defaults (`XPT`, `outDir`, and `Ocdir`).

This is where study and enterprise-level metadata come into play. If a parameter isn't specified and cannot have a default value, the macro can assign a value by referring to the appropriate metadata table. If, for example, `outDir` was not specified, the macro could attempt to identify the location by using study-level metadata. Similarly, if `OCdir` was null, the macro could use global metadata to locate the default Validator version's root directory.

Consider this call to the macro, which is preceded by a macro call that allocates libraries for a study:

```
%setup(study=j:\LargePharm\Ablixa)
%runOCval(model=sdtm, ver=3.1.3)
```

The macro uses a combination of user-specified values (`model` and `ver`), default values (`useDefine`, `outName`, *et al.*) and metadata-supplied values (for `XPT`, `outDir` and `OCdir`) to produce the BAT file.

Table 2, below, shows how the macro directly or indirectly uses the macro parameters and metadata when it builds the file (refer to items **displayed like this**).

Table 2: Parameter, Metadata Usage in the Generated BAT File

```
@echo off
rem Model [model]  IGVersion [ver]
rem Report parameters: Cutoff [msgLim]
rem Configuration file     [OCdir\config\config-model-ver.xml]
rem Input/XPT directory    [XPT]
rem Output XLS             [outDir\outFile.xls]
rem Output BAT             [outDir\outFile.bat]
OCdir:
cd OCdir\lib
java -jar validator-cli-OCver.jar -task="Validate" -type="model" ^
-source="XPT\*.xpt" ^
-config="OCdir\config\config-model-ver.xml" ^
-report="outDir\outFile.xls " ^
-report:overwrite="yes" -report:cutoff="msgLim"
```

EXAMPLE 1: VALIDATE ALL DATASETS

A complete program to create and run the BAT file is shown below. If the program is run in the foreground, SAS will execute, then terminate once the BAT file is built and begins execution. Since the macro specifies processes should run asynchronously, the DOS window running the Validator will remain active until validation is complete. If the program is run as a remote process, the SAS Log will be updated and the Validator process will run to completion without user notification. Note that regardless of how the program is run, the problem of lack of parameter persistence is solved: if the program is saved, then the parameters are, by definition, also saved. To save a different set of parameters, simply create another call to the macro or save the modified call as a new program. The output BAT file is displayed in **Table 3**, below.

```
%setup(study=j:\LargePharm\Ablixa)
%runOCval(model=sdtm, ver=3.1.3)
```

Table 3: Generated BAT File

```
@echo off
rem Model SDTM  IGVersion 3.1.2
rem Report parameters: Cutoff 100
rem Configuration file     [h:\OC\1.3.1\config\config-SDTM-3.1.2.xml]
rem Input/XPT directory    [j:\LargePharm\Ablixa\sub\data\sdtm]
rem Output XLS             [j:\LargePharm\Ablixa\prog\SDTM\val\OCVal.xls]
rem Output BAT             [j:\LargePharm\Ablixa\prog\SDTM\val\OCVal.bat]
h:
cd h:\OC\1.3.1\lib
java -jar validator-cli-1.3.1.jar -task="Validate" -type="SDTM" ^
-source="j:\clients\LargePharm\Ablixa\sub\data\sdtm\*.xpt" ^
-config="h:\OC\1.3.1\config\config-SDTM-3.1.2.xml" ^
-report=" j:\LargePharm\Ablixa\prog\SDTM\val.xls " ^
```

```
-report:overwrite="yes" -report:cutoff="100"
```

EXAMPLE 2: VALIDATE A SINGLE DATASET

This differs from the previous example only by its addition of the `data` and `xpt` parameters. While this may seem to be a somewhat trivial difference, its implications for workflow are significant. Since the `xpt` parameter identified a LIBNAME, the macro attempts to locate native SAS dataset `WORK.AE`. If the dataset exists, it is converted into an XPT file that can be processed by the Validator.

```
%inc (or similar) to allocate autocall library and study, global metadata
… code that produces dataset work.AE …
%runOCval(model=sdtm, ver=3.1.3, xpt=work, data=ae)
```

Being able to validate a single dataset as shown in this example means that it is a simple matter to validate the dataset as soon as it is created. Feedback about dataset quality is both immediate and easy to obtain.

