Running the CDISC Open Rules Engine (CORE) in BASE SAS©

Lex Jansen, CDISC, USA

ABSTRACT

CDISC Conformance Rules are an integral part of the Foundational Standards and serve as the specific guidance to Industry for the correct implementation of the Standards in clinical studies. The overall goal of the CORE Initiative is to provide a governed set of unambiguous and executable Conformance Rules for each Foundational Standard, and to provide an open-source execution engine for the executable Rules which are available from the CDISC Library [1][2]. The source code of the CORE engine is available on the GitHub repository. A Command Line Interface (CLI) is available on the repository which allows users to run the rules under Windows, Mac, and Linux. If users want to run the Engine in their own Python environment or tooling, it can be implemented as it is available on PyPi (Python Package Index).

For SAS users it is not always an option to run applications as a Command Line Interface.

The presentation will begin with a brief overview of CDISC CORE. The CORE Engine will then be covered. Then the presentation will describe a proof of concept where the CDISC CORE CLI commands have been implemented into SAS processes as Python functions in PROC FCMP, passing parameters and code to the Python interpreter and returning the results to SAS. These Python functions can be called and executed by user-defined SAS functions, which can be called from the DATA step or any context where SAS functions are available.

INTRODUCTION

CDISC Conformance Rules are an integral part of the CDISC Foundational Standards and serve as the specific guidance to Industry for the correct implementation of the Standards in clinical studies. The overall goal of the CORE (CDISC Open Rules Engine) Project is to deliver a governed set of unambiguous and executable Conformance Rules for each Foundational Standard, and to provide a reference implementation of an open-source execution engine for the executable Rules which are retrieved from the CDISC Library.

All code used in this paper and the latest version of the paper are available on GitHub: https://github.com/lexjansen/cdisc-core-sas. When encountering issues with the code, please open an issue at https://github.com/lexjansen/cdisc-core-sas/issues.

CORE CONCEPT

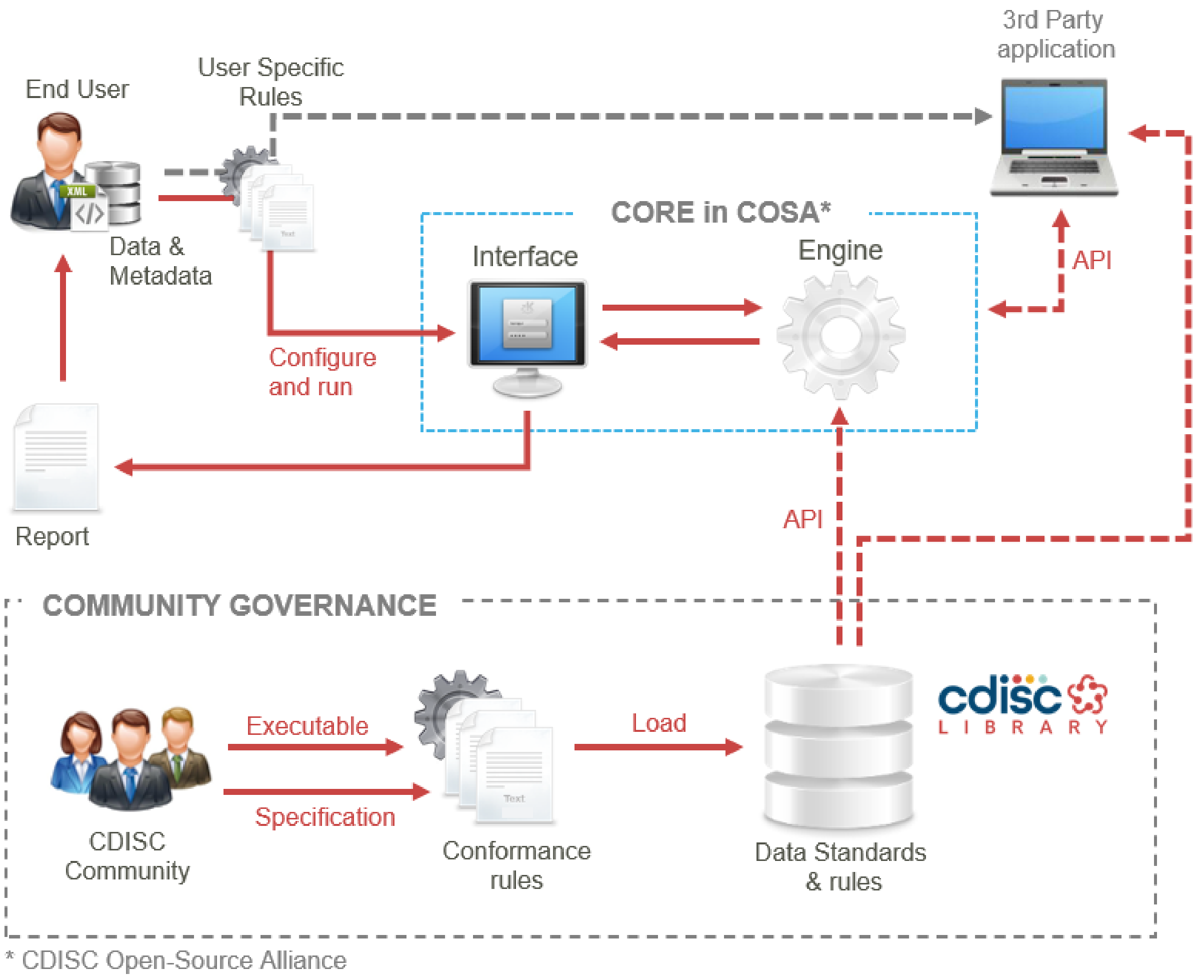
CORE consists of two parts:

- The Conformance Rules ("Rules")
- The Conformance Rules Engine ("Engine").

Figure 1 illustrates the CORE concept. CDISC, with the help of the CDISC Community, develops the rules according to the process that governs all CDISC Standards development. The Conformance Rules are stored in the CDISC Library along with the rest of the CDISC standards.

The Engine is an open-source software application whose purpose is to execute the Rules against clinical data and return results. The Engine is made available to the CDISC Community in GitHub and users may deploy it in a variety of processing environments including cloud, on-premises desktop, and on-premises server [3]. The Engine accesses the Rules from the CDISC Library via a Library API when it executes. Users may also add custom Rules for processing. The Engine is written in the Python programming language and comes with a permissive MIT open-source license.

Figure 1 The CORE Concept



CORE CONFORMANCE RULES

In the Rule development process human-readable Rule specifications are expressed in a machine-readable form. The human-readable Rule specification is interpreted by the Rule developer and authored in the CORE Rule Editor using a structured language (YAML). The CORE development team has developed a schema that defines the specific YAML syntax for expressing a Rule in YAML [4].

Just like the CORE Engine, the Rule Editor is open-source software that CDISC has made available for free to the CDISC Community via GitHub and which is listed in the CDISC Open Source Alliance (COSA). It has been released under the permissive MIT open-source license [5].

Figure 2 shows an example of a Rule in YAML. In this case the rule states that required variables (Core = “Req”) must be included in the dataset and cannot be null (lines 46-52). The rule is applicable to 3 SDTMIG versions (lines 7-45).The rule is applicable to all domains and classes (lines 66-72). Line 64 shows the message given: At least one Required variable has a null value.

More information about the CORE rule development process and the governance model for the rules can be found in references [2] and [6].

Figure 2 An Example CORE Rule in YAML

```yaml
# Variable: GEN
# Condition: Variable Core Status = Required
# Rule: Variable present in dataset and ^= null
Authorities:
  - Organization: CDISC
    Standards:
      - Name: SDTMIG···
      - Name: SDTMIG···
      - Name: SDTMIG
        References:
          - Citations:
              - Cited Guidance: Required variables must always be included in the dataset and
                  cannot be null for any record.
                Document: IG v3.4
                Section: 4.1.5
            Origin: SDTM and SDTMIG Conformance Rules
            Rule Identifier:
              Id: CG0014
              Version: '1'
            Version: '2.0'
        Version: '3.4'
Check:
  any:
    - all:
        - metadata: $var_perm
          operator: variable_metadata_equal_to
          value: Req
        - operator: empty
Core:
  Id: CORE-000356
  Status: Published
  Version: '1'
Description: 'Part B: Raise an error when a Required variable is null.'
Executability: Fully Executable
Operations:
  - id: $var_perm
    name: core
    operator: variable_library_metadata
Outcome:
  Message: At least one Required variable has a null value
Rule Type: Record Data
Scope:
  Classes:
    Include:
      - ALL
  Domains:
    Include:
      - ALL
Sensitivity: Record
```

THE CORE ENGINE

There are several ways to run the CORE Engine:

- As a CLI (Command Line Interface), which allows you to run the rules under Windows, Mac, and Linux. The compiled package can be downloaded, unzipped, and run [7].
  The repository also contains the steps for creating an executable version.

- Clone the repository and run `python core.py` from the root of the CORE project with appropriate parameters. See `python core.py --help` to see the full list of commands [3].
- An alternative to running the validation from the command line is to instead import the rules engine library in Python (available as a package on PyPi) and run rules against data directly (without needing your data to be in .xpt format) in your own environment or tooling [8].

This paper is based on version v.6.3, specifically the Windows CLI (core-windows.zip, October 12, 2023).

