

## OpenCDISC.org – an open source initiative delivering tools for validation of CDISC data

Max Kanevsky, Pinnacle 21, Plymouth Meeting, USA  
Niels Both, S-cubed, Copenhagen, Denmark  
Tim Stone, Pinnacle 21, Plymouth Meeting, USA

### ABSTRACT

For several years the FDA has requested that sponsors submit clinical trial data in CDISC SDTM format. Recently, the FDA has begun requiring submissions of other forms of data to be in standardized CDISC formats, for example, CDISC ADaM for analysis data, and CDISC SEND for animal toxicology studies. While commercial tools have been available to validate CDISC SDTM data since the beginning of this standards-adoption process, they have not been freely available to everybody. OpenCDISC.org was initiated in 2008 as an open source community dedicated to the creation of extensible tools for the implementation of CDISC standards, with an initial focus on data validation. The OpenCDISC Validator is built using Java and is configurable so that it can check not only various versions of CDISC data, but can also be tailored to meet the specific needs of a particular submission or a particular setting within a pharmaceutical company. The tool is configured using XML files, which can be modified or created using a variety of applications, including traditional base SAS(R).

This paper will present

- the architecture of OpenCDISC Validator and its XML configuration mechanism
- examples of how SAS can be utilized to configure the tool
- the disparate ways that CDISC SDTM data is currently validated
- how Validator might utilize the CDISC SHARE repository in the future

### INTRODUCTION

Since 2004 the guidance from the American Food and Drug Administration (FDA) has been to submit clinical data (Data Tabulations) in an industry-wide standard format, namely CDISC SDTM [1]. The FDA reviewers have used WebSDM as a tool to view and validate the received data. The validation output from the tool was used as feedback from FDA to the submitter. The checks used in this validation process were not defined by the CDISC organization, but by the WebSDM vendor in a discretionary fashion, based on both CDISC SDTM and some requirements specific to the WebSDM tool. Reviewers were limited to the list of checks pre-specified by the software, making updates reflecting best-practices or advances in the standard dependent on the vendor's development timeline. Additionally, this list of validation checks was only made publically available online a few years later [2].

Since that point in time, pharmaceutical companies have developed custom SDTM validation SAS macros, inspired by the same list of validation checks found in WebSDM. This led to disparate ways of validating CDISC SDTM datasets, as no standard list of validation checks for SDTM was defined by CDISC.

The lack of open and freely available SDTM validation tools was becoming a limiting factor to CDISC adoption, especially in organizations without the IT resources necessary to develop custom solutions using a language such as SAS. In 2008, OpenCDISC.org was launched as an open source community dedicated to building extensible tools and frameworks for the implementation of CDISC standards. The objective of OpenCDISC.org is to foster a collaborative environment where developers, data managers, statisticians, and regulators can jointly discuss, identify, and develop tools. OpenCDISC Validator is the first product of this collaboration providing a free and configurable CDISC Validator. The Validator has initially focused on CDISC SDTM, but has since expanded and matured to support other CDISC and custom data formats.

This paper describes the architecture and configurability of the OpenCDISC Validator and how SAS can be utilized to configure it. It also describes how in the future the Validator might use the CDISC SHARE repository as the source for validation rule definitions.

## OPENCDISC VALIDATOR ARCHITECTURE

The key architectural concept of OpenCDISC Validator is to decouple the definition of validation rules from application logic. This provides ultimate flexibility to create and maintain any number of validation rule definitions necessary to meet the diverse needs of sponsors, CROs, laboratories, and anyone else involved in collection, storage, and exchange of clinical data. The OpenCDISC Validator's architecture (Figure 1) is comprised of the following components:

- Configuration – an XML document defines the validation rules to be executed against a set of datasets. The rules are expressed according to the OpenCDISC Validation Framework, which is described in the next section. The Validator could have any number of configurations to support different standards (SDTM, ADaM, SEND, etc) or different standard versions (SDTM v3.1.1 vs. SDTM v3.1.2).
- Validation Engine – the core component of the architecture is developed in Java and houses the application logic, which reads and parses input datasets, interprets and executes validation rules described in the configuration file, and outputs a validation report. The current Validator release (v1.1) supports SAS XPORT and delimited text files as input.
- Validation Report – the results of validation are outputted as XML and include the error/warning messages, descriptions, and details of data records that failed validation. The Validator renders the report in Excel, CSV, or HTML based on user preferences.
- Interfaces – the Validator provides three interfaces: desktop client (GUI), command line (CLI), and application programming (API).