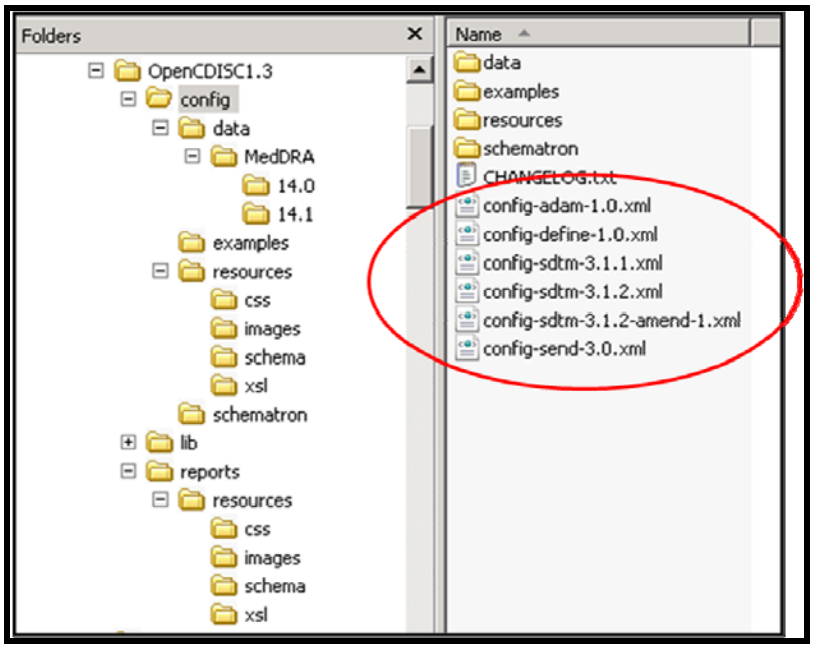
TOPIC 2: CUSTOMIZED CHECKS

In the previous section we learned how to bypass the Validator interface and easily specify the CDISC data model to be validated, where to look for input and output, and other features. Being able to simplify the validation process is fine, but the question remains: just what, exactly, is being checked? What are the validation rules, and how can we modify them? Not surprisingly, these questions are answered in this section.

WHAT ARE THE RULES?

Recall the OpenCDISC Validator directory structure, depicted in **Figure 2**. The `config` directory, one level below the Validator root directory, houses a set of files using the naming convention `config-`*`model-Igversion`*`.xml`. These are Object Document Model (ODM)-compliant files with a vendor extension defining the validation rules. The out-of-the-box directory contents for Version 1.3.1 of the Validator is shown in **Figure 3**, below.