The last 2 ways of running - using the non-compiled version – require cloning the cdisc-rules-engine GitHub repository and installing dependencies:

- Clone the repository:

```
git clone https://github.com/cdisc-org/cdisc-rules-engine.git
```

- Create a virtual environment:

```
python -m venv <virtual_environment_name>
```

- Activate the virtual environment:

```
.\<virtual_environment_name>\Scripts\Activate -- on windows
```

- Install the requirements.

```
python -m pip install -r requirements.txt
```

RUNNING THE CORE ENGINE AS A CLI

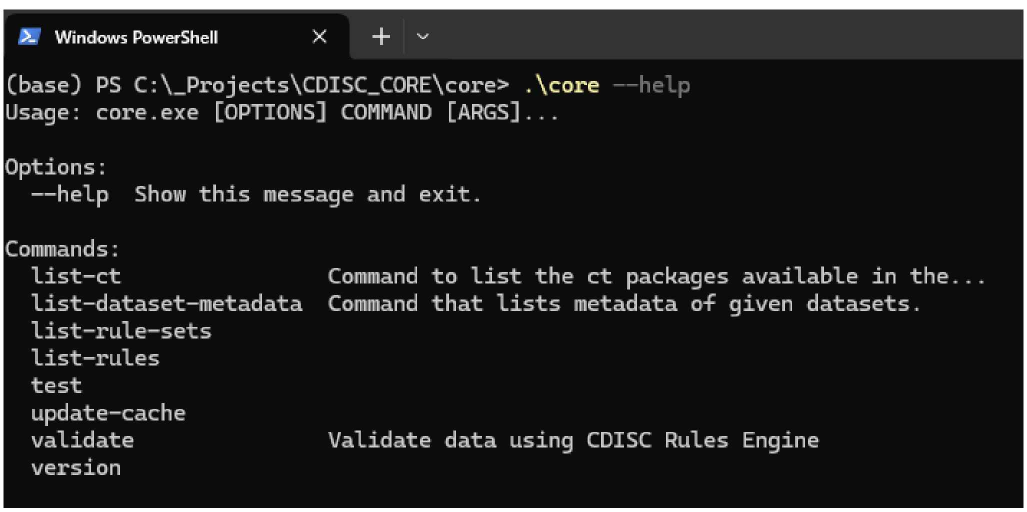
After the CORE Command has been downloaded and unzipped users can run CORE from a command prompt. Examples of commands will only be given for the Windows operating system. The commands are very similar for other distributions.

To see all available commands run:

```
.\core –help
```

Figure 3 shows the available CORE commands.

Figure 3 Available CORE CLI commands



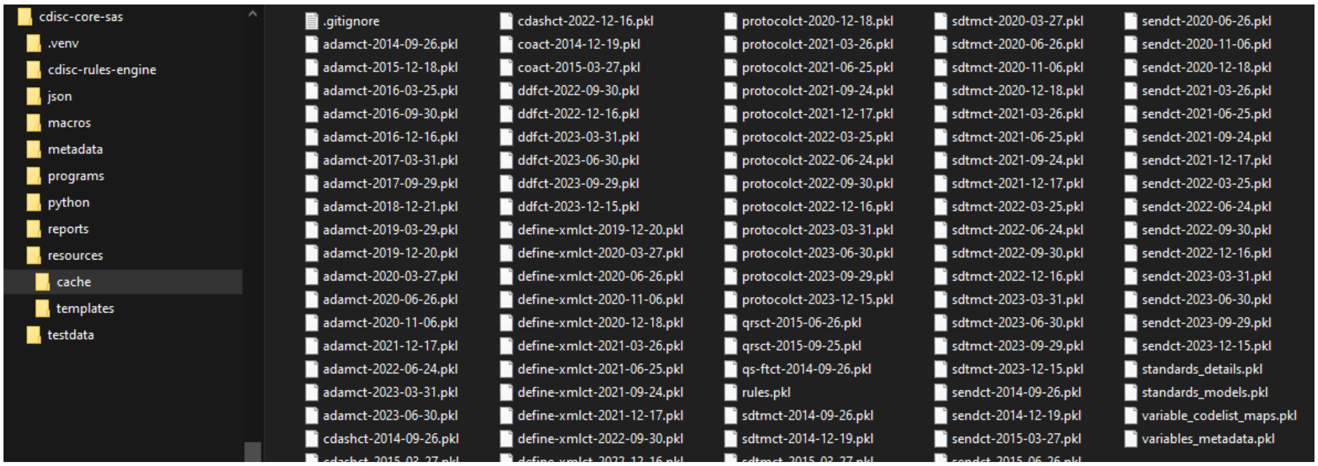
The CORE Cache

The CORE Engine stores rules and standards metadata from the CDISC Library as Python pickle files in a local cache folder. Pickle files typically have the . pck extension.

> *"Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling" or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling". [9].*

Figure 4 shows an example of a local CORE cache folder on Windows.

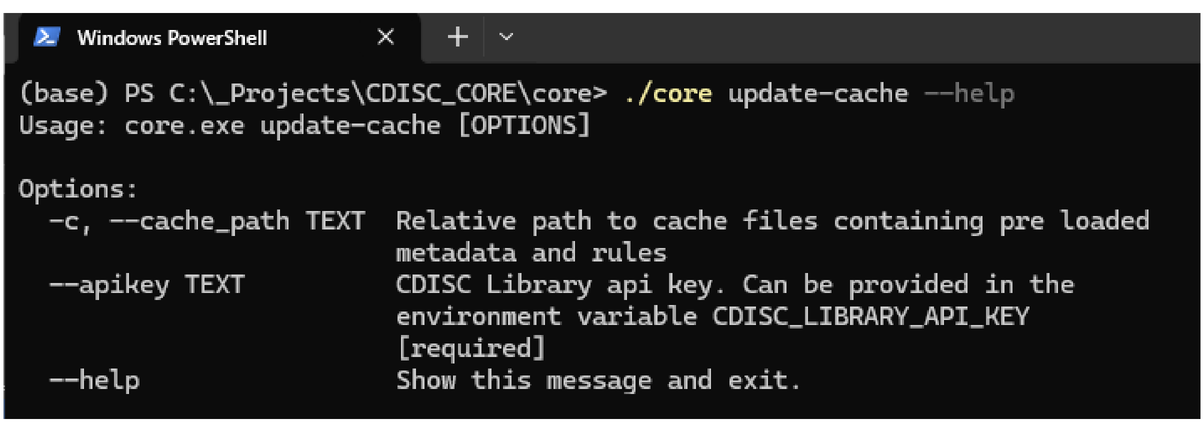
Figure 4 CORE local cache folder with pickle files



Rules get added to the CDISC Library on a regular basis. At any moment in time, the locally stored cache can be updated with the `update-cache` command (see Figure 5) to get the latest set of rules from the CDISC Library. Accessing the CDISC Library requires an API key, which is recommended to define as an environment variable - `CDISC_LIBRARY_API_KEY`.

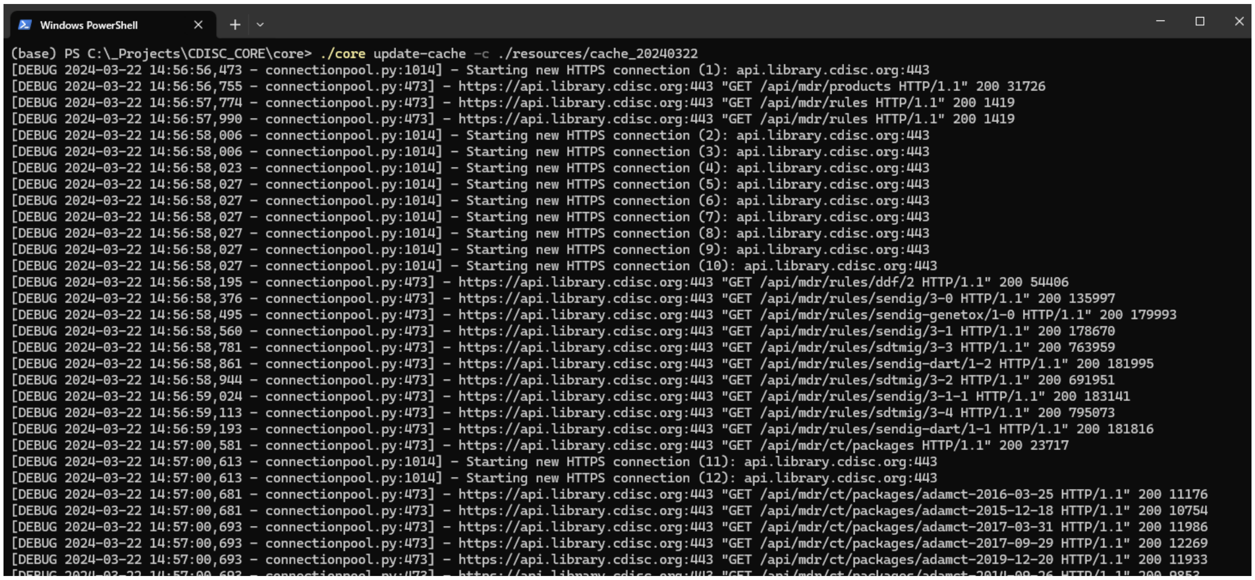
To obtain an API key, please follow the instructions found on the CDISC Wiki [10]. Please note it can take up to an hour after signing up to have an API key issued

Figure 5 Update the CORE local cache folder with the update-cache command



By default, the `update-cache` command gets the API key from an environment variable (`CDISC_LIBRARY_API_KEY`) and the default value for the cache path is: `./resources/cache`. However, both can be specified as parameters (Figure 6).