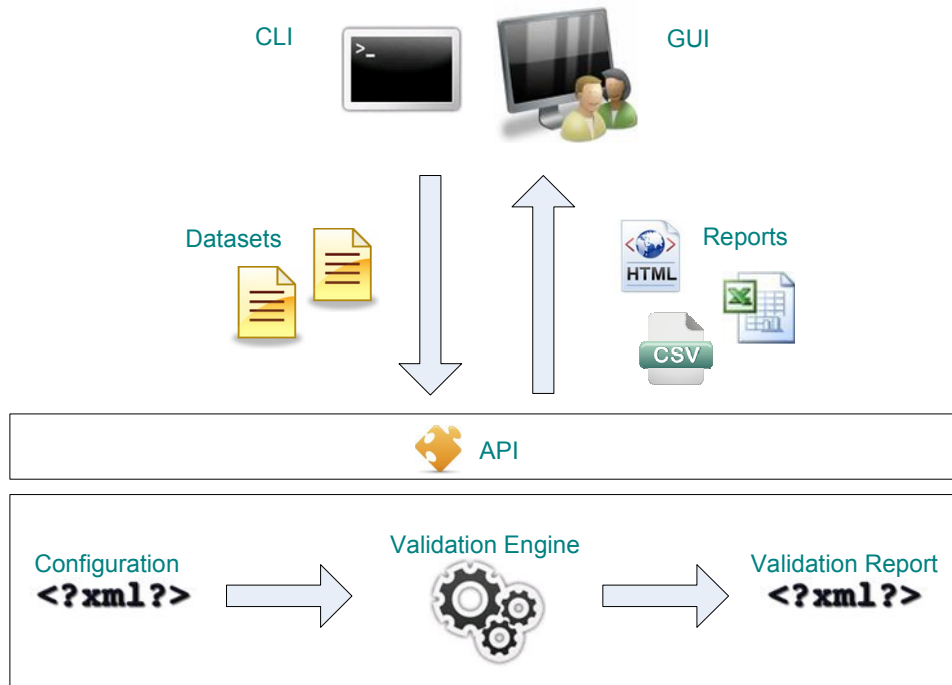


Figure 1: OpenCDISC Validator Architecture

## OPENCDISC VALIDATION FRAMEWORK

Validation rules are defined using an XML-based OpenCDISC Validation Framework [3] that is designed to be simple, yet powerful. Designed as an extension to the define.xml specification [4], it allows users to combine the metadata included in their study definition with a rich set of validation rules. This combination enables the Validator to accomplish a number of goals:

- Validate any CDISC compliant dataset, including SDTM, define.xml, ADaM, and SEND
- Support near-SDTM, SDTM+, and custom datasets
- Provide a portable format for sharing data validation rules between partners

The following sections describe in detail the structure of OpenCDISC configuration files and show how to use the OpenCDISC Validation Framework to define validation rules to meet the needs of your organization.

# PhUSE 2010

## CONFIGURATION FILE OVERVIEW

The Validator configuration file is an extension of define.xml specification, thus it utilizes the standard `ItemGroupDef`, `ItemRef`, and `ItemDef` tags to describe the structure of datasets including variable names, labels, and data formats. Here is an example of each tag as they might appear in a Validator configuration file:

```
<ItemGroupDef OID="DM" Name="DM" Repeating="No" IsReferenceData="No"
  Purpose="Tabulation" def:Label="Demographics"
  def:Structure="One record per subject"
  def:DomainKeys="STUDYID, USUBJID"
  def:Class="Special Purpose" def:ArchiveLocationID="Location.DM">

<ItemRef ItemOID="DM.AGE" OrderNumber="11" Mandatory="No"
  Role="Record Qualifier" val:Core="Expected"/>

<ItemDef OID="DM.AGE" Name="AGE" DataType="integer" Length="8" def:Label="Age"/>
```

As part of the Validator configuration extension, the `ItemRef` element can also contain the `val:Core` attribute, which takes the values `Permissible`, `Expected`, and `Required`, providing finer control over how variables should be validated based on the three levels defined in the SDTM implementation guide.