Figure 3: Configuration Directory

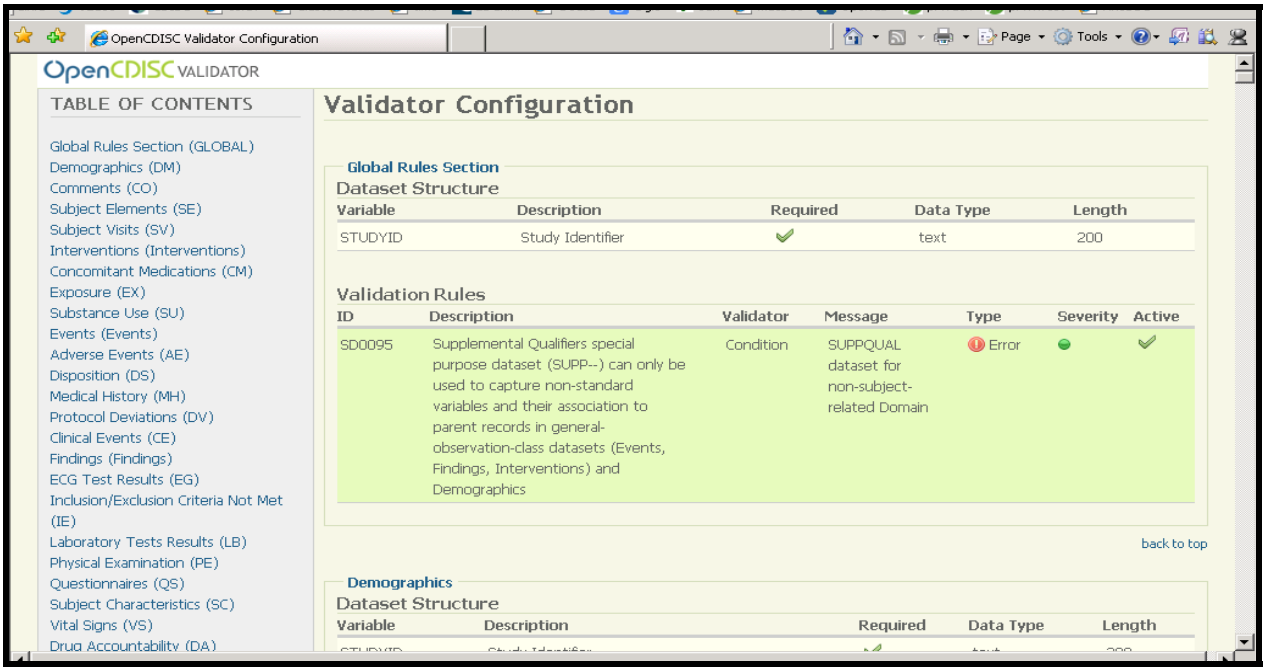


Each file contains this line near the top, prior to the ODM tag:

```
<?xml-stylesheet type="text/xsl" href="resources/xsl/config.xsl"?>
```

This tag says, in effect "when the XML file is opened, don't display the raw XML. Instead, transform it using the XSL [**Ex**tensible **S**tylesheet **L**anguage] file found in the `resources\xsl` directory." The XSL transforms the XML into a well-formatted (syntactically and aesethically) HTML page. The transformed SDTM IG version 3.1.2 XML file is shown below, in **Figure 4**.

Figure 4: Rules XML Transformed Into HTML



Recall the earlier comment that when you are in the FDA web site looking for validation rules, you are given a link to the OpenCDISC web site. The configuration/XML files shown above are what the FDA is using as the rules for valid CDISC data models. It is important to remember that the XML is not only a reference for *you*, it is also a reference for the *FDA reviewer*.