Figure 6 Updating the CORE local cache folder with the update-cache command



Warnings like the following can be ignored:

```
[WARNING 2024-03-22 15:01:08,502 - connectionpool.py:322] - Connection pool
is full, discarding connection: api.library.cdisc.org. Connection pool
size: 10
```

Validating Data

The following command is used to validate data:

```
.\core validate -s <standard> -v <standard_version> -d path/to/datasets
# ex: .\core.exe validate -s sdtmig -v 3-4 -d .\xpt\
```

Figure 7 shows all available parameters for the validate command (`.\core validate –help`).

Figure 7 Parameters for the CORE validate command

```
Windows PowerShell

Usage: core.exe validate [OPTIONS]

  Validate data using CDISC Rules Engine

  Example:

  python core.py -s SDTM -v 3.4 -d /path/to/datasets

Options:
  -ca, --cache TEXT                 Relative path to cache files containing pre
                                    loaded metadata and rules
  -ps, --pool-size INTEGER          Number of parallel processes for validation
  -d, --data TEXT                   Path to directory containing data files
  -dp, --dataset-path TEXT          Absolute path to dataset file
  -l, --log-level [info|debug|error|critical|disabled|warn]
                                    Sets log level for engine logs, logs are
                                    disabled by default
  -rt, --report-template TEXT       File path of report template to use for
                                    excel output
  -s, --standard TEXT               CDISC standard to validate against
                                    [required]
  -v, --version TEXT                Standard version to validate against
                                    [required]
  -ct, --controlled-terminology-package TEXT
                                    Controlled terminology package to validate
                                    against, can provide more than one
  -o, --output TEXT                 Report output file destination
  -of, --output-format [JSON|XLSX]
                                    Output file format
  -rr, --raw-report                 Report in a raw format as it is generated by
                                    the engine. This flag must be used only with
                                    --output-format JSON.
  -dv, --define-version TEXT        Define-XML version used for validation
  -df, --data-format [xpt]          Format in which data files are presented.
                                    Defaults to XPT.  [required]
  --whodrug TEXT                    Path to directory with WHODrug dictionary
                                    files
  --meddra TEXT                     Path to directory with MedDRA dictionary
                                    files
  -r, --rules TEXT
  -p, --progress [bar|percents|verbose_output|disabled]
                                    Defines how to display the validation
                                    progress. By default a progress bar like
                                    "[████████████████████████████--------]
                                    78%"is printed.
  -dxp, --define-xml-path TEXT      Path to Define-XML
  --help                            Show this message and exit.
```

The following command will validate the XPT files in folder ..\data\sdtm against the SDTM-IG 3.3 standard using the CDISC/NCI Controlled Terminology package from December 15, 2023:

```
.\core validate -s sdtmig -v 3-3 -ct sdtmct-2023-12-15 -d ..\testdata\sdtm\
-o sdtmig-3-3-report
```

The result of this command will be an Excel spreadsheet called **sdtmig-3-3-report.xlsx**. Figure 8 shows some screenshots of this spreadsheet.

Figure 8 Validation Report in Excel

	A	B
1	Conformance Details	
2	Report Generation	2024-03-24T12:50:26
3	Total Runtime	32.08 seconds
4	CORE Engine Version	0.6.2
5		
6	Standards Details	
7	Standard	SDTMIG
8	Version	V3.3
9	CT Version	sdtmct-2023-12-15
10	Define-XML Version	2.1.0
11	UNII Version	Not configured
12	Med-RT Version	Not configured
13	MedDRA Version	Not configured
14	WHODRUG Version	Not configured
15	SNOMED Version	Not configured

Conformance Details | Dataset Details | Issue Summary | Issue Details | Rules Report

	A	B	C	D	E	F
1	Dataset	Label	Location	Modified Time Stamp	Size (kb)	Number of Records
2	ae.xpt	Adverse Events	..\testdata\sdtm	2020-08-21T09:14:28	38.08	74
3	cm.xpt	Concomitant Medications	..\testdata\sdtm	2020-08-21T09:14:26	39.44	68
4	dd.xpt	Death Details	..\testdata\sdtm	2020-08-21T09:14:25	4.08	3
5	di.xpt	Device Identifiers	..\testdata\sdtm	2020-08-21T09:14:24	16.8	34
6	dm.xpt	Demographics	..\testdata\sdtm	2020-08-21T09:14:29	13.04	18
7	ds.xpt	Disposition	..\testdata\sdtm	2020-08-21T09:14:29	22.24	53
8	ec.xpt	Exposure as Collected	..\testdata\sdtm	2020-08-21T09:14:26	848	1590
9	ex.xpt	Exposure	..\testdata\sdtm	2020-08-21T09:14:26	823.12	1583
10	fa.xpt	Findings About Events or Interventions	..\testdata\sdtm	2020-08-21T09:14:23	29.68	78
11	ft.xpt	Functional Tests	..\testdata\sdtm	2020-08-21T09:14:24	6058.16	4488

Conformance Details | Dataset Details | Issue Summary | Issue Details | Rules Report | +

	A	B	C	D	E
1	Dataset	CORE-ID	Message	Issues	Explanation
2	AE	CORE-000266	If AESER = "N" then none of the seriousness criteria (AESCAN, AESCONG, AESDISAB, AESDTH, AESHOSP, AESLIFE, AESOD, AESMIE) could be equal to "Y".	1	
3	DM	CORE-000238	RFXENDTC does not equal the latest value of EX.EXSTDTC or EX.EXENDTC	1	
4	DS	CORE-000213	More than one record per subject per DSSCAT per EPOCH	34	
5	LB	CORE-000289	LBORRES is not a continuous measurement or Empty, when LBORNRHI is not empty.	6	
6	LB	CORE-000290	LBORRES is not a continuous measurement or Empty, when LBORNRLO is not empty.	6	
7	LB	CORE-000298	LBORRES is not a numeric result/empty and LBSTNRLO is not empty.	6	
8	LB	CORE-000299	LBORRES is not a numeric result/empty and LBSTNRHI is not empty.	6	

Conformance Details | Dataset Details | Issue Summary | Issue Details | Rules Report | +

CORE-ID	Message	Executability	Datase	USUBJID	Reco	Seque	Variable(s)	Value(s)
CORE-000107	An appropriate subject identifier is not present. APID is required in all Associated Persons Data. In addition to STUDYID, DOMAIN, and TSSEQ being required for all domains based on one of the 3 general observation classes, one of USUBJID, APID, SPDEVID, or POOLID must also be present.	fully executable	TS				APID, DOMAIN, POOLID, SPDEVID, STUDYID, TSSEQ, USUBJID	Not in dataset, TS, Not in dataset, Not in dataset, CDISCPILOT01, 1.0, Not in dataset
CORE-000213	More than one record per subject per DSSCAT per EPOCH	fully executable	DS	CDISC001	2	2	DSSCAT, EPOCH	STUDY PARTICIPATION, TREATMENT
CORE-000213	More than one record per subject per DSSCAT per EPOCH	fully executable	DS	CDISC001	3	3	DSSCAT, EPOCH	STUDY TREATMENT, TREATMENT
CORE-000213	More than one record per subject per DSSCAT per EPOCH	fully executable	DS	CDISC002	5	2	DSSCAT, EPOCH	STUDY TREATMENT, TREATMENT
CORE-000213	More than one record per subject per DSSCAT per EPOCH	fully executable	DS	CDISC002	6	3	DSSCAT, EPOCH	STUDY PARTICIPATION, TREATMENT
CORE-000213	More than one record per subject per DSSCAT per EPOCH	fully executable	DS	CDISC003	8	3	DSSCAT, EPOCH	STUDY PARTICIPATION, TREATMENT

Conformance Details | Dataset Details | Issue Summary | Issue Details | Rules Report

CORE-ID	Version	CDISC RuleID	FDA RuleID	PMDA RuleID	Message	Status
CORE-000001	1	CG0176			IECAT equals "INCLUSION" and IEORRES is not equal to "N".	SUCCESS
CORE-000002	1	CG0208			SESTDTC is required.	SUCCESS
CORE-000003	1	CG0299			TRLOC is present when TRLOC is not included in the TR domain.	SKIPPED
CORE-000004	1	CG0101			ECOCCUR indicates dose was not given, but ECDOSE is not blank or has a value less than or equal to 0	SUCCESS
CORE-000005	1	CG0102			EXTRT is PLACEBO, but EXDOSE is not equal to 0.	SUCCESS
CORE-000006	1	CG0131			DTHFL is not "Y" or null	SUCCESS
CORE-000007	1	CG0435			DTHFL is not "Y", when DTHDTC is populated	SUCCESS
CORE-000008	1	CG0132			DTHFL is not "Y", when SS.SSSTRESC equals "DEAD"	SKIPPED
CORE-000009	1	CG0152			ELEMENT variable has a non-null value when ETCD has a value of 'UNPLAN'	SUCCESS

Conformance Details | Dataset Details | Issue Summary | Issue Details | Rules Report

The Rules Report tab displays the run status of each rule selected for validation. The possible rule run statuses are:

- SUCCESS - The rule ran, and data was validated against the rule. May or may not produce results.
- SKIPPED - The rule was unable to be run. Usually due to missing required data but could also be cause by rule execution errors.

After cloning the cdisc-rules-engine GitHub repository and installing dependencies the CLI can also run as a Python program. Here is an example:

```
python core.py validate -s sdtmig -v 3-3 -ct sdtmct-2023-12-15 -d
..\testdata\sdtm\ -o sdtmig-3-3-report
```

RUNNING THE CORE ENGINE IN SAS

RUNNING A CLI (COMMAND LINE INTERFACE) IN SAS

SAS has various techniques to execute operating system commands:

- X statement
- SYSTASK statement
- %SYSEXEC statement
- CALL SYSTEM statement
- SYSTEM function
- FILENAMEC statement with the PIPE option

There are a few SAS options related to executing operating system commands:

- XSYNC- Controls whether an X command or statement executes synchronously or asynchronously.

