Controlling OpenCDISC using R

Martin Gregory
PhUSE 2012
Budapest, October 2012
Introduction

OpenCDISC Validator is a project of OpenCDISC

- Java application
- checks compliance with rules defined by CDISC
- provides a graphical user interface (GUI)
- provides a command line interface (CLI)
- allows user extensions to the rules
We are concerned with:

- running the validator in a production environment
- automating the running of the validator

We will not cover:

- interpreting the results
- evaluating the implementation of the CDISC rules
Features of the validator GUI

1. Validate Data
2. Generate Define.xml
3. ADaM
4. SAS® Transport (XPORT)
5. config-adam-1.0.xml
6. /data/projects/opencdisc_v1.0/test.adam/define.xml

Optional

Report Format: Excel

Progress:
Waiting to begin.
Features of the validator CLI

The Command Line Interface (CLI) allows control of:

- whether a validation or generation task is to be run
- which type of standard to use (SDTM, ADaM, Define.xml, SEND, Custom);
- the location of the files to be validated;
- the OpenCDISC configuration file, define file and codelists to use; and
- the location and format of the report to produce.

Full details of the options supported can be obtained by executing the command

```
java -jar .../lib/validator-cli-1.3.jar -help
```
Limitations of the GUI

![Image of OpenCDISC Validator interface]

- **What would you like to do?**
  - Validate Data
  - Generate Define.xml

- **Standard:** ADaM

- **Source Format:** SAS® Transport (XPORT)

- **Source Data:**
  - /data/projects/opencdisc_v1.0/test.adam/adqsadas.xpt
  - /data/projects/opencdisc_v1.0/test.adam/advs.xpt
  - /data/projects/opencdisc_v1.0/test.adam/adqsnpix.xpt
  - /data/projects/opencdisc_v1.0/test.adam/adlbh.xpt
  - /data/projects/opencdisc_v1.0/test.adam/adae.xpt
  - /data/projects/opencdisc_v1.0/test.adam/adtte.xpt
  - /data/projects/opencdisc_v1.0/test.adam/adsl.xpt

  You can select multiple files or folders as sources.

- **Configuration:** config-adam-1.0.xml

- **Define.xml:**
  - /data/projects/opencdisc_v1.0/test.adam/define.xml

  Optional

- **Report Format:** Excel
  - Report Settings

- **Progress:**
  - Waiting to begin.
Limitations of the CLI

- *all or nothing* source specification
- commands are very long
- for a given version, paths and some options are fixed
- a bare directory name cannot be specified
Specify calls in a parameter file

- one line per call
- only variable information in the file:
  - files to validate allowing relative paths
  - OpenCDISC validator rules (configuration file) to use
  - name and location of report
  - report type
  - optionally, the version of the validator to use
- answers all the limitations:
  - inputs identified once
  - selection of files can be processed by specifying a call per file
  - only required information need be specified
  - define.xml used if present
Specify calls in a parameter file

Sample parameter file

```plaintext
# Fields separated by white space:
# source config report-location report-type [version]

sdtm/ae.xpt  sdtm-3.1.2  report/sdtm_ae.xls  Excel
sdtm/dm.xpt  sdtm-3.1.2  report/sdtm_dm.xls  Excel 1.3
adam/*.xpt   adam-1.0   report/adam.xls      Excel 1.2.1
adam/*.xpt   ~config/my-adam-1.0.xml  report/my-adam.csv  Csv
```
Use function/script to read the parameter file

Parameter file

- call 1
- call 2
- ... 
- call N

runocdv()

parse & check

call options "array"

execute call 1

execute call N

Outputs

- report 1
- report N
- log file
Features required in the scripting language

- independent of operating system
- good operating system interface to
  - run programs and collect their output
  - manage files
- anonymous, arbitrary and dynamic data structures
- support regular expressions
- hide operating system details
Scripting language candidates

- Perl
  - satisfies all conditions, but
  - implementations for Microsoft Windows differ
- Python
  - satisfies all conditions,
  - same implementation on each OS
  - does not require Python to be installed
- R
  - satisfies all conditions,
  - same implementation on each OS
  - more likely to be already available
  - support for validated environments
R function runocdv()

runocdv() signature

```r
runocdv(file,
    ocd.path=c("1.3"="/opt/OpenCDISC/V1.3",
                "1.2.1"="/opt/OpenCDISC/V1.2.1"),
    ocd.def.ver="1.3",
    ocd.java="/usr/lib/java/bin/java",
    log=NA
)
```
Function runocdv() can only be called from R
We use an R script `call.runocdv.r` which can be called from the OS