WHY WOULD WE CHANGE THE RULES?

Given that the XML files constitute what might be considered the Gold Standard for each model-IG version, why would we want to change the rules? There are several valid (no pun intended) reasons for doing this:

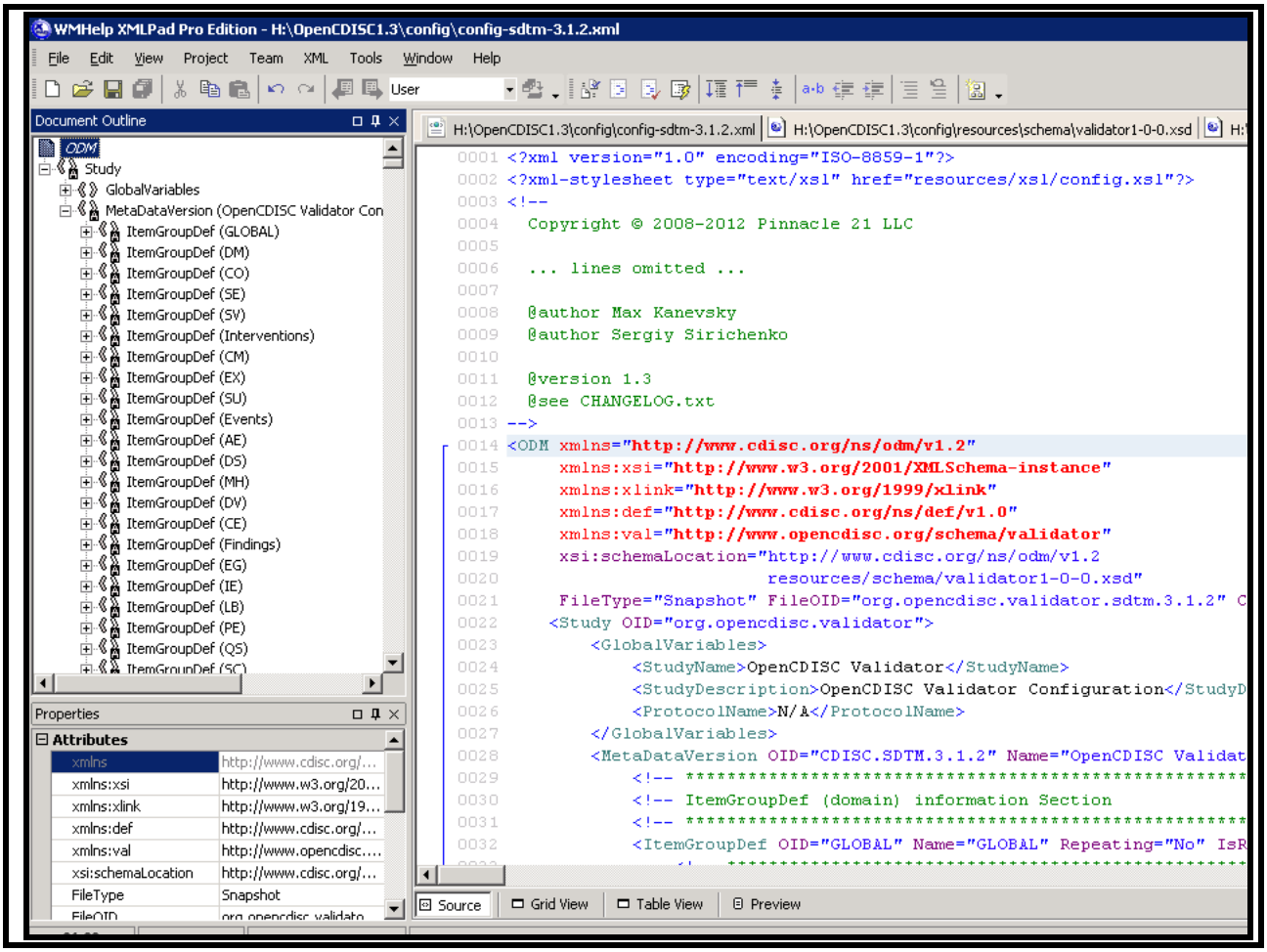
- A study may have datasets and/or variables that require scrutiny that goes beyond what the validation rules provide. One alternative is to write a SAS program to perform the extra validation. The other option is to work within the OpenCDISC validation framework and write a custom check.
- You may be running the Validator at a point in your workflow where certain checks are not appropriate and should be turned off, rather than remain active and possibly produce false positives.
- You may want to turn off an OpenCDISC check may have known issues (`xxTPT` variables in SDTM 3.1.2, for example, could legitimately have values such as "`VISIT 2`" – OpenCDISC 1.3, however, looked only for ISO 8601-compliant values and would flag "`VISIT 2`" as an error).