The most important extension is the `val:ValidationRules` section that is located under `MetaDataVersion` and contains all of the validation rule definitions. All rules must be placed inside this section and are referenced in any of the `ItemGroupDefs` using the `val:ValidationRuleRef` tag as follows:

```
<val:ValidationRuleRef RuleID="SD0006" Active="Yes"/>
```

## VALIDATION RULES

The Validation Rule is the cornerstone of the OpenCDISC Validator's capabilities. To allow for a rich set of validations, there are several different XML tags that are used to invoke particular checks by the validation engine. When defining a Validation Rule, the configuration author may choose from one of the following types:

- Match Rule
- Unique Rule
- Regular Expression Rule
- Conditional Rule
- Required-When Rule
- Lookup Rule
- Metadata Rule
- Find Rule

While each type has its own unique set of Attributes—data within the XML tag that provides information about the rule—and distinct tag name, all rules share these common attributes that help with report generation and general validation:

- `ID` – specifies a unique rule ID, which allows the rule to be referenced in other areas of the configuration and identifies the rule in the report output.
- `Variable` – indicates variables involved in the rule.
- `Message` – a short sentence indicating the cause of the failure.
- `Description` – a more detailed explanation of what was wrong with data that caused this rule to fail.
- `Category` – groups similar rules into meaningful categories, for example `Consistency`, `Format`, or `Metadata`.
- `Type` – indicates the type of issue that will be generated if data fails the rule. Acceptable values include `Information`, `Warning`, and `Error`.
- `Severity` – the rule's relative importance. Acceptable values include `Low`, `Medium`, and `High`.

Combining all of these attributes, we get the basic form of a Validation Rule XML tag:

```
<val:RuleTagName ID="id" Variable="variable" Rule-Specific Attributes
  Message="message" Description="description"
  Type="type" Severity="severity" Warn="warn"/>
```

## PhUSE 2010

Now we'll take a look at how to work with the various specific rule types.

### Match Rule

The Match Rule allows a variable's data to be checked against a set of acceptable terms defined within the rule. It introduces the `Terms` and `Delimiter` attributes which provide the list of acceptable variable values and the character used to separate them in that list, respectively. The default value for `Delimiter` is a comma (,) if one is not specified. The `Terms` attribute is required.

The following is an example of a Match rule that checks the contents of --STRTPT variable against BEFORE, COINCIDENT, AFTER, or U acceptable terms:

```
<val:Match ID="CT0056" Variable="%Domain%STRTPT" Terms="BEFORE, COINCIDENT, AFTER, U"
  Delimiter="," Message="Invalid %Domain%STRTPT value" Description="Start Relative
  to Reference Time Point should be populated with 'BEFORE', 'COINCIDENT',
  'AFTER', or 'U'" Category="Terminology" Type="Warning" Severity="Medium"/>
```

Note the presence of the `%Domain%` string, which acts as a placeholder for the current domain abbreviation. So when this particular rule is executed against the Exposure domain the `Variable` attribute is automatically set to EXSTRTPT by the validation engine.

### Unique Rule

The Unique Rule checks for the uniqueness of a given variable's value across all records in the current data source, optionally grouping by one or more other variable values. For example, the CDISC specification states that the Sequence variable must be unique for a given Subject ID. In this case, the variable analyzed for uniqueness would be the Sequence, grouped by the Subject ID (since different subjects may have the same Sequence number). This is done with the addition of the `GroupBy` attribute, which takes a comma-separated list of variable names on which to compare the uniqueness of the data in the variable specified in the `Variable` attribute.