- For UNIX-like OS, the path to Rscript is included in the definition of the script:
  ```bash
  #! /usr/local/bin/Rscript
  ```
  and so can be called as
  ```bash
  call.runocdv.r file
  ```
- On Microsoft Windows it may be necessary to call `Rscript` explicitly:
  ```bash
  \path-to-R\bin\Rscript call.runocdv.r file
  ```
Sample log file

Source: /data/projects/runocdv-v1.0/sdtm/ae.xpt
Define: /data/projects/runocdv-v1.0/sdtm/define.xml
Standard: /opt/OpenCDISC/V1.3/config/config-sdtm-3.1.2.xml
Report: /data/projects/runocdv-v1.0/report/sdtm-ae.xls
Type: Excel
OpenCDISC: 1.3

OpenCDISC Validator Messages

Beginning validation, please wait...
The validation has completed.

Log: no log was created.
Details: Overall flow of the function

Parameter file
- call 1
- call 2
- ...
- call N

runocdv()
- parse & check
- call options "array"
- execute call 1
- ...
- execute call N

Outputs
- report 1
- report N
- log file
Overall flow of the function

- initialise
- read the parameter file
- check the parameters and construct 2-D arrays for:
  - command arguments
  - messages
- execute commands writing messages to the log
**Initialisation**

- check parameter file was specified and exists
- set up some vectors of valid values
- initialise the log file and write the header

```r
writeLog(c(paste("NOTE: runocdv.r Revision: 1.0 on server ",
                     Sys.info()[["nodename"]],
                     paste("NOTE: processing job list",
                           basename(ocdfile),"at",Sys.time(),""))
```

which produces

**NOTE: runocdv.r Revision: 1.0 on server green**
**NOTE: processing job list src2_1.3.ocd at 2012-07-29 13:44:53**
We want the input data as a list of vectors:

<table>
<thead>
<tr>
<th>Vector element</th>
<th>List element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call 1</td>
<td>Call 2</td>
</tr>
<tr>
<td>source</td>
<td>sdtm/ae.xpt</td>
</tr>
<tr>
<td>config</td>
<td>sdtm-3.1.2</td>
</tr>
<tr>
<td>report</td>
<td>report/ae.xls</td>
</tr>
<tr>
<td>report type</td>
<td>Excel</td>
</tr>
<tr>
<td>version</td>
<td>1.3</td>
</tr>
</tbody>
</table>

```r
fid <- file(ocdfile, "r")
fid.contents <- readLines(fid,-1,TRUE)
fid.jobs <- fid.contents[!grepl("^([[:space:]]*#|[^[:space:]]*$)", fid.contents)]
ocd.param <- strsplit(fid.jobs,"[[:space:]]")
```
Constructing the command arguments

For each call, i.e. element of ocd.params we check that:

- at least one source file exists:
  ```r
  if (length(Sys.glob(ocd.param[[i]][1]))==0) {
    # write error message
  } else {...}
  ```

- the configuration file exists

- the report directory exists:
  ```r
  if (!file.exists(dirname(ocd.param[[i]][3]))) {
    # write error message and stop processing this job
  } else {...}
  ```
Forming canonical paths

The `normalisePath()` function constructs a canonically correct path for the operating system.
Here we construct the java command specifying the appropriate jar file:

```r
ocd.args[[i]][5] <-
paste(ocd.java,'-jar',
    normalizePath(paste(ocd.path[ocd.param[[i]][5]],
        "/lib/validator-cli-",
        ocd.param[[i]][5],'.jar',sep=""),
        mustWork=FALSE),
    sep=' ')```
Executing the validator

- The `system()` function executes a command using the user’s default shell (UNIX) or directly (Microsoft Windows)
- `intern=TRUE` captures stdout and stderr to character vector
- `ignore.stderr=FALSE` is default
  
  `ocd.jcmd.out <- system(ocd.java.cmd,intern=TRUE)`
- and write the messages to the log file

  `writeLog(ocd.jcmd.out)`
runocdv()

- allows repeated runs
- avoids error-prone manual selections
- enables batch processing
- saves reports where you want them
- handles multiple versions of the validator
- is independent of operating system

Additionally, R is eminently suitable as a scripting language

Note: Code available on the PhUSE Wiki