Some knowledge of an ODM-compliant XML file structure and of the Validator extensions is required in order to get a reasonable comfort level.

XML, ODM, AND "NODES TO KNOW"

As noted earlier, the rules for validation are contained in model version-specific XML files that conform to the ODM schema. Either of the methods for altering the rules requires a rudimentary knowledge of the structure of the XML files. **Figure 5**, below shows the SDTM 3.1.2 file opened using XMLPad, an open-source XML editor.

Figure 5: Config File Opened using XMLPad

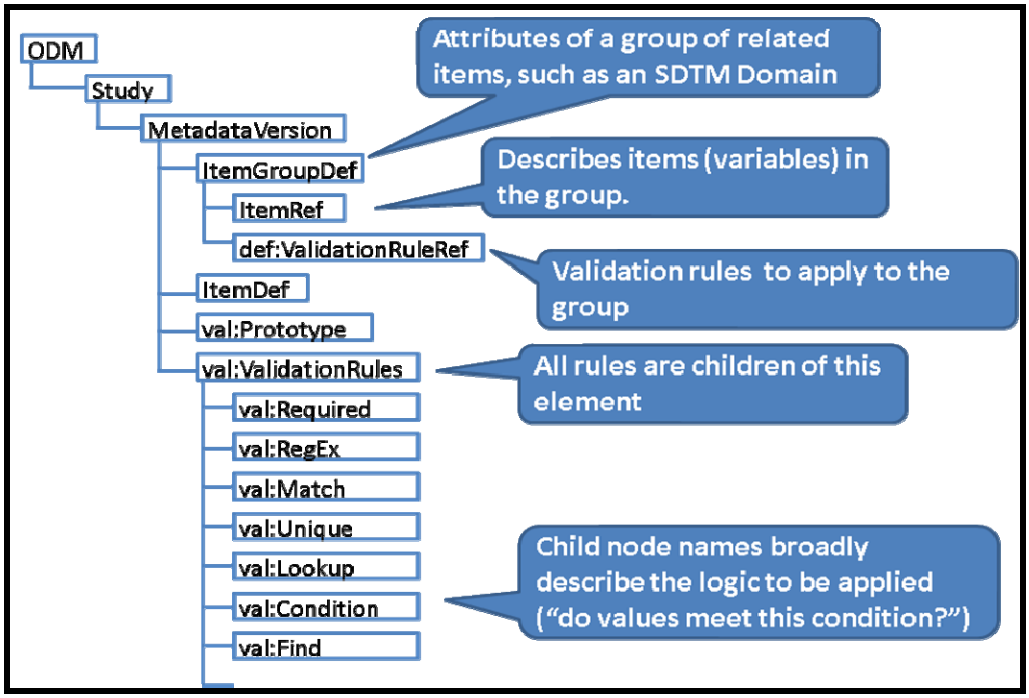


Although it initially appears a bit daunting, after a bit of examination there's a reassuring consistency and predictability to both the XML and the way it's displayed:

- The file is a hierarchy of nodes structured according to the ODM schema. The schema (not shown here and beyond the scope of this paper) controls the naming, order, and cardinality of elements. It also identifies permissible values of various element attributes (e.g., `ORIGIN`, `DATATYPE`) and allows for inclusion of third-party (aka "vendor") extensions to the schema (in the context of this paper, the Validator's rules)
- The file contains descriptions of item groups (datasets), individual items (variables), and references to and definitions of validation rules.

The full ODM specification has hundreds of elements and attributes. **Figure 6**, next page, depicts nodes that are relevant to validation as implemented by the OpenCDISC Validator.

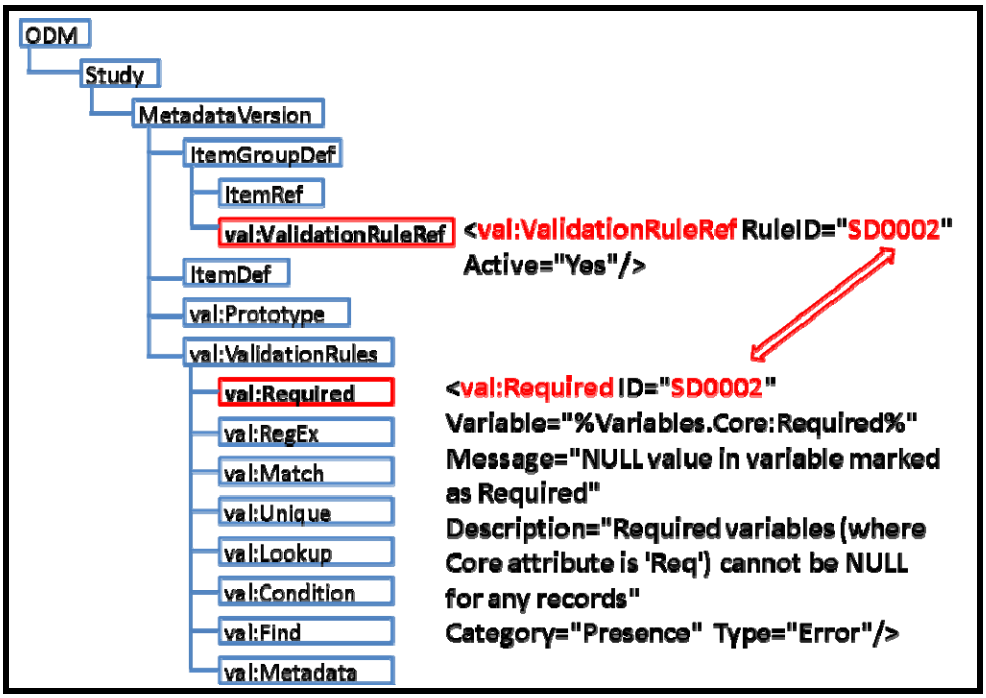
Figure 6: Validation-Related Nodes



In English, the Figure can be loosely interpreted as "there will be a set of validation rules (`def:Validation-RuleDef`) after the specification of each dataset (`itemGroupDef`) and its variables (itemDef). After all datasets (`itemGroupDef`) have been defined, there are elements (`def:ValidationRules`) that define what, exactly, is evaluated for each type (`val:Required`, `val:RegEx`, *et al.*) of test."