The validation rule that checks the uniqueness of Sequence variable is defined as follows:

```
<val:Unique ID="SD0005" Variable="%Domain%SEQ" GroupBy="USUBJID"
  When="%Domain%SEQ != ''" Message="Non-unique value for SEQ"
  Description="Identifies records where non-unique values for Sequence Number
  variable exist within a subject" Category="Consistency" Type="Error"
  Severity="High"/>
```

### Regular Expression Rule

The Regular Expression, or Regex, Rule allows a variable's value to be validated against a pre-defined pattern. For example, you are able to enforce the length of a variable's value, and specify that it must be alpha-numeric, etc. The Regular Expression Rule introduces the `Test` attribute, which contains the regular expression string used to validate the data. If the data does not match the regular expression, then the record will fail that Validation Rule.

The following Regex rule checks that TESTCD variable is under 8 characters, does not start with a number, and only contains alphanumeric characters:

```
<val:Regex ID="SD0018" Variable="%Domain%TESTCD" Test="[A-Za-z][A-Za-z0-9_]{0,7}"
  Message="Invalid value for --TESTCD variable" Description="The value of Short
  Name of Measurement, Test or Examination (--TESTCD) must be limited to 8
  characters, cannot start with a number, and cannot contain characters other than
  letters, numbers, or underscores." Category="Format" Type="Warning"
  Severity="Low"/>
```

### Conditional Rule

The Conditional Rule allows comparisons to be made between variable values in the current record, and optionally only if a particular precondition is satisfied. In more general terms, using this rule we can make sure one or more variables have their expected values if another set of variable values indicates that they should. This is accomplished using an expression language to establish the variable relationships.

An expression consists of one or more variables, and conditions that the value of those variables must meet. These conditions may be grouped together using parenthesis, and are linked logically by either `@and` or `@or`. When we use `@and`, we're saying that the condition on both the left and the right side of the `@and` must be true. When we use `@or`, only one of the sides must be true. When defining the expressions, we are always writing what must be true for the

## PhUSE 2010

rule to succeed. If the conditions are not met, then the current record will fail for that rule, and a message will be printed to the report.

To make a condition, you use a variable name, an operator, and either another variable name or a constant value. This takes the following form:

```
variable operator variable/value
```

A value may be a word, which must be surrounded by single quotation marks, or a number, which may just be written by itself. Additionally, to specify a null, or blank, value, an empty set of single quotation marks ("") can be used. If two variables are used, their values are compared against one another. This comparison is done using the operator, which may be one of the following:

==	equal to; the value on the left must match the value on the right for the condition to be true
!=	not equal to; the value on the left must not match the value on the right for the condition to be true
@eqic	equal to, ignore case; the value on the left must be the same text as the value on the right for the condition to be true, and the case of each letter in that text is not important
@neqic	not equal to, ignore case; the value on the left must not be the same text as the value on the right for the condition to be true, and the case of each letter in that text is not important
@gt	greater than; the value on the left must be greater than the value on the right for the condition to be true
@gteq	greater than or equal to; the value on the left must be greater than or equal to the value on the right for the condition to be true
@lt	less than; the value on the left must be less than the value on the right for the condition to be true
@lteq	less than or equal to; the value on the left must be less than or equal to the value on the right for the condition to be true
@re	matches the regular expression; the value on the left must match the regular expression pattern given on the right for the condition to be true
!( )	negation; generates the opposite result (true » false, false » true) of the combined result of the conditions contained in the parenthesis

**Table 1: Conditional Rule Operators**

The expressions are applied using the `Test` and `When` attributes that the Conditional Rule introduces. Both attributes accept an expression string as their value, but only the `Test` attribute is required. If the `When` attribute is used, that expression will be evaluated first. If the result is true, then the `Test` expression will be evaluated and its result will determine if the current record passes or fails the Validation Rule. If the `When` expression evaluates to false, then the record passes. Likewise, if the `When` attribute is not used, the result of the `Test` expression will determine the record's validity.

