**PhUSE 2014**

**Paper TS04**

# Running OpenCDISC from SAS®

Mark Crangle, ICON Clinical Research, Marlow, United Kingdom

## ABSTRACT

The increasing importance of producing CDISC compliant datasets means this is no longer an afterthought when CDISC datasets are created – it is now an integral part of dataset design, production and validation. Although standards are considered when datasets are defined, often compliance is only checked once all datasets have been created. If any issues are identified, changes must be made and another round of validation conducted. If the checks could be automated as part of the development and validation process this would reduce the need for re-work.

OpenCDISC is an open source, metadata driven tool for ensuring clinical data compliance with CDISC standards including SDTM and ADaM. The tool utilizes xml metadata to apply the validation checks specified by CDISC and can be run using an easy to use Java GUI. Commonly the tool is run on a full package of clinical data but it can also be run on individual datasets to ensure that dataset's compliance to the CDISC standards. This paper briefly explores the merits of the OpenCDISC validator and goes on to explain how the tool can be configured to work for individual datasets, how to run from SAS and how the results can be processed.

## INTRODUCTION

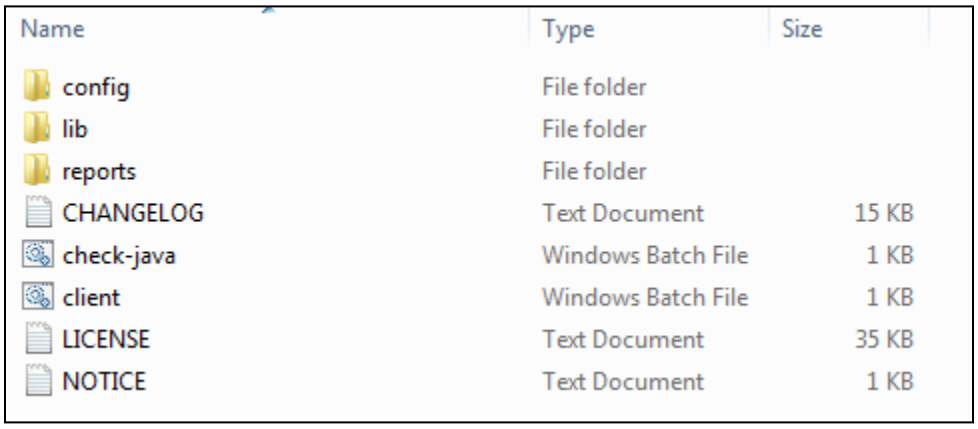
The OpenCDISC Validator is an open source, metadata driven tool, written in Java that can be used for ensuring clinical datasets are compliant with CDISC standards. Being written in Java, with a built in command-line interface in addition to a graphical user interface (GUI), it can be used on a variety of operating systems. Commonly the tool is run to check consistency across a full package of data once they have already been validated versus the SAP/Dataset Specs but the metadata driven nature of the tool means that it can be configured to work with individual datasets. This can be coupled with the ability to run the tool from within SAS and to read the output from the tool back into a SAS dataset to integrate the CDISC compliance checks into the validation process.

Commonly, the tool is run across a package of data, once all the datasets have been completed, validated and converted to xpt format for submission, but this can result in re-work if datasets definitions need to be re-visited when issues are found. A more efficient method would be to use the tool at the time of dataset validation so that the derivations are checked against the dataset definitions and CDISC standards at the same time. The configuration files for the validation tool can be adapted to make this possible and the tool can be run directly from SAS to allow it to be incorporated into validation programs.

The solutions explored in this paper are applicable to the validation of both SDTM and ADaM datasets and on a variety of operating systems but the examples shown will focus on ADaM datasets and use the Windows 7 operating system. The OpenCDISC validator is regularly updated and this paper is using version 1.5.

## USING THE OPENCDISC VALIDATOR

The OpenCDISC validator is packaged with an easy to use graphical user interface (GUI) which is the most common method of using the tool. The GUI is run using the client.bat file that is packaged with the Windows version of the tool:



| Name | Type | Size |
|---|---|---|
| config | File folder | |
| lib | File folder | |
| reports | File folder | |
| CHANGELOG | Text Document | 15 KB |
| check-java | Windows Batch File | 1 KB |
| client | Windows Batch File | 1 KB |
| LICENSE | Text Document | 35 KB |
| NOTICE | Text Document | 1 KB |

Although this paper focuses on using the command line interface to the OpenCDISC validator, the options available in the GUI correspond to those available from the command line so are described below:

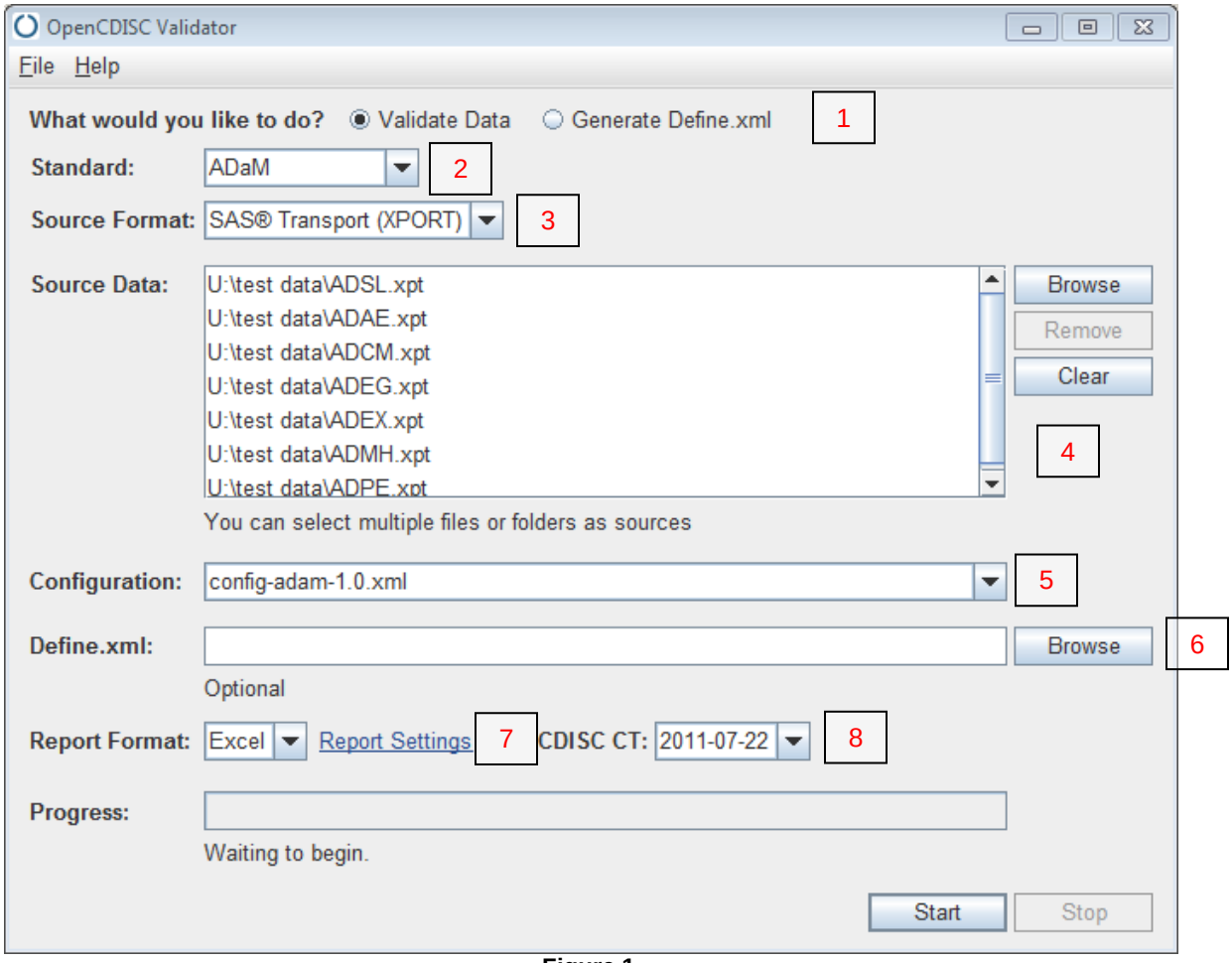



**Figure 1**

**1:** It is possible to use the OpenCDISC validator to create the Define.xml that is required for electronic submission. This functionality is not relevant to this paper.

**2**: As mentioned above, the tool can be used to validate different types of CDISC standard items. This paper will focus on ADaM data but the same approach will also work for SDTM data or for custom tabular data.

**3:** It is possible to include datasets in xpt, xml or delimited format. This paper will use xpt format datasets as this is the format required for electronic submission of standard data.

**4:** Datasets to be checked are specified here.

**5:** The configuration specifies the checks that will be performed on the data and the metadata that these checks will be using. The OpenCDISC validator is packaged with configuration files the use the latest versions of the implementation guides and validation checks from CDISC. It is possible to change these files to alter the checks being performed, change the metadata used or create new files from scratch if using the tool to validate custom data tabulations.

**6:** If a define.xml has already been created for your data then this can be specified here to provide more relevant data checks.

**7:** The validation report can be generated in excel or CSV. Excel format is the easiest to read but the CSV file can be more easily read into a SAS dataset if required. The report settings link can be used to specify the filename of the report.