- XWAIT - Specifies whether you must type EXIT at the DOS prompt before the DOS shell closes.
- XMIN - Specifies opening the application specified in the X command in a minimized state or in the default active state.

The following code builds commands to update the cache and then validates data with the CORE engine:

```
options noquotelenmax;
options noxwait xsync xmin;

%let core_exe = \_Projects\CDISC_CORE\core\core.exe;
%let project_folder = /_github/lexjansen/cdisc-core-sas;
%let core_cache_folder = &project_folder/resources/cache;
%let core_template = &project_folder/resources/templates/report-
template.xlsx;
%let test_data_folder = &project_folder/testdata/sdtm;
%let core_report = &project_folder/reports/sdtmig-3-3-report;
%let core_log = %sysfunc(pathname(work))/core_command;

%let core_command_1 = &core_exe update-cache;
%let core_command_1 = &core_command_1 -c &core_cache_folder;

%let core_command_2 = &core_exe validate;
%let core_command_2 = &core_command_2 -ca &core_cache_folder;
%let core_command_2 = &core_command_2 -rt &core_template;
%let core_command_2 = &core_command_2 -dp &test_data_folder/dm.xpt;
%let core_command_2 = &core_command_2 -dp &test_data_folder/ae.xpt;
%let core_command_2 = &core_command_2 -s sdtmig;
%let core_command_2 = &core_command_2 -v 3-3;
%let core_command_2 = &core_command_2 -ct sdtmct-2023-12-15;
%let core_command_2 = &core_command_2 -o &core_report;
%let core_command_2 = &core_command_2 -r CORE-000006 -r CORE-000007
-r CORE-000012 -r CORE-000013 -r CORE-000019 -r CORE-000266 -r CORE-
000356;

x "&core_command_1  > ""&core_log._1.log"" 2>&1";
%put &=sysrc;

data _null_;
  infile "&core_log._1.log" truncover;
  input;
  put _infile_;
run;

x "&core_command_2  > ""&core_log._2.log"" 2>&1";
%put &=sysrc;

data _null_;
  infile "&core_log._2.log" truncover;
  input;
  put _infile_;
run;
```

Note:

- The code only validates two datasets: ae.xpt and dm.xpt
- The code only validates against a limited set of rules, using the -r command line option

- Command output is saved to a file (`> ""&core_log._1.log"" 2>&1`) and printed to the log.

This approach works well, but there can be an issue. The assumption is that the X command is valid in the current SAS session. This may not be the case especially in shared SAS environments. In certain SAS environments SAS administrators may have specified the **NOXCMD** or **XCMD OFF** system options. When specified, the **NOXCMD** options disables the following:

- PIPE and NAMEPIPE device types in the FILENAME statement
- CALL SYSTEM routine
- X command
- Dynamic Data Exchange (DDE)
- %SYSEXEC macro
- SYSTASK statement
- PIPE and NAMEPIPE device types in the FILENAME function.

Indeed, specifying -NOXCMD at SAS invocation results in the following message in the SAS log:

```
51          ;
ERROR: Shell escape is not valid in this SAS session.
52
53          x "&core_command_1  > ""&core_log._1.log"" 2>&1"
53        !                                                  ;
```

The remainder of this paper will show how the use of PROC FCMP works around the **NOXCMD** limitation.

The CDISC CORE CLI commands will be implemented into SAS processes as Python functions in PROC FCMP, passing parameters and code to the Python interpreter and returning the results to SAS. These Python functions can be called and executed by user-defined SAS functions, which can be called from the DATA step or any context where SAS functions are available.

SAS PROC FCMP WITH PYTHON SUPPORT

Starting with the May 2019 release of SAS 9.4M6, the PROC FCMP procedure added support for submitting and executing functions written in Python from within a SAS session using the new Python object. FCMP, or the SAS Function Compiler, enables users to write their own functions and subroutines that can then be called from just about anywhere a SAS function can be used in SAS. Users are not restricted to using Python only inside a PROC FCMP statement. You can create an FCMP function that calls Python code, and then call that FCMP function from the DATA step [11][12].

Prerequisites

Before you can run Python code in SAS, you must install SAS 9.4M6 (May 2019 update) or later deployments. The following environment setup steps must be completed before you can use PROC FCMP to run the Python code [13].

- Install Python. The CDISC CORE engine requires Python 3.9 or Python 3.10.
- Set the **MAS_M2PATH** environment variable to specify the absolute path to the mas2py.py file included in your SAS installation. The mas2py.py file is used to execute Python code within a Python process that is launched by SAS Micro Analytic Service.
- Set the **MAS_PYPATH** environment variable to specify the absolute path to the Python executable.

In the SAS example in the GitHub repository (https://github.com/lexjansen/cdisc-core-sas) that contains the code used in this paper, the two environmental variables are define in a SAS configuration file:

```
options set = MAS_PYPATH = "&project_folder/.venv/Scripts/python.exe";
options set = MAS_M2PATH = "%sysget(SASROOT)/tkmas/sasmisc/mas2py.py";
```

The following (optional) environmental variables can be set at the operating system:

- **MAS_PYLOG_FILE** - Will create a local Python Logging file. Default: Filename is overwritten before logging. '+log.txt': will append data to 'log.txt'.
- **MAS_PYLOG_LEVEL** - The logging level: {ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF}. Default: WARN.
- **MAS_PYOUT_FILE** - Filename for python process STDOUT / STDERR. Default: Filename is overwritten before logging. '+out.txt': will append data to 'out.txt'.

These files can be useful when debugging the execution of Python functions by PROC FCMP.

To be able to run the code that comes with this paper, the user should clone the cdisc-core-sas GitHub repository (https://github.com/lexjansen/cdisc-core-sas), create a virtual Python environment, and then install the additional packages that are needed by the CDISC CORE engine:

- Clone the repository:

```
git clone https://github.com/cdisc-org/cdisc-rules-engine.git
```

- Create a virtual environment:

```
python -m venv <virtual_environment_name>
```

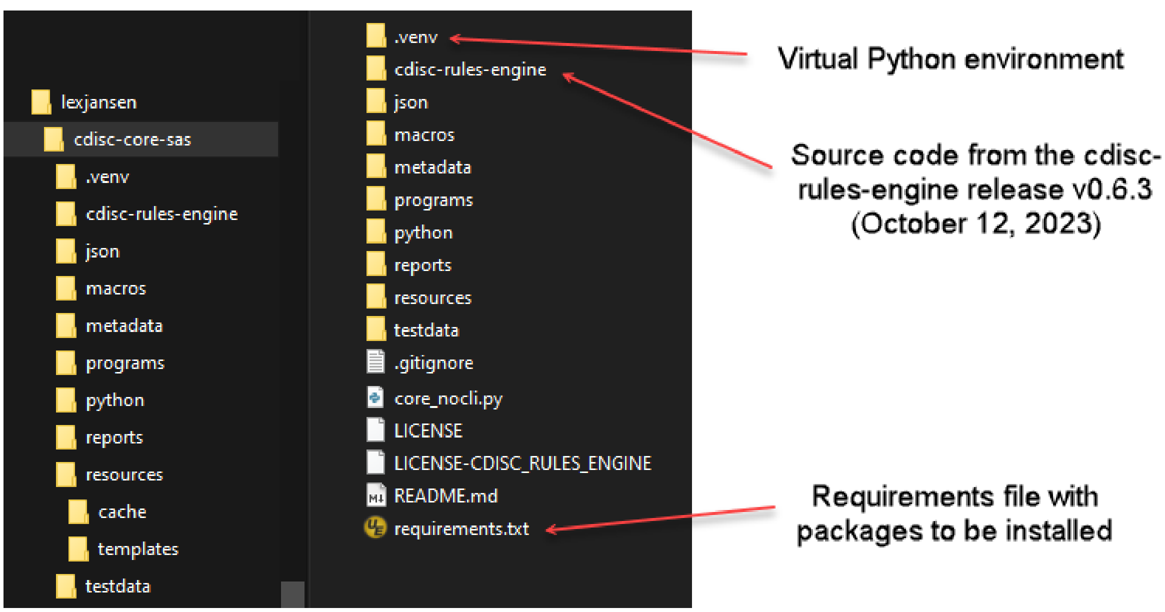
- Install the requirements.

```
python -m pip install -r requirements.txt
```

The cdisc-core-sas repository comes bundled with the source code of the v0.6.3 release (October 12, 2023) of the CDISC CORE engine.

Figure 9 shows an example of a cloned cdisc-core-sas repository with a virtual environment (.venv).

Figure 9 Cloned Github repository



The Python function called by SAS needs to know where to find the cdisc-rules-engine library. For this reason we need an operational system environment variable (**CORE_PATH**) that defines this location of the cdisc-rules-engine library[1].

It is also recommended to create an operational system environment variable (**CDISC_LIBRARY_API_KEY**) that has the API key for accessing the CDISC library.

You may have to work with your SAS administrators to implement these prerequisites.

PROC FCMP Python Objects

PROC FCMP Python objects enable you to embed and import Python functions into SAS programs. The Python code is not converted to SAS code. Instead, the Python code runs in the Python interpreter of your choice and returns the results to SAS. With a small Python code modification, you can run your Python functions from SAS and easily program in both languages at the same time [14][15].