The following Conditional Rule validates that Start Date must be less than or equal to End Date:

```
<val:Condition ID="SD0013" Test="%Domain%STDTC @lteq %Domain%ENDTC"
  When="%Domain%STDTC != '' @and %Domain%ENDTC != ''" Message="Begin day must be
  less than or equal to end day" Description="Identifies records that violate the
  condition [(Start Date/Time of Observation less than or equal to End Date/Time
  of Observation)], limited to records where [Start Date/Time of Observation is
  not null and End Date/Time of Observation is not null]" Category="Limit"
  Type="Error" Severity="High"/>
```

### Required-When Rule

The Required-When Rule ensures that a variable has a value when a specific condition is met. The condition is specified in the `When` attribute, which uses the same expression language as described in the Conditional Rule above.

The following Required-When Rule validates that Original Units are required when Result or Finding in Original Units are provided:

```
<val:Required ID="SD0026" Variable="%Domain%ORRESU"
  When="%Domain%ORRES != '' @and %Domain%ORRES @re '[-+]?[0-9]*\.\?[0-9]+' "
  Message="Missing units on value" Description="Identifies records that violate
  the condition [Original Units is not null], limited to records where [Result or
```

## PhUSE 2010

```
Finding in Original Units is not null]" Category="Consistency" Type="Warning"
Severity="Medium"/>
```

Also, note the regular expression used in the `When` attribute that ensures the rule is only executed when the value of `Result` or `Finding in Original Units` is numeric.

### Lookup Rule

The Lookup Rule allows values from particular variables to be validated against a set of values from an alternative source. This source may be another data source in the validation set, or an external file. The rule tag is setup slightly differently for each of these kinds of sources, so it's best to go over them individually.

### Cross-Dataset Lookup:

The most direct of the available Lookups, the cross-dataset lookup compares records across multiple files in your source list. For instance, the CDISC specification contains rules where records in one Domain must have links on certain key variables to another Domain. This is the setup used to satisfy those kinds of conditions. To setup this lookup, the `From` attribute is used to specify the name of the domain that values in the current domain will be compared against. The `Variable` attribute is used to specify which variables should be used for comparison between the two datasets, as a comma-separated list of remote / local variable pairs. This is done in the form `REMOTE-NAME == LOCAL-NAME`, and must be in this form even if the remote and local names are identical. Finally, the `Where` attribute behaves exactly like the `Variable` attribute, with the exception that the `Where` attribute has a literal value on the right-hand side, instead of a local variable value. This allows you to narrow the scope of the lookup.

The following Cross-Dataset Lookup Rule checks that all referenced USUBJIDs are listed in the Demographics dataset:

```
<val:Lookup ID="SD0064" Variable="USUBJID == USUBJID" From="DM" When="USUBJID != ''"
  Message="Invalid subject" Description="Identifies non-Demographics domain
  subjects not found in the Demographics domain" Category="Cross-reference"
  Type="Error" Severity="High"/>
```

### External File Lookup:

The format of the external file lookup is nearly identical to that of the cross-dataset one. The only difference is that instead of using the `From` attribute to specify the dataset name, a file path is used instead with a prefix that tells the Validator how to parse it. This path may be relative to the directory where the OpenCDISC Validator is running, or it may be an absolute path. The prefix takes the form `FILE:Type:Path`, where `Type` may be `CSV`, `TAB` (tab-delimited text), `PIPE` (pipe-delimited text), or `XPT` (SAS).

The External File Lookup is especially useful for validating data against CDISC Controlled Terminology, which is published by NCI-EVS [5] as a tab-delimited text file. Here is an example Controlled Terminology rule:

```
<val:Lookup ID="CT0004" Variable="CDISCSubmissionValue == AGEU"
  From="FILE:TAB:%System.ConfigDirectory%/data/SDTM Terminology.txt"
  Where="CodelistCode == 'C66781'" When="AGEU != ''" Message="Value for AGEU not
  found in AGEU controlled terminology codelist" Description="Variable values
  should be populated with terms found in 'Age Unit' (C66781) CDISC controlled
  terminology codelist" Category="Terminology" Type="Error" Severity="High"/>
```

The `%System.ConfigDirectory%` string refers to the path of the directory containing the configuration file, allowing users to specify a relative path to the external lookup file from that location.