**8:** The tool is packaged with different versions of the ADaM, SDTM and SEND Controlled Terminology which can be selected here.
When using the GUI, the report destination cannot be changed and the default location is in the reports subfolder of the OpenCDISC package. As shown below, when using the command line interface to run the validation tool, the location of the report can be set explicitly so there is greater flexibility about where the report is saved.

Once these options are set then the report can be generated using the "Start" button. As mentioned above, the report can be generated in either an excel or CSV format. Commonly (as this is the default) the GUI will be used to generate an excel file which contains all of the individual errors (up to 65535) found and some summaries of the number of errors by type and by dataset while the CSV format just contains a list of the errors used.

## OPENCDISC METADATA

The rules that the OpenCDISC validator uses are specified in xml configuration files. Versions of these for ADaM, SDTM and SEND datasets (as well as define.xml) are packaged with the tool and are saved in the config subfolder of the downloaded directory:

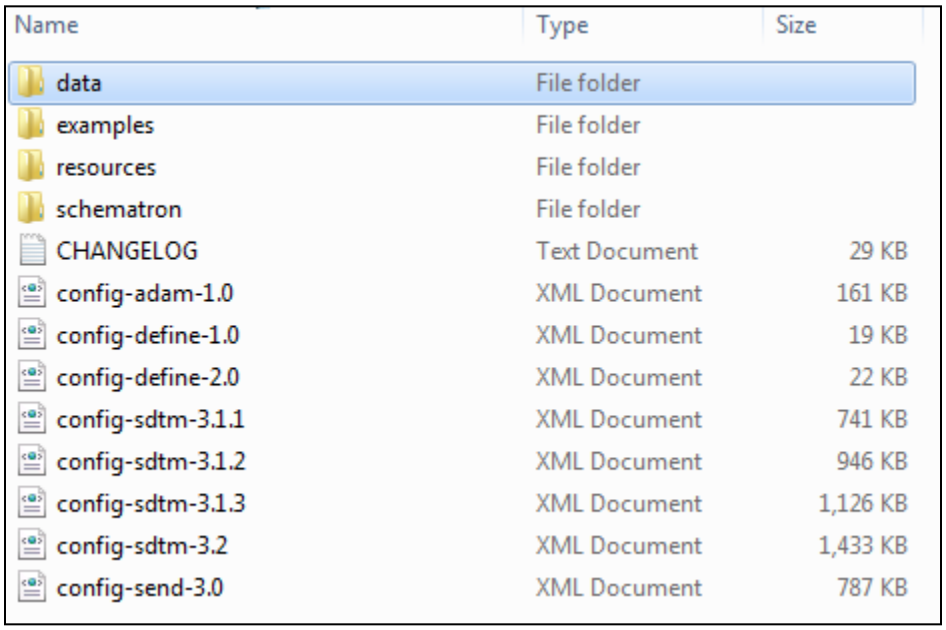
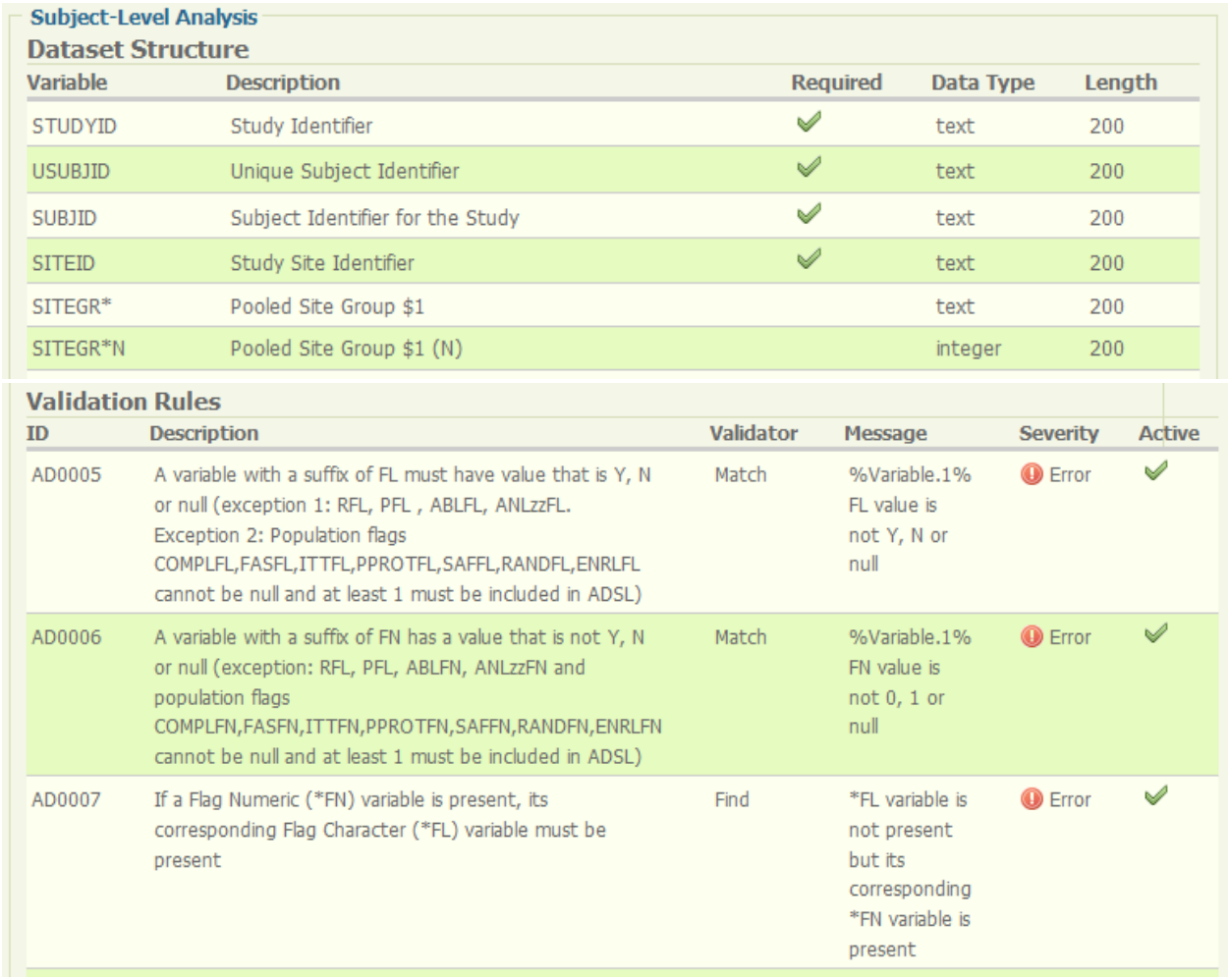