A typical workflow for using Python objects in PROC FCMP is the following:

- Declare a Python object and a dictionary object
- Insert Python source code into SAS
- Publish Python source code
- Call the Python source code
- Return results from the dictionary

Example:

```
proc fcmp;
    declare object py(python);
    submit into py;
    def PyProduct(var1, var2):
        "Output: MyKey"
        newvar = var1 * var2
        return newvar,
        endsubmit;
    rc = py.publish();
    rc = py.call("PyProduct", 5, 10);
    MyResult = py.results["MyKey"];
    file log;
    put MyResult=;
run;
```

The Python object and all the Python object methods are valid only inside a PROC FCMP statement. For example, attempting to declare a Python object in a DATA step program results in an error. However, it is possible to call Python functions from the DATA step by creating PROC FCMP functions or subroutines that contain Python functions. PROC FCMP functions that contain Python functions are valid in the DATA step and can be called like other functions that are created using PROC FCMP.

[1] The repository related to this paper (https://github.com/lexjansen/cdisc-core-sas) includes the latest release of the cdisc-rules-engine repository (https://github.com/cdisc-org/cdisc-rules-engine) in the cdisc-rules-engine folder. The code related to this paper has been tested with the CORE_PATH environment variable pointing to this folder. You can point the CORE_PATH environment variable to a development version of the cdisc-rules-engine repository. However, you may have to update Python functions and macros to make the code related to this paper work with this development version. It is the intention of the author to update the code in https://github.com/lexjansen/cdisc-core-sas when new release of the CORE engine are released.

Also, by using the INFILE method, we can read external source code from a file into a Python object at parse time.

Example:

We have a file (timesfive.py) with the Python code that defines a Python function:

```
def TimesFive(PythonArg):
    "Output: MyKey"
    newvar = PythonArg * 5
    return newvar
```

We can create a SAS function MyPyFunc that calls this Python function. We can use MyPyFunc in a data step:

```
proc fcmp outlib=work.fcmp.pyfuncs;
  function MyPyFunc(FCMParg);
    declare object py(python);
    submit into py("&project_folder/python/timesfive.py");
    rc = py.publish();
    rc = py.call("TimesFive",FCMParg);
    MyFCMPResult = py.results["MyKey"];
    return(MyFCMPResult);
  endfunc;
run;

options cmplib=work.fcmp;
data _null_;
   x = MyPyFunc(5);
   put x=;
run;
```

The LOG file will display `x=25`.

There are a few limitations related to PROC FCMP functions:

- User-defined functions in PROC FCMP only support positional parameters.
- User-defined functions in PROC FCMP do not support optional parameters.
- User-defined functions in PROC FCMP do not support default values for parameters.

Especially with CORE commands that have many options – like the CORE validate command – these limitations are problematic. Users would have to remember the exact order of the parameters and specify values that are common defaults. A solution for these issues is to wrap the functions in macros that can have named parameters and default values.

Turning CORE CLI Commands into PROC FCMP Functions

We saw in Figure 3 (Available CORE CLI commands) that the CORE CLI supports several commands. We want to be able to run these commands in SAS. To support this, we will:

- Create Python functions where the command options are function parameters
- Create PROC FCMP functions or subroutines that contain the Python functions
- Create macros that call the functions and subroutines. These macros can also do parameter checks that are not already done by the Python functions.

All CORE CLI commands were implemented (Table 1), except for the **test** command. The **test** command is used by CORE Rule developers. The scope for our implementation is the validation of data.

Table 1 CORE CLI commands implementation

CLI Command	Python Function	PROC FCMP SAS Function	SAS Macro
list-ct	core_list_ct	core_list_ct	core_list_ct
list-dataset-metadata	core_list_dataset_metadata	core_list_dataset_metadata	core_list_dataset_metadata
list-rule-sets	core_list_rule_sets	core_list_rule_sets	core_list_rule_sets
list-rules	core_list_rules	core_list_rules	core_list_rules
test	-	-	-
update-cache	core_update_cache	core_update_cache	core_update_cache
validate	core_validate_data	core_validate_data	core_validate_data
version	core_version	core_version	-

In the **cdisc-rules-engine** GitHub repository there is a Python file, core.py, that defines the CORE CLI.

The core.py file uses the Click package to create the command line interface [16].

Appendix 1 shows the Python code in core.py to implement the **validate** command. The Python package imports are not displayed.

To turn this code in a Python function that can be used by PROC FCMP we have to do several things:

- Wrap the code in a Python function, for example core_validate_data.
- Take out all code related to the Python Click package (@click decorators, ctx function argument).
- Replace ctx.exit() statements with return statements, adding return messages where needed.
- Add default values to the Python validate() function.
- Since SAS does not support Tuples, we need to convert a comma separated list into a Python supported datatype, like tuples and lists.
- Move parameter checks from the Click package to the SAS macro,for example %core_validate_data.
- The Python function called by SAS needs to know where to find the cdisc-rules-engine library. For this reason we need an operational system environment variable (CORE_PATH) that defines this location. Every Python function will have code added to find the cdisc-rules-engine library:

```
# Add top-level folder to path so that project folder can be found
core_path = os.environ["CORE_PATH"]
lib_path = os.path.abspath(os.path.join(__file__, core_path))
if lib_path not in sys.path: sys.path.append(lib_path)
```

Appendix 2 shows the Python function that is called by SAS PROC FCMP implement the **validate** command.

Now that the Python functions have been defined, they can be called by a SAS function in PROC FCMP. Below is the code used for the **core_version**, **core_data_validate**, and **core_update_cache** functions:

```
proc fcmp outlib = macros.core_funcs.python;

  function core_version() $ 32;
    length message $ 128;
    declare object py(python);
```

```
    submit into py("&project_folder/python/core_version.py");
    rc = py.publish();
    rc = py.call('core_version');
    message = py.results['message_return_value'];
    return(message);
  endfunc;

  function core_validate_data(
    cache $, pool_size, data $, dataset_path $, log_level $,
    report_template $,standard $, version $, output $,
    output_format $, raw_report, controlled_terminology_package $,
    define_version $, data_format $, define_xml_path $,whodrug $,
    meddra $, rules $) $ 128;
    length message $ 128;
    declare object py(python);
    submit into py("&project_folder/python/core_validate_data.py");
    rc = py.publish();
    rc = py.call('core_validate_data',
      cache, pool_size, data, dataset_path, log_level, report_template,
      standard, version, output, output_format, raw_report,
      controlled_terminology_package, define_version, data_format,
      define_xml_path, whodrug, meddra, rules);
    message = py.results['message_return_value'];
    return(message);
  endfunc;

  subroutine core_update_cache(apikey $, cache_path $);
    declare object py(python);
    submit into py("&project_folder/python/core_update_cache.py");
    rc = py.publish();
    rc = py.call('core_update_cache', apikey, cache_path);
  endsub;

  ...

run;
```

Creating Macros to Run CORE Commands

The last step is to create macros that call the PROC FCMP functions so that we can define named parameters and default parameter values.
Also, we can do additional parameter validation. Some of the validation checks:

- Does a required parameter have a value?
- Does a file exist?
- Does a folder exist?

Below is an example of a macro, in this case a partial listing of the **core_validate_data** SAS macro.

```
%macro core_validate_data(
  cache_path = %sysfunc(sysget(CORE_PATH))/resources/cache,
  pool_size = 10,
  data =,
  dataset_path =,
  log_level = disabled,
  report_template =
%sysfunc(sysget(CORE_PATH))/resources/templates/report-template.xlsx,
```

```
  standard =,
  version =,
  controlled_terminology_package =,
  output =,
  output_format = XLSX,
  raw_report = 0,
  define_version =,
  data_format = XPT,
  define_xml_path =,
  whodrug =,
  meddra =,
  rules =
  ) / minoperator;

  ...
%*******************************************************************;
%* Parameter checks
*;
%*******************************************************************;

...

  data _null_;
    message = core_validate_data("&cache_path", &pool_size, "&data",
"&dataset_path", "&log_level", "&report_template", "&standard",
"&version", "&output",  "&output_format",  &raw_report,
"&controlled_terminology_package" , "&define_version", "&data_format",
"&define_xml_path", "&whodrug", "&meddra", "&rules");
    if not missing(message) then putlog "ERR" "OR: " message;
  run;

  %exit_macro:

%mend core_validate_data;
```

Parameter descriptions can be found in the macro headers.

USING THE MACROS TO RUN CORE COMMANDS

This section gives examples on the use of the macros that implement the CORE commands.

For every macro there is an example program in programs folder in the GitHub repository (https://github.com/lexjansen/cdisc-core-sas).

Before running any of these example programs the user needs to run the **create_core_functions.sas** program in the **programs** folder to create the PROC FCMP functions dataset.

Every example program starts with the same lines of code:

```
%* This code assumes that your SAS environment can run Python objects. ;
%* Check the programs/config.sas file for Python configuration.           ;

%* update this macro variable to your own location;
%let project_folder = /_github/lexjansen/cdisc-core-sas;

%include "&project_folder/programs/config.sas";
```

Make sure to update the **project_folder** macro variable to your own location.

%core_update_cache

Purpose: get the latest set of rules and standards metadata from the CDISC Library.

```
%core_update_cache(
  /* apikey= <your API key>, */
  cache_path = &project_folder/resources/cache
  );
```

This macro call assumes that you have an environment variable `CDISC_LIBRARY_API_KEY`. If not, you can specify the API key in the macro call.

The result of this macro call is that the latest set of rules and standards metadata from the CDISC Library is extracted to the local cache folder (see Figure 4 CORE local cache folder with pickle files).