### General Notes:

The latest Validator release (version 1.1) added the `WhereFailure` attribute to the Lookup Rule, which allows the failure of the `Where` condition to be interpreted in different ways. The default value, `Fail`, produces failure messages when the `Where` condition evaluates to false. The other possible value, `Ignore`, allows a failure of the `Where` condition to be ignored, effectively limiting the scope of the lookup to a particular subset of the target data.

## PhUSE 2010

### Metadata Rule

The Metadata Rule checks the metadata of other datasets based on the data in the local dataset. This functions similarly to the Lookup Rule, but only checks the definition of the dataset, not its contents. This is the only Validation Rule that does not require the `Variable` attribute in all cases. For instance, you may only want to supply the `From` attribute, like in the case of SDTM validation rule SD0076.

The following is the definition of rule SD0076 that checks that the variable specified in IDVAR references an existing variable in the related domain:

```
<val:Metadata ID="SD0076" Variable="[IDVAR]" From="[RDOMAIN]"
  When="RDOMAIN != '' @and IDVAR != ''" Message="Referenced key variable not
  found" Description="Identifies Supplemental Qualifiers domain reference to a key
  variable that isn't defined in the target domain" Category="Consistency"
  Type="Error" Severity="High"/>
```

### Find Rule

The Find Rule is an existence check. Instead of comparing variable values, like most of the prior rules, it looks for existence of the values instead. This rule is especially helpful for checking for existence of datasets or variables described in `define.xml`. The `Terms` attribute is used to specify the values to find.

The following example defines a Find Rule that checks that all variables specified in `define.xml` for a particular domain exist in the dataset:

```
<val:Find ID="SD0054" Variable="VARIABLE" Terms="%Variables.Define%"
  Message="Variable in define.xml not in dataset" Description="Variables listed in
  the data definition document (define.xml) should be included in the dataset."
  Category="Metadata" Type="Warning" Severity="Low"/>
```

## CONFIGURING THE VALIDATOR USING SAS

As described above, the checks that the validation engine performed on the data are defined in a customizable configuration file. When downloaded, the tool comes with a number of standard configuration files, such as those for SDTM versions 3.1.1 and 3.1.2. In many cases the configuration files can be used without the need for any changes, however in some situations a specific user might want to use a more situation-specific set of rules. It is often necessary for a user to switch off some of the checks as they are not applicable for a given sponsor, or to add sponsor-defined controlled terminology to existing CDISC extensible code lists. This section describes how the configuration files can be modified to meet specific needs.

When there are only a few minor changes needed to an already existing configuration file, the best solution is to simply manually edit the XML file, with a normal ASCII file editor. Just manually editing the configuration files is a good way to make few and simple changes and to get acquainted with the configuration of the Validator.

In most cases however it is not feasible to make the changes manually. SAS can be used to modify the XML files, using normal base SAS. The first example below shows how to use base SAS code to modify the configuration, to turn a check SD0066 off.

```
data config ;
  infile 'C:\temp\opencdisc-validator\config\config-sdtm-3.1.2.xml'
  firstobs=1 lrecl=4096 dlm="$" dsd trunccover;
  input line :$CHAR1000. ; *That format conserves the leading spaces;
run;

data config;
  set config;
  *Changes SD0066 from active to deactivate;
  if index(line, '<val:ValidationRuleRef RuleID="SD0066" Active="Yes"/>')>0 then do;
    line = tranwrd (line, 'Active="Yes"', 'Active="No"');
  end;
run;

data _null_;
  set CONFIG;
  file 'C:\temp\ opencdisc-validator\config\config-sdtm-new.xml' lrecl=4096;
  *Conserve leading spaces;
```

## PhUSE 2010

```
a=length(line)-length(left(line));
put @a line $;
run;
```

The first datastep above, reads in the configuration file located in "C:\temp\opencdisc-validator\config", the second step changes the `Active` attribute, from Yes to No, and the third step writes out a new updated copy of the configuration file.