**Figure 2**

As you can see, configuration files for each version of the standards released by CDISC so different configuration files can be selected depending on which version is used for the datasets. These files incorporate the variable and dataset metadata associated with each version of the standards and the checks used which are defined using the validation checks published by CDISC and by the OpenCDISC community.

The configuration files can be opened in a web browser (such as Internet Explorer) where the xml files are rendered using an XSL stylesheet. In this format, the file is split into sections, one for global rules, then one for each data type. For example, in the ADaM configuration, there is one section for the ADSL domain and one for the BDS structure. Each section is then broken down into a subsection for dataset structure that is checked against and one for the validation rules that are used.



Subject-Level Analysis

Dataset Structure

| Variable | Description | Required | Data Type | Length |
|---|---|---|---|---|
| STUDYID | Study Identifier | ✓ | text | 200 |
| USUBJID | Unique Subject Identifier | ✓ | text | 200 |
| SUBJID | Subject Identifier for the Study | ✓ | text | 200 |
| SITEID | Study Site Identifier | ✓ | text | 200 |
| SITEGR* | Pooled Site Group $1 | | text | 200 |
| SITEGR*N | Pooled Site Group $1 (N) | | integer | 200 |

Validation Rules

| ID | Description | Validator | Message | Severity | Active |
|---|---|---|---|---|---|
| AD0005 | A variable with a suffix of FL must have value that is Y, N or null (exception 1: RFL, PFL , ABLFL, ANLzzFL. Exception 2: Population flags COMPLFL,FASFL,ITTFL,PPROTFL,SAFFL,RANDFL,ENRLFL cannot be null and at least 1 must be included in ADSL) | Match | %Variable.1% FL value is not Y, N or null | Error | ✓ |
| AD0006 | A variable with a suffix of FN has a value that is not Y, N or null (exception: RFL, PFL, ABLFN, ANLzzFN and population flags COMPLFN,FASFN,ITTFN,PPROTFN,SAFFN,RANDFN,ENRLFN cannot be null and at least 1 must be included in ADSL) | Match | %Variable.1% FN value is not 0, 1 or null | Error | ✓ |
| AD0007 | If a Flag Numeric (*FN) variable is present, its corresponding Flag Character (*FL) variable must be present | Find | *FL variable is not present but its corresponding *FN variable is present | Error | ✓ |

**Figure 3**

The checks defined in these configuration files are setup as if the tool is run across a package of data so there are a number of checks that check the contents of the package as a whole and check for consistency across the data. These checks include:

- Checking an ADaM package contains the required ADSL dataset. (AD0001)
- Checking subjects present in a dataset are also included in ADSL/DM (AD0053)
- Subjects in both DM and ADSL have consistent values for AGE, AGEU, SEX, RACE, SUBJID, SITEID, ARM (AD0204-AD0210)

These checks will not be relevant if we are using the tool to check individual datasets as part of the dataset validation process so we need to create a new version of the configuration with these checks deactivated.