To understand the linkage between a *reference* to a rule and the *definition* of a rule, refer to **Figure 7**:

Figure 7: Linking Rule References and Definition



There can be any number of references to a specific rule. If more than one dataset uses the rule, a reference to it can appear in each dataset's `itemGroupDef`:

```
<val:ValidationRuleRef RuleID="SD0002" Active="Yes"/>
```

There is a *single* definition of the rule. Since the example we have chosen it involves identifying a Required variable, the test fits the "Required" rule type. The XML tag begins with `val:Required`, and contains attributes (`Variable`, `Message`, `Description`, `Type`) that are defined by the Validator schema.

```
<val:Required ID="SD0002" Variable="%Variables.Core:Required%" Message="NULL value
in variable marked as Required"  Description="Required variables (where Core
attribute is 'Req') cannot be NULL for any records"  Category="Presence"
Type="Error"/>
```

With this highly simplified interpretation in mind, let's look at two ways to modify the XML.

APPROACH 1: ALTERING EXISTING RULES

The simplest, and most limiting, method is to simply turn off a rule. Suppose we have review the transformed HTML in **Figure 4** and have decided to turn off RuleID SD0002. This is a straightforward process:

- Using an XML or plain text editor, locate all references to RuleID SD0002 (search for `RuleID="SD0002"`). Change the `Active` attribute from "`Yes`" to "`No`"
- Save the XML to a *different* file in the `\config` directory, preserving the model as the second hyphen-delimited value (this is necessary for the Validator interface to locate config files appropriate to the odel). For example: `config-sdtm-3.1.1-cust.xml`

The next time the Validator is run for SDTM, the new XML file will appear in the "Configuration" selection list, as shown in **Figure 8.**

Figure 8: GUI Picks Up New Configuration File

APPROACH 2: CREATING NEW RULES

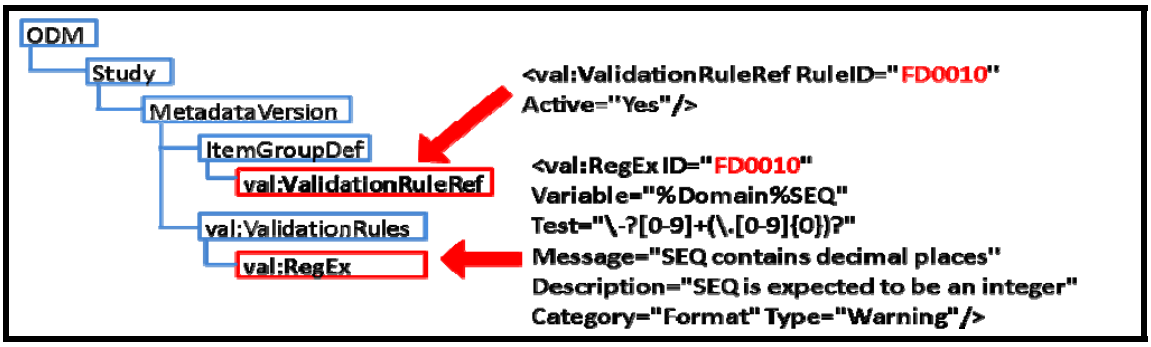
Creating a new validation rule requires a deeper knowledge of the Validator's capabilities. As **Figure 7** implied, there are a variety of rule types. The new rule should match the Validator rule category. If the new rule requires that a variable have a unique value, the rule would be written using attributes defined for the "Unique" rule. A complete enumeration of the rule types and syntax is beyond the scope of this paper and can be found on the OpenCDISC web site (www.OpenCDISC.org).

For the sake of illustration, let's assume that we want to add a rule that an SDTM sequence variable (*domain*`SEQ`) should be an integer. The steps are:

- Determine the rule type. The validation test can be stated as a Regular Expression, so `type=RegEx`
- Using the documentation from the OpenCDISC web site, determine what attributes are required for the test. Among these is `RuleID`, which should be unique and not conflict with *any* existing rule type's RuleID.
- Identify variables that will be evaluated with the new test.
- Create a copy of the SDTM configuration file that will be modified.
- Edit the file created above
  - Add `val:ValidationRuleRef` elements to each dataset using the new rule
  - Add `val:RegEx` as an element below (a "child" element of) `val:ValidationRules`
    - Save the XML.