The example below illustrates how to add a codelist item to one of the existing codelists in the file `SDTM_Terminology.txt`. In particular, it addresses a need to add the codelist item 'VERY VERY SMALL', to the codelist named `Size`.

```
data _null_;
  infile 'C:\temp\opencdisc-validator\config\data\SDTM Terminology.txt'
  dlm='09'X dsd truncover firstobs=1 lrecl=4096;

  input
  code :$200.
  codelist_code :$200.
  codelist_ext :$200.
  codelist_name :$200.
  cdisc_sub_value :$200.
  cdisc_synonym :$2000.
  cdisc_def :$2000.
  nci_pref :$2000. ;

  file 'C:\temp\opencdisc-validator\config\data\SDTM Terminology_new.txt'
  delimiter='09'x DSD DROPOVER lrecl=32767;

  do;
    put code $ @;
    put codelist_code $ @;
    put codelist_ext $ @;
    put codelist_name $ @;
    put cdisc_sub_value $ @;
    put cdisc_synonym $ @;
    put cdisc_def $ @;
    put nci_pref $ ;
  end;

  *Create the new codelist item;
  if codelist_name='Size' and cdisc_sub_value='SMALL' then do;
    code='new'; cdisc_sub_value='VERY VERY SMALL';
    cdisc_def=''; nci_pref='';

    put code $ @;
    put codelist_code $ @;
    put codelist_ext $ @;
    put codelist_name $ @;
    put cdisc_sub_value $ @;
    put cdisc_synonym $ @;
    put cdisc_def $ @;
    put nci_pref $ ;
  end;
run;
```

In the example above, the tab delimited ASCII file containing the `data _null_ step` imports the records from the tab delimited file 'SDTM Terminology.txt' then adds a record and the corresponding new tab delimited file is created, with a new name. For the change to take effect, the original tab delimited file should be overwritten by the new one.

## PhUSE 2010

Both code samples above are examples of how SAS can be used to make simple changes to OpenCDISC Validator configuration files. If more sophisticated changes are needed, more sophisticated SAS code might be needed, but the principle steps remain the same:

- Import the configuration file
- Make the change to it
- Export the configuration file.

### DISPARATE WAYS OF VALIDATING CDISC SDTM DATA

As mentioned previously, there are currently several different methods to validate CDISC SDTM data with each having a differing interpretation of what checks should be performed. There are even multiple interpretations at the FDA. The WebSDM validation rules were shown to disagree with rules defined by the Janus specification [6]. Therefore, compliance with WebSDM rules is not sufficient to ensure the data will be successfully loaded or be useful to the tools within Janus. Likewise, the various commercial validation tools each have individual definitions of validations, resulting in wide-spread inconsistencies concerning what it means to validate SDTM data, even when the countless home grown tools within the industry are not considered.

Additionally, not all validation done by the different checkers is necessarily based on rules from the CDISC SDTM definition or implementation guide. For example, Sergiy Sirichenko of PAREXEL presented at the CDISC European Interchange in 2010 how some WebSDM rules are not, strictly speaking, based on the CDISC SDTM [7].

To address this problem, OpenCDISC has created a public Validation Rules Repository, pooling together the knowledge and resources of the CDISC community to produce a central listing of validation rules for each standard [8]. A shared understanding of how to validate SDTM, as well as other CDISC content models, is needed to achieve a higher degree of similarity between SDTMs produced by different companies. Just recently the CDISC ADaM team has published a (draft) list of validations to be used on CDISC ADaM data [9]. A similar list is being prepared by CDISC Advisory Board (CAB) Validation Project team for SDTM.

### CDISC SHARE PROJECT

The CDISC Shared Health and Research Electronic Library (SHARE) project is a recent CDISC project, which can improve the way that the CDISC standards are used and evolve. CDISC SHARE is a global, accessible electronic library, which will contain the existing CDISC standards in a machine-readable format. The environment will also allow for the addition of data elements for new therapeutic areas. The SHARE repository has not been created yet, but inclusion of validation rules in the repository would facilitate a shared understanding of how to validate CDISC data and provide a source of input for the configuration of validators such as the OpenCDISC Validator.