In order to edit the configuration files it needs to be opened with a text editor such as Notepad or Wordpad which will give access to the underlying xml code. Each dataset type is defined within an ItemGroupDef tag and the validation rules applicable to that dataset type are defined by a val:ValidationRuleRef identifier. The description, message and severity are defined by the separate <val:ValidationRules> tag so in this way each validation rule can be applied to more than one dataset type by including a ValidationRuleRef in each ItemGroupDef that it applies to.

In order to de-activate a particular check we need to change the “Active” attribute of the ValidationRuleRef entry for the check we want to turn off.

```
<val:ValidationRuleRef RuleID="AD0001" Active="Yes"/>
<val:ValidationRuleRef RuleID="AD0061" Active="Yes"/>
<val:ValidationRuleRef RuleID="AD1010" Active="Yes"/>
```

In the default configuration files, all the rules have Active set equal to Yes. In order to de-activate a particular check this just needs to be set equal to No.

```
<val:ValidationRuleRef RuleID="AD0001" Active="No"/>
<val:ValidationRuleRef RuleID="AD0061" Active="Yes"/>
<val:ValidationRuleRef RuleID="AD1010" Active="Yes"/>
```

The file then needs to be saved back into the config folder. For the purposes of the updated file is saved as config-adam-indiv-1.0.xml. This file can then be viewed through a web browser and the checks altered are no longer being displayed as active:

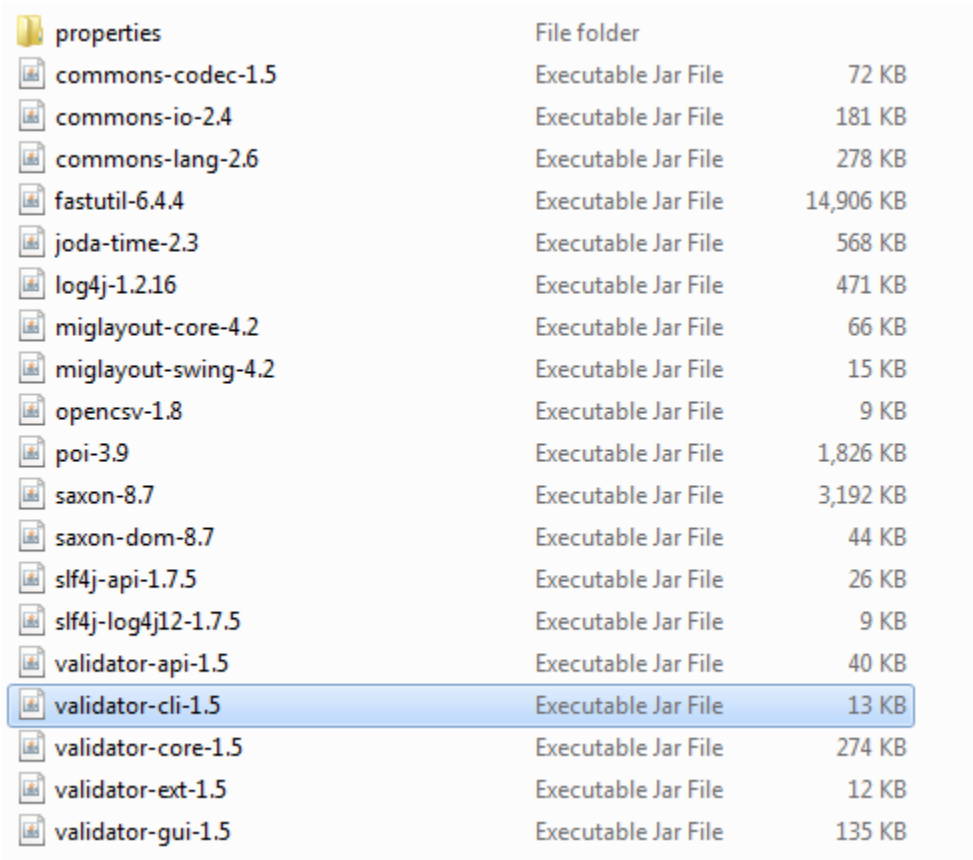
**Validation Rules**

| ID | Description | Validator | Message | Severity | Active |
|---|---|---|---|---|---|
| AD0001 | ADaM Subject level (ADSL) dataset should be included in every submission. | Find | Missing ADSL dataset | Error | |

**Figure 4**

## USING THE COMMAND LINE INTERFACE

Although using the gui is the most common method of running the validator all of the options from the gui are also available when using the command line interface (cli) method to run the validator. This interface is briefly documented on the OpenCDISC website but is not generally used as commonly as the gui. In the Windows version of the validator package, the command line interface is accessed using the validator-cli-1.5.jar file which is saved in the lib sub-folder.