%core_list_ct

Purpose: list the Controlled Terminology packages available in the cache.

```
filename ct "&project_folder/json/core_ct.json";

%core_list_ct(
  subsets =,
  output =  %sysfunc(pathname(ct)),
  cache_path = &project_folder/resources/cache
  );

data _null_;
   rc = jsonpp('ct','log');
run;

libname jsonfile json fileref=ct;

data metadata.core_ct(keep=value rename=(value=ct_package));
  set jsonfile.alldata;
run;

filename ct clear;
libname jsonfile clear;
```

The macro extracts a JSON file with the available Controlled Terminology packages in the cache. The JSON file can be easily converted to a SAS dataset as the code demonstrates. The dataset has one variable (ct_package) with the Controlled Terminology package.

Figure 10 core_ct dataset with available Controlled Terminology packages

CORE_CT

	ct_package
129	sdtmct-2023-06-30
130	sdtmct-2023-09-29
131	sdtmct-2023-12-15
132	sendct-2014-09-26
133	sendct-2014-12-19
134	sendct-2015-03-27
135	sendct-2015-06-26
136	sendct-2015-09-25
137	sendct-2015-12-18
138	sendct-2016-03-25

%core_list_dataset_metadata

Purpose: list metadata of given datasets.

```
filename meta "&project_folder/json/core_dataset_metadata.json";

%core_list_dataset_metadata(
  dataset_path = %str
    (&project_folder/testdata/sdtm/dm.xpt,
     &project_folder/testdata/sdtm/ae.xpt,
     &project_folder/testdata/sdtm/ex.xpt,
     &project_folder/testdata/sdtm/lb.xpt),
    output =  %sysfunc(pathname(meta))
  );

data _null_;
   rc = jsonpp('meta','log');
run;

libname jsonfile json fileref=meta ordinalcount=none;

data metadata.core_dataset_metadata;
  set jsonfile.root;
run;

filename meta clear;
libname jsonfile clear;
```

The macro extracts a JSON file with the dataset metadata. The JSON file can be easily converted to a SAS dataset as the code demonstrates.

Figure 11 core_dataset_metadata with dataset metadata

CORE_DATASET_METADATA

	domain	filename	full_path	size	label	modification_date
1	DM	dm.xpt	/_github/lexjansen/cdisc-core-sas/testdata/sdtm/dm.xpt	13040	Demographics	2020-08-21T09:14:29
2	AE	ae.xpt	/_github/lexjansen/cdisc-core-sas/testdata/sdtm/ae.xpt	38080	Adverse Events	2020-08-21T09:14:28
3	EX	ex.xpt	/_github/lexjansen/cdisc-core-sas/testdata/sdtm/ex.xpt	823120	Exposure	2020-08-21T09:14:26
4	LB	lb.xpt	/_github/lexjansen/cdisc-core-sas/testdata/sdtm/lb.xpt	2763040	Laboratory Test Results	2020-08-21T09:14:29

%core_list_rule_sets

Purpose: list the rule sets available in the cache.

```
filename rulesets "&project_folder/json/core_rule_sets.json";

%core_list_rule_sets(
  output =  %sysfunc(pathname(rulesets)),
  cache_path = &project_folder/resources/cache
  );

data _null_;
   rc = jsonpp('rulesets','log');
run;

libname jsonfile json fileref=rulesets;

data metadata.core_rulesets(keep=standard version);
  length standard $32 version $16;
  set jsonfile.alldata;
  standard = strip(scan(value, 1, ','));
  version = strip(scan(value, 2, ','));
run;

proc sort data = metadata.core_rulesets;
  by standard version;
run;

filename rulesets clear;
libname jsonfile clear;
```

The macro extracts a JSON file with the available CORE rulesets in the cache. The JSON file can be easily converted to a SAS dataset as the code demonstrates. The dataset has two variables (standard and version) with the CORE rulesets.

Figure 12 core_rulesets dataset with available CORE rulesets

CORE_RULESETS

	standard	version
1	DDF	2
2	SDTMIG	3-2
3	SDTMIG	3-3
4	SDTMIG	3-4
5	SENDIG	3-0
6	SENDIG	3-1
7	SENDIG	3-1-1
8	SENDIG-DART	1-1
9	SENDIG-DART	1-2
10	SENDIG-GENETOX	1-0

%core_list_rules

Purpose: list the rules available in the cache.

```
filename rules "&json_folder/core_rules_&core_standard.-
&core_standard_version..json";

%core_list_rules(
  output =  %sysfunc(pathname(rules)),
  standard = &core_standard,
  version = &core_standard_version,
  cache_path = &project_folder/resources/cache
  );
```

The **core_list_rules** macro extracts a JSON file with the available CORE rules for a given standard and version in the cache. By utilizing the dataset with CORE rulesets, we can extract all rules, organized by standard. We also utilize a macro (**get_core_rules**) to turn the JSON file into a SAS dataset (Figure 13).

```
data _null_;
  set metadata.core_rulesets;
  length code $ 1024;
  if upcase(standard) = "DDF" then
  %* For DDF only get JSON ;
    code = cats('%nrstr(%get_core_rules(core_standard=',
lowcase(standard), ', core_standard_version=', version, ', dsout=));');
  else
    code = cats('%nrstr(%get_core_rules(core_standard=',
lowcase(standard), ', core_standard_version=', version, '));');
  put code=;
  call execute(code);
run;
```

This dataset can be used to select rules when validating data, based on standard, standard version, class, domain, and dataset.

Figure 13 core_rules dataset with available CORE rules

CORE_RULES

	core_standard	core_standard_version	core_id	author	sensitivity	executability	description	rule_type
509	sdtmig	3-4	CORE-000001	CDISC	Record	fully executable	Raise an error when IECAT="INCLUSION" and IEORRES^="N".	Record Data
510	sdtmig	3-4	CORE-000002	CDISC	Record	fully executable	Raise an error when SESTDTC is null.	Record Data
511	sdtmig	3-4	CORE-000003	CDISC	Dataset	fully executable	Raise an error when TRLOC is present.	Record Data
512	sdtmig	3-4	CORE-000004	CDISC	Record	fully executable	When ECOCCUR indicates no dose, then ECDOSE needs to be null or > 0; it cannot ...	Record Data
513	sdtmig	3-4	CORE-000005	CDISC	Record	fully executable	When EXTRT is PLACEBO, EXDOSE must equal 0	Record Data
514	sdtmig	3-4	CORE-000006	CDISC	Record	fully executable	Raise an error when DTHFL ^= "Y" and not null	Record Data
515	sdtmig	3-4	CORE-000007	CDISC	Record	fully executable	Raise an error when DTHDTC is not empty and DTHFL not equal to "Y"	Record Data
516	sdtmig	3-4	CORE-000008	CDISC	Record	fully executable	Raise an error when DTHFL not equal to "Y" and SS.SSSTRESC = "DEAD".	Record Data
517	sdtmig	3-4	CORE-000009	CDISC	Record	fully executable	Verify that ELEMENT value is blank when ETCD is equal to UNPLAN	Record Data
518	sdtmig	3-4	CORE-000010	CDISC	Record	fully executable	Verify ARMCD value length is <= 20	Record Data
519	sdtmig	3-4	CORE-000011	CDISC	Record	fully executable	Verify the value for IEORRES is Y	Record Data
520	sdtmig	3-4	CORE-000012	CDISC	Dataset	fully executable	Raise an error when AEOCCUR exists in AE dataset.	Record Data
521	sdtmig	3-4	CORE-000013	CDISC	Dataset	fully executable	Raise an error when AESTAT is present in AE dataset.	Record Data
522	sdtmig	3-4	CORE-000014	CDISC	Record	fully executable	Raise an error when --PRESP is not equals to "Y" and --OCCUR is present in dataset ...	Record Data
523	sdtmig	3-4	CORE-000015	CDISC	Dataset	fully executable	Raise an error when --PRESP does not exists in a dataset and --OCCUR exist.	Record Data
524	sdtmig	3-4	CORE-000016	CDISC	Dataset	fully executable	Raise an error when --OCCUR is not empty and --PRESP is not equal to "Y".	Record Data
525	sdtmig	3-4	CORE-000017	CDISC	Record	fully executable	Verify RDOMAIN is not null when IDVARVAL is not null	Record Data
526	sdtmig	3-4	CORE-000018	CDISC	Record	fully executable	Raise an error when --PRESP is equal to "Y", --STAT is blank, --OCCUR is present in ...	Record Data
527	sdtmig	3-4	CORE-000019	CDISC	Record	fully executable	Raise and error if Variable label length > 40 characters	Variable Metadata Check

	message	standards	classes_include	classes_exclude	domains_include	domains_exclude	datasets
509	IECAT equals "INCLUSION" and IEORRES is not equal to "N".	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	FINDINGS		IE		
510	SESTDTC is required.	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	SPECIAL PURPOSE		SE		
511	TRLOC is present when TRLOC is not included in the TR domain.	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	FINDINGS		TR		
512	ECOCCUR indicates dose was not given, but ECDOSE is not blank or has a value less tha...	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	INTERVENTIONS		EC		
513	EXTRT is PLACEBO, but EXDOSE is not equal to 0.	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	INTERVENTIONS		EX		
514	DTHFL is not "Y" or null	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	SPECIAL PURPOSE		DM		
515	DTHFL is not "Y", when DTHDTC is populated	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	SPECIAL PURPOSE		DM		
516	DTHFL is not "Y", when SS.SSSTRESC equals "DEAD"	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	FINDINGS		SS		DM
517	ELEMENT variable has a non-null value when ETCD has a value of 'UNPLAN'	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	SPECIAL PURPOSE		SE		
518	ARMCD value length is greater than 20	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	SPECIAL PURPOSE;TRIAL DESIGN		DM;TA		
519	IEORRES = N for an exclusion criteria. Only exclusion criteria not met should be included in...	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	FINDINGS		IE		
520	AEOCCUR is present in AE dataset.	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	EVENTS		AE		
521	AESTAT variable is present in AE dataset.	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	EVENTS		AE		
522	--OCCUR should only be provided when --PRESP is equal to "Y".	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	EVENTS;INTERVENTIONS			AE;DS;DV;EX	
523	--PRESP is missing in dataset when --OCCUR is present.	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	EVENTS;INTERVENTIONS			AE;DS;DV;EX	
524	--PRESP should be populated as "Y" when --OCCUR is provided.	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	EVENTS;INTERVENTIONS			AE;DS;DV;EX	
525	RDOMAIN has a missing value when IDVARVAL has a non-missing value	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	SPECIAL PURPOSE		CO		
526	--OCCUR is blank when --PRESP is equal to "Y" and --STAT is not provided.	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	EVENTS;INTERVENTIONS			AE;DS;DV;EX	
527	Variable label length should be less than or equal to 40 characters	SDTMIG 3.2;SDTMIG 3.3;SDTMIG 3.4	ALL		ALL		