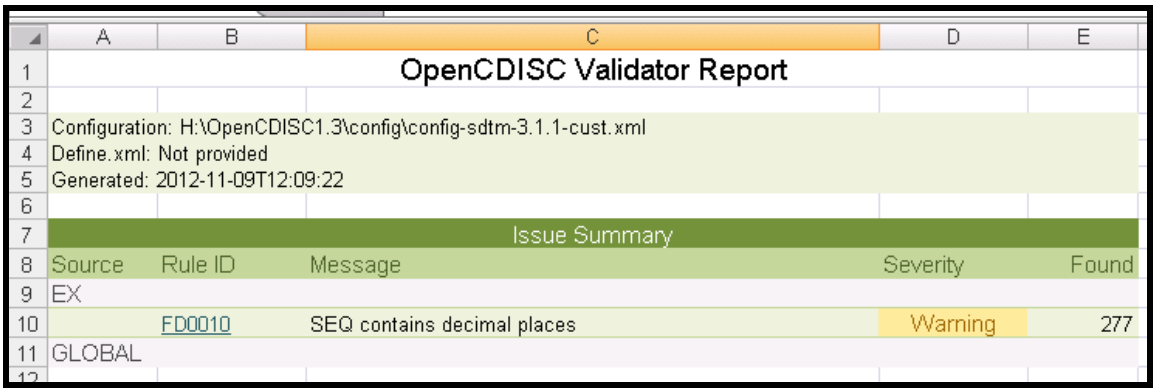
Seen graphically, in **Figure 9,**below, the linkage of the new rule to the rule definition resembles that of **Figure 7**.

Figure 9: Custom Rule Reference, Definition



As in Approach 1, the next time the Validator is run for SDTM, the new XML file will appear in the "Configuration" selection list. The results file does not distinguish between standard and custom checks; If the validation criteria of the new test is not met, it will be reported in the same manner as any other test in the configuration file, as shown in **Figure 10**, below.

Figure 10: Validation Report Showing Custom Check



	A	B	C	D	E
1			OpenCDISC Validator Report		
2					
3	Configuration: H:\OpenCDISC1.3\config\config-sdtm-3.1.1-cust.xml				
4	Define.xml: Not provided				
5	Generated: 2012-11-09T12:09:22				
6					
7			Issue Summary		
8	Source	Rule ID	Message	Severity	Found
9	EX				
10		FD0010	SEQ contains decimal places	Warning	277
11	GLOBAL				

CUSTOM CHECKS AND THE BAT FILE MACRO

Recall that the macro discussed in the previous section had parameters which paralleled those of the Validator CLI. When building the `config` parameter in the BAT file, we used the macro's `model` and `ver` parameters. If custom check files are created, they need to be accessible to the macro. This could be easily implemented by adding a parameter to the macro that contains the custom part of the file name, performing a test to ensure the configuration file exists, and adding it to the `–config` parameter in the BAT file:

```
%runOCval(model=sdtm, ver=3.1.3, cust=seqCHK)
```

The macro inserts the custom configuration file suffix in the BAT file as follows:

```
-config="h:\OC\1.3.1\config\config-SDTM-3.1.2-seqCHK.xml" ^
```

CONCLUSION

This paper has described ways to extend the OpenCDISC Validator framework beyond the typical point-and-click usage. Some topics not discussed (such as how to repurpose the configuration files and the workflow required to move to a new version of the Validator) are "ideas for another day." It's worth noting that what we discussed – the BAT file generator and customized checks – require a knowledge of tools old (OS scripts such as BAT files) and new (XML, XML schema, regular expressions and the OpenCDISC API).

If you're not familiar with these topics, yourlearning curve will be unfavorably shaped. The time spent acquiring the requisite skills is, however, time well spent. It will enable you to confidently navigate the XML rules files and extend the Validator's capabilities. Even more important is the expansion of your skills into the XML technologies that are becoming increasingly prevalent in the industry. You could code the entire validation suite in SAS, but that would be

thinking “inside the box.” Although the SAS “box” is a large and comfortable place, in the long run you do yourself and your skill set a disservice by ignoring new technologies that are rapidly changing the industry landscape.

ACKNOWLEDGMENTS AND DISCLAIMER

This paper was written in the author’s capacity as a satisfied user of OpenCDISC and in recognition of the benefits and principles of open source software. The author has no personal, financial interest in OpenCDISC or Pinnacle 21.

CONTACT INFORMATION

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

Name:	Frank DiIorio
Enterprise:	CodeCrafters, Inc.
Work Phone:	919-602-8421
E-mail:	Frank@CodeCraftersInc.com
Web:	www.CodeCraftersInc.com