As mentioned above, all of the options available in the GUI can be specified as arguments to the .jar file when it is run. The full range available options and keywords are documented on the OpenCDISC website and can also be viewed by opening the command line shell, moving to lib folder where the OpenCDISC validator is saved and running the command:

```
java -jar validator-cli-1.5.jar –help
```

The options applicable to validating CDISC datasets in xpt format are as follows:

| Parameter Type | Parameter | Valid Values | Default | Description | GUI Option (from Figure 1) |
|---|---|---|---|---|---|
| General | -task | Validate, Generate | Validate | Validate data or generate a Define.xml | 1 |
| | -type | SDTM, ADaM, SEND, Define, Custom | SDTM | Data Standard/Structure to validate | 2 |
| | | | | | |
| Source Data | -source | | | Path to the source data files | 4 |
| | -source:type | SAS, Delimited | SAS | File format (SAS corresponds to xpt files) | 3 |
| | | | | | |
| Configuration | -config | | | Path to the xml configuration document specifying the rules/metadata to validate | 5 |
| | -config:define | | | Path of the define.xml for the study | 6 |
| | -config:cdisc | | | CDISC Controlled Terminology Version | 8 |
| | | | | | |
| Report | -report | | | Path and filename where the validation report will be saved | 7 |
| | -report:type | Excel, CSV | Excel | Report format | 7 |
| | -report:cutoff | | 1000 | Number of times a record detail is printed per distinct domain issue | 7 |
| | -report:overwrite | yes, no | | indicates if report should be overwritten if same name provided | 7 |

In order to use the CLI, you will need to open a command shell session for your operating system. Depending on the operating system and how it is setup this can be done in a variety of ways but on most Windows systems this is accessed through the Start menu -> All Programs -> Accessories -> Command Prompt.

Once open, locate the folder where the OpenCDSIC tool is saved and go to the lib subfolder which contains the CLI file. The basic command to run the tool using the CLI is:

```
java -jar validator-cli-1.5.jar
```

This is then followed by specifying the parameters required for your run. For example, to match the options from Figure 1 the following call to the CLI would be used:

```
java -jar validator-cli-1.5.jar -task=Validate -type=ADAM -source="U:/test data/*.xpt" -
config="U:/test data/opencdisc-validator/config/config-adam-1.0.xml" -config:cdisc=2011-
07-22 -report="U:/test data/opencdisc-validator/reports/report.xls" -report:type=Excel -
report:cutoff=1000 report:overwrite=Yes
```

## INTEGRATING OPENCDISC INTO THE DATASET VALIDATION PROCESS

As we have now established, the OpenCDISC validator can be run from the command line, and this can be combined with the facitly to run operating system commands from a SAS session to run the OpenCDISC validator directly from SAS as part of dataset validation. Many methods of running system commands from SAS are available for a variety of operating systems and have been documented elsewhere so comparison of the various methods will not be conducted here. Due to issues with matching quotations around macro parameters the method below writes the commands to run the validator to a Windows batch file in a temporary location and this batch file is then run using the X Command that is available in SAS.

This facility can be combined with updates made to the configuration file to remove cross-package checks to allow individual datasets to be checked for CDISC compliance directly from a SAS session. The output from the validator can be read into a SAS dataset and viewed to determine compliance. This method could be housed within a macro that is called each time a dataset is validated so that compliance is checked “in-line” with other validation activities.

#### STEP 1 – SETTING UP MACRO PARAMETERS

A number of macro parameters will be needed in the validation macro as below:

```
%macro RunOpenCDISC(
 dsname  = , /*Name of dataset to be checked*/
 libname = , /*Library of dataset to be checked*/
 OCver   = , /*Version of OpenCDISC validator being used*/
 OCLoc   = , /*Location of the OpenCDISC validator and configuration file*/
 Config  = , /*Name of configuration file*/
 Batloc  = , /*Location where the batch file should be stored (usually a temporary
              folder*/
 Xptloc  =   /*Location where the XPT version of the dataset should be saved*/
);
```

As mentioned above, datasets need to be converted to XPT format before they can be used by the OpenCDISC validator. In this implementation of the solution this conversion is handled as part of the macro but this could also be routinely done as part of the dataset development process. The XPT files could also be stored in a temporary location and just used for OpenCDISC validation or could be stored in a permanent location and used as part of a CDISC submission.

Many of the macro parameters above could be given default values so that they don’t need to be specified each time the macro is called if the same values are often used.