%core_validate_data

Purpose: validate data.

```
%core_validate_data(
  cache_path = &project_folder/resources/cache,
  data= &project_folder/testdata/sdtm,
  standard = sdtmig,
  version = 3-3,
  controlled_terminology_package = %str(sdtmct-2023-12-15),
  output= &project_folder/reports/&report_name._sdtmig_3-3,
  output_format = %str(XLSX, JSON),
  raw_report = 0,
  data_format = XPT,
  define_xml_path = &project_folder/testdata/sdtm/define.xml,
  whodrug = &project_folder/testdata/dictionaries/whodrug,
  meddra = &project_folder/testdata/dictionaries/meddra,
  rules =
  );
```

In this example we validate all XPT files in the `&project_folder/testdata/sdtm` folder,and we are not limiting the rules (rules =). The result will be an Excel spreadsheet similar to the one in Figure 8 together with a JSON file that contains the same content as in the Excel spreadsheet.

Using the core_rules dataset from Figure 12, we can limit the validation to rules that are specific to datasets we want to validate. An example is below. A macro variable **core_rules** is created that contains the applicable rules. We use this macro variable in the **%core_validate_data** macro parameter:
`rules = "&core_rules"`

```
/* Example of selecting rules */
proc sql noprint;
  select distinct core_id into :core_rules separated by ','
  from metadata.core_rules
  where (domains_include in ('ALL' 'AE' 'DM')) and
        (domains_exclude ne 'DM') and
        (domains_exclude ne 'AE')and
        (core_standard = "sdtmig" and core_standard_version =  "3-3")
  order by core_id;
quit;

options noquotelenmax;
%core_validate_data(
  cache_path = &project_folder/resources/cache,
  dataset_path = %str
    (&project_folder/testdata/sdtm/dm.xpt,
     &project_folder/testdata/sdtm/ae.xpt),
  standard = sdtmig,
  version = 3-3,
  controlled_terminology_package = %str(sdtmct-2023-12-15),
  output= &project_folder/reports/&report_name._sdtmig_3-3,
  output_format = %str(XLSX, JSON),
  raw_report = 0,
  data_format = XPT,
  define_xml_path = &project_folder/testdata/sdtm/define.xml,
  whodrug = &project_folder/testdata/dictionaries/whodrug,
  meddra = &project_folder/testdata/dictionaries/meddra,
  rules = "&core_rules"
  );
```

The result will be an Excel spreadsheet similar to the one in Figure 8 together with a JSON file that contains the same content as in the Excel spreadsheet.

CONCLUSION

This paper shows that it is possible to implement CDISC CORE Engine CLI commands in the SAS environment. By creating PROC FCMP functions or subroutines that contain Python functions that implement CORE commands, SAS macros can call these Python functions from the DATA step. Named parameters and default values can be easily implemented in SAS macros.

REFERENCES

[1] CDISC CORE – Available at https://www.cdisc.org/core

[2] de Donder, Nick & Peter van Reusel. (2023). CDISC Conformance rules and the CDISC Open Rules Engine Continuing the Road to Adoption. Proceedings of PHUSE EU Connect 2023. Retrieved from https://phuse.s3.eu-central-1.amazonaws.com/Archive/2023/Connect/EU/Birmingham/PAP_SI04.pdf

[3] CDISC CORE Engine GitHub Repository. Available at https://github.com/cdisc-org/cdisc-rules-engine

[4] YAML: YAML Ain't Markup Language™. https://yaml.org/

[5] CDISC CORE Rules Editor GitHub Repository. Available at https://github.com/cdisc-org/conformance-rules-editor

[6] Wyckmans, Marisa & Els Janssen (2023). Developing and Implementing CDISC CORE rules: from zero to hero. Proceedings of PHUSE EU Connect 2023. Retrieved from: https://phuse.s3.eu-central-1.amazonaws.com/Archive/2023/Connect/EU/Birmingham/PAP_SI05.pdf

[7] CDISC CORE Engine GitHub Repository Releases. Available at https://github.com/cdisc-org/cdisc-rules-engine/releases

[8] cdisc-rules-engine on the Python Package Index (PyPI). Available at https://pypi.org/project/cdisc-rules-engine/

[9] pickle — Python object serialization. Retrieved from https://docs.python.org/3/library/pickle.html

[10] Getting Started: Access to CDISC Library API using API Key Authentication. https://wiki.cdisc.org/display/LIBSUPRT/Getting+Started%3A+Access+to+CDISC+Library+API+using+API+Key+Authentication

[11] Zizzi, Mike (2019). SAS or Python? Why not use both? Using Python functions inside SAS programs. Retrieved from: https://blogs.sas.com/content/sgf/2019/06/04/using-python-functions-inside-sas-programs/

[12] SAS Institute. "FCMP Procedure." *Base SAS Procedures Guide*. Available at https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/n0pio2crltpr35n1ny010zrfbvc9.htm

[13] SAS Institute. "Configuring SAS to Run the Python Language." Configuring SAS to Run External Languages. Available at https://go.documentation.sas.com/doc/en/bicdc/9.4/biasag/n1mquxnfmfu83en1if8icqmx8cdf.htm

[14] SAS Institute. "Using PROC FCMP Python Objects." *Base SAS Procedures Guide*. Available at https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lecompobjref/p18qp136f91aaqn1h54v3b6pkant.htm

[15] SAS Institute. "Python Object Language Elements." *Base SAS Procedures Guide*. Available at https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/lecompobjref/p0x7optrcqo9vmn119q1yhxpj35n.htm

[16] “click” on the Python Package Index (PyPI). Available at https://pypi.org/project/click/

CONTACT INFORMATION

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

Lex Jansen
Sr Director, Data Science Development, CDISC (contract through Lex Jansen Consulting LLC)
Email: ljansen@cdisc.org or lexjansen@gmail.com