### CONCLUSION

OpenCDISC.org provides a free, open source validation tool, which is used industry wide by organizations working with CDISC data. The OpenCDISC Validator is flexible and can be configured to fit different versions of the CDISC models. The development of the tool is community-based, allowing for various perspectives from across the industry to be reconciled into a single, accessible validation resource.

Currently there are several different interpretations of how to validate CDISC SDTM. A more unified understanding of how to validate SDTM, as well as the other CDISC content models, is needed to achieve a higher degree of verifiable consistency in SDTM-compliant data and submissions. The CDISC ADaM team has recently defined a draft list of validations to use on ADaM. Development of such lists greatly aids in the correct adoption of these models by different companies, and it is suggested that other CDISC teams also define similar validation checklists to help reinforce these specifications.

CDISC is currently working to define the SHARE repository. It is suggested that such a repository should contain machine readable lists of validation checks such as the one defined by the ADaM team.

### REFERENCES

- [1] Guidance for Industry: Study Data Specifications. US Food and Drug Administration.  
<http://www.fda.gov/downloads/ForIndustry/DataStandards/StudyDataStandards/UCM199599.pdf>
- [2] Validation checks performed by WebSDM(tm) on SDTM version 3.1.1 Datasets. PhaseForward, Inc.  
<http://www.phaseforward.com/products/safety/documents/ValidationChecksPerformedbyWebSDMtm.Q107.pdf>
- [3] OpenCDISC Validation Framework.  
<http://www.opencdisc.org/projects/validator/opencdisc-validation-framework>
- [4] Case Report Tabulation Data Definition Specification (define.xml).  
[http://www.cdisc.org/stuff/contentmgr/files/0/464923b10ea16b477151fcaa9f465166/misc/crt\\_ddspecification\\_1\\_0\\_0.pdf](http://www.cdisc.org/stuff/contentmgr/files/0/464923b10ea16b477151fcaa9f465166/misc/crt_ddspecification_1_0_0.pdf)

## PhUSE 2010

- [5] NCI EVS Terminology Resources. <http://www.cancer.gov/cancertopics/terminologyresources/page6>
- [6] Carol Vaughn, Gregory Ridge, William Friggle (2008), Sanofi Aventis, Inc., *"Experiences Submitting CDISC SDTM and Janus Compliant Datasets"*, proceeding of the PharmaSUG 2008 conference, <http://www.lexjansen.com/pharmasug/2008/rs/rs04.pdf>
- [7] Sergiy Sirichenko's overview from CDISC European Interchange 2010. <http://www.opencdisc.org/downloads/websdm-to-sdtm.xls>
- [8] OpenCDISC Validation Rules Repository. <http://www.opencdisc.org/projects/validator/cdisc-validation-rules-repository>
- [9] CDISC ADaM Validation Checks Version 0.2 <http://www.cdisc.org/adam>

### CONTACT INFORMATION

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

Max Kanevsky  
Pinnacle 21, Inc  
600 W. Germantown Pike, Suite 400  
Plymouth Meeting, PA 19462  
Work Phone: 215-858-1126  
Email: [mkanevsky@pinnacle21.net](mailto:mkanevsky@pinnacle21.net)  
Web: [www.pinnacle21.net](http://www.pinnacle21.net)

Niels Both  
S-cubed  
Ørestads Boulevard 61F 6tv,  
2300 Copenhagen  
Denmark  
Work Phone: +45 3644 7677  
Email: [nb@s-cubed.dk](mailto:nb@s-cubed.dk)  
Web: [www.s-cubed.dk](http://www.s-cubed.dk)

Tim Stone  
Pinnacle 21, Inc  
600 W. Germantown Pike, Suite 400  
Plymouth Meeting, PA 19462  
Work Phone: 302-373-6907  
Email: [tim@pinnacle21.net](mailto:tim@pinnacle21.net)  
Web: [www.pinnacle21.net](http://www.pinnacle21.net)

Brand and product names are trademarks of their respective companies.