#### STEP 2 – WRITING THE SYSTEM COMMANDS TO A BATCH FILE

As mentioned above this implementation of the solution writes the system commands needed to run the validator to a batch file due to issues with the multiple quotation marks needed and the fact that many of the parameters required for the system commands are macro variables. There may be other ways to run the commands while using some of the macro quoting functions available within SAS.

The batch file location is set using a filename statement

```
*Generate bat file;
filename batchout "&batloc\RunOpenCDISC-&dsname..bat";
```

Then the commands are written to the batch file using a datastep:

```
data _null_;
  file batchout;

  ...
  run;
```

The batch files are usually saved to a temporary location and then deleted at the end of the macro run but one possible advantage to this method is that the batch files for each dataset check could be saved and run directly when needed. This would mean the user wouldn’t have to navigate to the location of the OpenCDISC validator each time they wanted to recheck a dataset, they could just rerun the batch file that has already been created.

**STEP 3 – RUNNING THE BATCH FILE**

The batch file created in step 2 is then run using the X command which can pass commands to the operating system:

```
options xsync;
x " ""&batloc\runOpenCDISC-&program..bat"" ";
```

The xsync option is used so that SAS waits for the batch file to finish running before the next SAS statement is run. This is necessary in this case as the next part of the macro is to process the output from the OpenCDISC validator so this needs to complete running before this step.

Running the OpenCDISC validator over a large dataset can take some time, and although it is possible to modify the configuration of the validator to take advantage of multi-core processors, this would not reduce run times as this only allows each core to process a different dataset concurrently. Therefore, if this macro was programmed to be run at the end of each dataset validation program it would be worth considering a "switch" to turn off the validation check, perhaps when a rerun of all dataset was being done. In this case it would be more efficient to wait for all datasets to be completed then run the validator across the whole package, taking advantage of the multi-core processors that many computers have these days.

**STEP 4 – PROCESSING THE RESULTS FROM THE VALIDATOR**

As mentioned above, the default output from the OpenCDISC validator is in excel format, and this is certainly the most user friendly format but due to the formatting of the report is not as easy to read in to a SAS dataset. The CSV report that can be produced instead of an excel report is much more suited to this task. The CSV report will simply contain a list of all the errors, warnings and notes identified by the validator, and although it doesn't contain the summary of issues reports that the excel file has, this is not as relevant when only one dataset is being checked.

The CSV report that is generated is in the following format:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| Name | Record | Variable | Value | Rule ID | Message | Category | Type |
| ADLB | 25897 | PARCAT3, PARAMCD | Secondary Laboratory, HEOSLES | AD0124 | Inconsistent value for PARCAT3 within a unique PARAMCD | Consistency | Error |
| ADLB | 25898 | PARCAT3, PARAMCD | Secondary Laboratory, HHCTS | AD0124 | Inconsistent value for PARCAT3 within a unique PARAMCD | Consistency | Error |
| ADLB | 25899 | PARCAT3, PARAMCD | Secondary Laboratory, HHGBS | AD0124 | Inconsistent value for PARCAT3 within a unique PARAMCD | Consistency | Error |
| ADLB | 25900 | PARCAT3, PARAMCD | Secondary Laboratory, HLYMLES | AD0124 | Inconsistent value for PARCAT3 within a unique PARAMCD | Consistency | Error |
| ADLB | 25901 | PARCAT3, PARAMCD | Secondary Laboratory, HMONOLES | AD0124 | Inconsistent value for PARCAT3 within a unique PARAMCD | Consistency | Error |
| ADLB | 25902 | PARCAT3, PARAMCD | Secondary Laboratory, HNEUTLES | AD0124 | Inconsistent value for PARCAT3 within a unique PARAMCD | Consistency | Error |
| ADLB | 25903 | PARCAT3, PARAMCD | Secondary Laboratory, HPLATS | AD0124 | Inconsistent value for PARCAT3 within a unique PARAMCD | Consistency | Error |
| ADLB | 25904 | PARCAT3, PARAMCD | Secondary Laboratory, HRBCS | AD0124 | Inconsistent value for PARCAT3 within a unique PARAMCD | Consistency | Error |
| ADLB | 25905 | PARCAT3, PARAMCD | Secondary Laboratory, HWBCS | AD0124 | Inconsistent value for PARCAT3 within a unique PARAMCD | Consistency | Error |
| ADLB | 27236 | CRIT1, CRIT1FL | null, Y | AD0137 | CRITyFL is populated and CRITy is not populated | Consistency | Error |
| ADLB | 27236 | CRIT3FL, CRIT3 | Y, null | AD0137 | CRITyFL is populated and CRITy is not populated | Consistency | Error |
| ADLB | 27360 | CRIT1, CRIT1FL | null, Y | AD0137 | CRITyFL is populated and CRITy is not populated | Consistency | Error |
| ADLB | 27360 | CRIT3FL, CRIT3 | Y, null | AD0137 | CRITyFL is populated and CRITy is not populated | Consistency | Error |
| ADLB | 27675 | CRIT1, CRIT1FL | null, Y | AD0137 | CRITyFL is populated and CRITy is not populated | Consistency | Error |
| ADLB | 27675 | CRIT3FL, CRIT3 | Y, null | AD0137 | CRITyFL is populated and CRITy is not populated | Consistency | Error |
| ADLB | 28774 | CRIT1, CRIT1FL | null, Y | AD0137 | CRITyFL is populated and CRITy is not populated | Consistency | Error |
| ADLB | 28774 | CRIT3FL, CRIT3 | Y, null | AD0137 | CRITyFL is populated and CRITy is not populated | Consistency | Error |
| ADLB | 29841 | AVALC, PARAMCD, AVAL | 111, CAPOA1S, 111 | AD0150 | Inconsistent value for AVAL | Consistency | Error |
| ADLB | 31318 | AVALC, PARAMCD, AVAL | 111, CAPOA1S, 111 | AD0150 | Inconsistent value for AVAL | Consistency | Error |