APPENDICES

APPENDIX 1 : CODE IN THE CORE-RULES-ENGINE PYTHON FILE CORE.PY TO IMPLEMENT THE VALIDATE COMMAND

```
import asyncio
import json
import logging
import os
import pickle
from datetime import datetime
from multiprocessing import freeze_support
from typing import Iterable, Tuple

import click
from pathlib import Path
from cdisc_rules_engine.config import config
from cdisc_rules_engine.constants.define_xml_constants import DEFINE_XML_FILE_NAME
from cdisc_rules_engine.enums.default_file_paths import DefaultFilePaths
from cdisc_rules_engine.enums.progress_parameter_options import
ProgressParameterOptions
from cdisc_rules_engine.enums.report_types import ReportTypes
from cdisc_rules_engine.models.validation_args import Validation_args
from cdisc_rules_engine.models.test_args import TestArgs
from scripts.run_validation import run_validation
from scripts.test_rule import test as test_rule
from cdisc_rules_engine.services.cache.cache_populator_service import CachePopulator
from cdisc_rules_engine.services.cache.cache_service_factory import
CacheServiceFactory
from cdisc_rules_engine.services.cdisc_library_service import CDISCLibraryService
from cdisc_rules_engine.utilities.utils import (
    generate_report_filename,
    get_rules_cache_key,
)
from scripts.list_dataset_metadata_handler import list_dataset_metadata_handler
from version import __version__

def valid_data_file(file_name: str, data_format: str):
    fn = os.path.basename(file_name)
    return fn.lower() != DEFINE_XML_FILE_NAME and fn.lower().endswith(
        f".{data_format.lower()}"
    )

@click.group()
def cli():
    pass
```

```python
@click.command()
@click.option(
    "-ca",
    "--cache",
    default=DefaultFilePaths.CACHE.value,
    help="Relative path to cache files containing pre loaded metadata and rules",
)
@click.option(
    "-ps",
    "--pool-size",
    default=10,
    type=int,
    help="Number of parallel processes for validation",
)
@click.option(
    "-d",
    "--data",
    required=False,
    help="Path to directory containing data files",
)
@click.option(
    "-dp",
    "--dataset-path",
    required=False,
    multiple=True,
    help="Absolute path to dataset file",
)
@click.option(
    "-l",
    "--log-level",
    default="disabled",
    type=click.Choice(["info", "debug", "error", "critical", "disabled", "warn"]),
    help="Sets log level for engine logs, logs are disabled by default",
)
@click.option(
    "-rt",
    "--report-template",
    default=DefaultFilePaths.EXCEL_TEMPLATE_FILE.value,
    help="File path of report template to use for excel output",
)
@click.option(
    "-s", "--standard", required=True, help="CDISC standard to validate against"
)
```

```python
@click.option(
    "-v", "--version", required=True, help="Standard version to validate against"
)
@click.option(
    "-ct",
    "--controlled-terminology-package",
    multiple=True,
    help=(
        "Controlled terminology package to validate against, "
        "can provide more than one"
    ),
)
@click.option(
    "-o",
    "--output",
    default=generate_report_filename(datetime.now().isoformat()),
    help="Report output file destination",
)
@click.option(
    "-of",
    "--output-format",
    multiple=True,
    default=[ReportTypes.XLSX.value],
    type=click.Choice(ReportTypes.values(), case_sensitive=False),
    help="Output file format",
)
@click.option(
    "-rr",
    "--raw-report",
    default=False,
    show_default=True,
    is_flag=True,
    help="Report in a raw format as it is generated by the engine. "
    "This flag must be used only with --output-format JSON.",
)
@click.option(
    "-dv",
    "--define-version",
    help="Define-XML version used for validation",
)
@click.option(
    "-df",
    "--data-format",
    help="Format in which data files are presented. Defaults to XPT.",
    type=click.Choice(["xpt"], case_sensitive=False),
```

```
    default="xpt",
    required=True,
)
@click.option("--whodrug", help="Path to directory with WHODrug dictionary files")
@click.option("--meddra", help="Path to directory with MedDRA dictionary files")
@click.option("--rules", "-r", multiple=True)
@click.option(
    "-p",
    "--progress",
    default=ProgressParameterOptions.BAR.value,
    type=click.Choice(ProgressParameterOptions.values()),
    help=(
        "Defines how to display the validation progress. "
        'By default a progress bar like "[████████████████████████████--------
]   78%"'
        "is printed."
    ),
)
@click.option("-dxp", "--define-xml-path", required=False, help="Path to Define-XML")
@click.pass_context
def validate(
    ctx,
    cache: str,
    pool_size: int,
    data: str,
    dataset_path: Tuple[str],
    log_level: str,
    report_template: str,
    standard: str,
    version: str,
    controlled_terminology_package: Tuple[str],
    output: str,
    output_format: Tuple[str],
    raw_report: bool,
    define_version: str,
    data_format: str,
    whodrug: str,
    meddra: str,
    rules: Tuple[str],
    progress: str,
    define_xml_path: str,
):
    """
    Validate data using CDISC Rules Engine
```

```
    Example:

    python core.py -s SDTM -v 3.4 -d /path/to/datasets
    """

    # Validate conditional options
    logger = logging.getLogger("validator")

    if raw_report is True:
        if not (len(output_format) == 1 and output_format[0] ==
ReportTypes.JSON.value):
            logger.error(
                "Flag --raw-report can be used only when --output-format is JSON"
            )
            ctx.exit()

    cache_path: str = os.path.join(os.path.dirname(__file__), cache)

    if data:
        if dataset_path:
            logger.error(
                "Argument --dataset-path cannot be used together with argument --
data"
            )
            ctx.exit()
        dataset_paths: Iterable[str] = [
            str(Path(data).joinpath(fn))
            for fn in os.listdir(data)
            if valid_data_file(fn, data_format)
        ]
    elif dataset_path:
        if data:
            logger.error(
                "Argument --dataset-path cannot be used together with argument --
data"
            )
            ctx.exit()
        dataset_paths: Iterable[str] = [
            dp for dp in dataset_path if valid_data_file(dp, data_format)
        ]
    else:
        logger.error(
            "You must pass one of the following arguments: --dataset-path, --data"
        )
        # no need to define dataset_paths here, the program execution will stop
```

```
    ctx.exit()

run_validation(
    Validation_args(
        cache_path,
        pool_size,
        dataset_paths,
        log_level,
        report_template,
        standard,
        version,
        set(controlled_terminology_package),  # avoiding duplicates
        output,
        set(output_format),  # avoiding duplicates
        raw_report,
        define_version,
        data_format.lower(),
        whodrug,
        meddra,
        rules,
        progress,
        define_xml_path,
    )
)
```

APPENDIX 2 : PYTHON FUNCTION CALLED BY SAS PROC FCMP TO IMPLEMENT THE VALIDATE COMMAND

```
def core_validate_data(cache, pool_size, data, dataset_path, log_level,
report_template, standard, version, output, output_format, raw_report,
controlled_terminology_package, define_version, data_format, define_xml_path, whodrug,
meddra, rules):

      """Output: message_return_value"""

      import os
      import sys

      # Add top-level folder to path so that project folder can be found
      core_path = os.environ["CORE_PATH"]
      lib_path = os.path.abspath(os.path.join(__file__, core_path))
      if lib_path not in sys.path: sys.path.append(lib_path)

      import asyncio
      import json
      import logging
      import os
      import pickle
      from datetime import datetime
      from multiprocessing import freeze_support
      from typing import Tuple
      import re

      from pathlib import Path
      from cdisc_rules_engine.config import config
      from cdisc_rules_engine.constants.define_xml_constants import
DEFINE_XML_FILE_NAME
      from cdisc_rules_engine.enums.default_file_paths import DefaultFilePaths
      from cdisc_rules_engine.enums.progress_parameter_options import
ProgressParameterOptions
      from cdisc_rules_engine.enums.report_types import ReportTypes
      from cdisc_rules_engine.models.validation_args import Validation_args
      from scripts.run_validation import run_validation
      from cdisc_rules_engine.services.cache.cache_populator_service import
CachePopulator
      from cdisc_rules_engine.services.cache.cache_service_factory import
CacheServiceFactory
      from cdisc_rules_engine.services.cdisc_library_service import
CDISCLibraryService
      from cdisc_rules_engine.utilities.utils import (
```

```
        generate_report_filename,
        get_rules_cache_key,
    )
    from scripts.list_dataset_metadata_handler import list_dataset_metadata_handler
    from version import __version__

    def valid_data_file(file_name: str, data_format: str):
        fn = os.path.basename(file_name)
        return fn.lower() != DEFINE_XML_FILE_NAME and fn.lower().endswith(
            f".{data_format.lower()}"
        )

    def validate(
        standard: str,
        version: str,
        cache: str = core_path + "/" + DefaultFilePaths.CACHE.value,
        pool_size: int =10,
        log_level: str = 'disabled',
        data: str = '',
        dataset_path: Tuple[str] =[],
        report_template: str = core_path + "/" +
DefaultFilePaths.EXCEL_TEMPLATE_FILE.value,
        output_format: Tuple[str] = [ReportTypes.XLSX.value],
        raw_report: bool = True,
        output: str = generate_report_filename(datetime.now().isoformat()),
        controlled_terminology_package: Tuple[str] = [],
        define_version: str = '',
        data_format: str = "XPT",
        rules: Tuple[str] = [],
        define_xml_path: str = '',
        whodrug: str ='',
        meddra: str = '',
        progress: str = 'disabled'
    ):
        """
        Validate data using CDISC Rules Engine

        Example:

        python core.py -s SDTM -v 3.4 -d /path/to/datasets
        """

        validation_message = ""

        dataset_path = [item.strip(' ') for item in dataset_path if item !='']
```

```
          output_format = [item.strip(' ') for item in output_format if item !='']
          controlled_terminology_package = [item.strip(' ') for item in
controlled_terminology_package if item !='']
          rules = [item.strip(' ') for item in rules if item !='']


          if not log_level:
              log_level = 'disabled'

          # Validate conditional options
          logger = logging.getLogger("validator")

          if raw_report is True:
              if not (len(output_format) == 1 and output_format[0] ==
ReportTypes.JSON.value):
                  logger.error(
                      "Flag --raw-report can be used only when --output-format is
JSON"
                  )
                  validation_message = "Flag --raw-report can be used only when --
output-format is JSON"
                  return validation_message

          cache_path: str = os.path.join(os.path.dirname(__file__), cache)

          if data:
              if dataset_path:
                  logger.error(
                      "Argument --dataset-path cannot be used together with argument -
-data"
                  )
                  validation_message = "Argument --dataset-path cannot be used
together with argument --data"
                  return validation_message
              dataset_paths: Iterable[str] = [
                  str(Path(data).joinpath(fn))
                  for fn in os.listdir(data)
                  if valid_data_file(fn, data_format)
              ]
          elif dataset_path:
              if data:
                  logger.error(
                      "Argument --dataset-path cannot be used together with argument -
-data"
                  )
```

```
                validation_message = "Argument --dataset-path cannot be used
together with argument --data"
                return validation_message
            dataset_paths: Iterable[str] = [
                dp for dp in dataset_path if valid_data_file(dp, data_format)
            ]
        else:
            logger.error(
                "You must pass one of the following arguments: --dataset-path, --
data"
            )
            validation_message = "You must pass one of the following arguments: --
dataset-path, --data"
            # no need to define dataset_paths here, the program execution will stop
            return validation_message

        run_validation(
            Validation_args(
                cache_path,
                pool_size,
                dataset_paths,
                log_level,
                report_template,
                standard,
                version,
                set(controlled_terminology_package),  # avoiding duplicates
                output,
                set(output_format),  # avoiding duplicates
                raw_report,
                define_version,
                data_format.lower(),
                whodrug,
                meddra,
                rules,
                progress,
                define_xml_path,
            )
        )

        return validation_message

    return_message = validate(
         cache=cache,
         pool_size=int(pool_size),
         data=data,
```

```
            dataset_path=re.split(';|,', dataset_path),
            log_level=log_level,
            report_template=report_template,
            standard=standard,
            version=version,
            output=output,
            output_format=re.split(';|,', output_format),
            raw_report=(raw_report == 1),
            controlled_terminology_package=re.split(';|,',
controlled_terminology_package),
            define_version=define_version,
            data_format=data_format,
            whodrug=whodrug,
            meddra=meddra,
            rules=re.split(';|,', rules)
        )

    return return_message
```