Column A confirms the name of the dataset being validated.
Column B gives the record number when the issue has been found.
Column C shows the variables that are failing the check and column D gives the values of these variables where the check is failed.
Column E is the rule ID, column F briefly explains the issue and columns G and H identify the category and type of the issue. Columns E, F, G and H correspond to the information in the configuration file used.

It is also important to be able to recognize the report when the dataset has been checked and no issues found and when the dataset cannot be checked because it does not fit into either the ADSL or BDS dataset types. When the checks have been created the report just contains the header row and all subsequent rows are blank but when the dataset does not fit into either of the types that OpenCDISC recognizes then the follow report is created:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| Name | Record | Variable | Value | Rule ID | Message | Category | Type |
| ADMH | | | | MISSING_CONFIG | Unrecognized domain | System | Warning |

This case is usually acceptable as there will often be some datasets such as ADAE, ADMH etc. that don't fit into the BDS type but it may be something that has to be checked if there are datasets that are expected to be in the BDS type that have this message in the report.

There are a variety of ways to read in a CSV file which won't be covered in this paper but the simplest way perhaps is to use PROC IMPORT as this automatically handles determines variable types and lengths etc. Whatever method is used its important to note that the first row of the CSV file contains the column headers so any import should start from row 2. Once the report is in a SAS dataset, it is simply output using a PROC PRINT into the same output file as the PROC COMPARE output which is used to validate the dataset against the study specifications/SAP. The data is also summarized using a PROC FREQ to determine how many records are affected by each issue to indicate which the more serious issues are.

#### FUTURE ENHANCEMENTS

There is scope for using the report further once it has been read into SAS. Possible further uses include:

- The report could be checked programmatically to determine if there are any issues identified and further code run depending on whether the report is clean. Perhaps the PROC COMPARE code is only run once the dataset is CDISC compliant (although perhaps checking the PROC COMPARE output before running OpenCDISC would work better)
- The OpenCDISC output could be kept in a permanent dataset so that when the validator is rerun, output can be checked versus the previous report to track issues and identify whether any new issues have been found. This would also allow warning and notes identified by the OpenCDISC validator that are deemed to be acceptable discrepancies to be documented and tracked so that they don't appear in further reports
- As mentioned above, a "switch" could be programmed into the macro so that the OpenCDISC validator is only run when required. For example when a full rerun of all dataset is being done it might be more efficient to have the run the OpenCDISC validator and the end of the run across all

## CONCLUSION

As creating standard datasets are now vital to ensure accurate and transparent data, adherence to CDISC standards is now an important consideration at all stages of dataset specification, development and validation. The OpenCDISC validator is an open source tool for validating datasets but it is often only run once a package of datasets is completed, resulting in rework if issues are found.

As we have seen, the lesser known command line interface to the OpenCDISC validator allows the tool to be run from within a SAS session. This can be combined with updates to the OpenCDISC configuration file to remove checks that would be applied across dataset packages to allow CDISC compliance to be checked for individual datasets from SAS and "in-line" with other validation checks. The CSV report from the OpenCDISC validator can be read back into the SAS session to be analyzed, used for further processing and combined with other validation output, such as that from PROC COMPARE, to create a complete validation report confirming adherence both to study specifications and CDISC standards.

## REFERENCES

1. www.opencdisc.org
2. Let SAS handle your CDISC compliance check: automating OpenCDISC Validator in SAS, Edwin J. van Stein, Paper TS04, PhUSE 2012

## CONTACT INFORMATION

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

Mark Crangle
2 Globeside Business Park
Marlow
Buckinghamshire
United Kingdom
SL7 1HZ
Email: Mark.Crangle@iconplc.com