Merck Serono
Living science, transforming lives

MERCK

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

Introduction

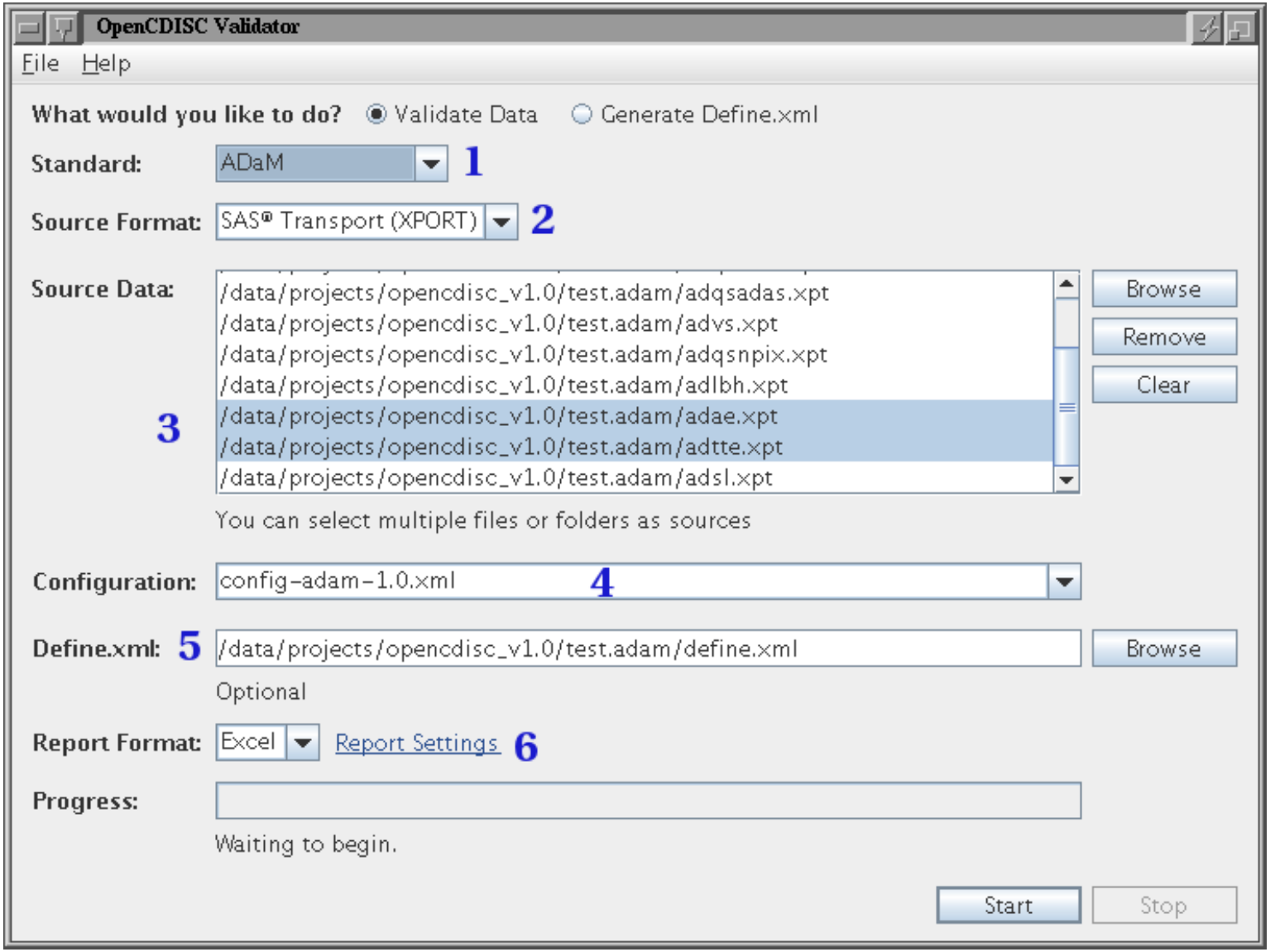
We are concerned with:

- 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



Features of the validator CLI

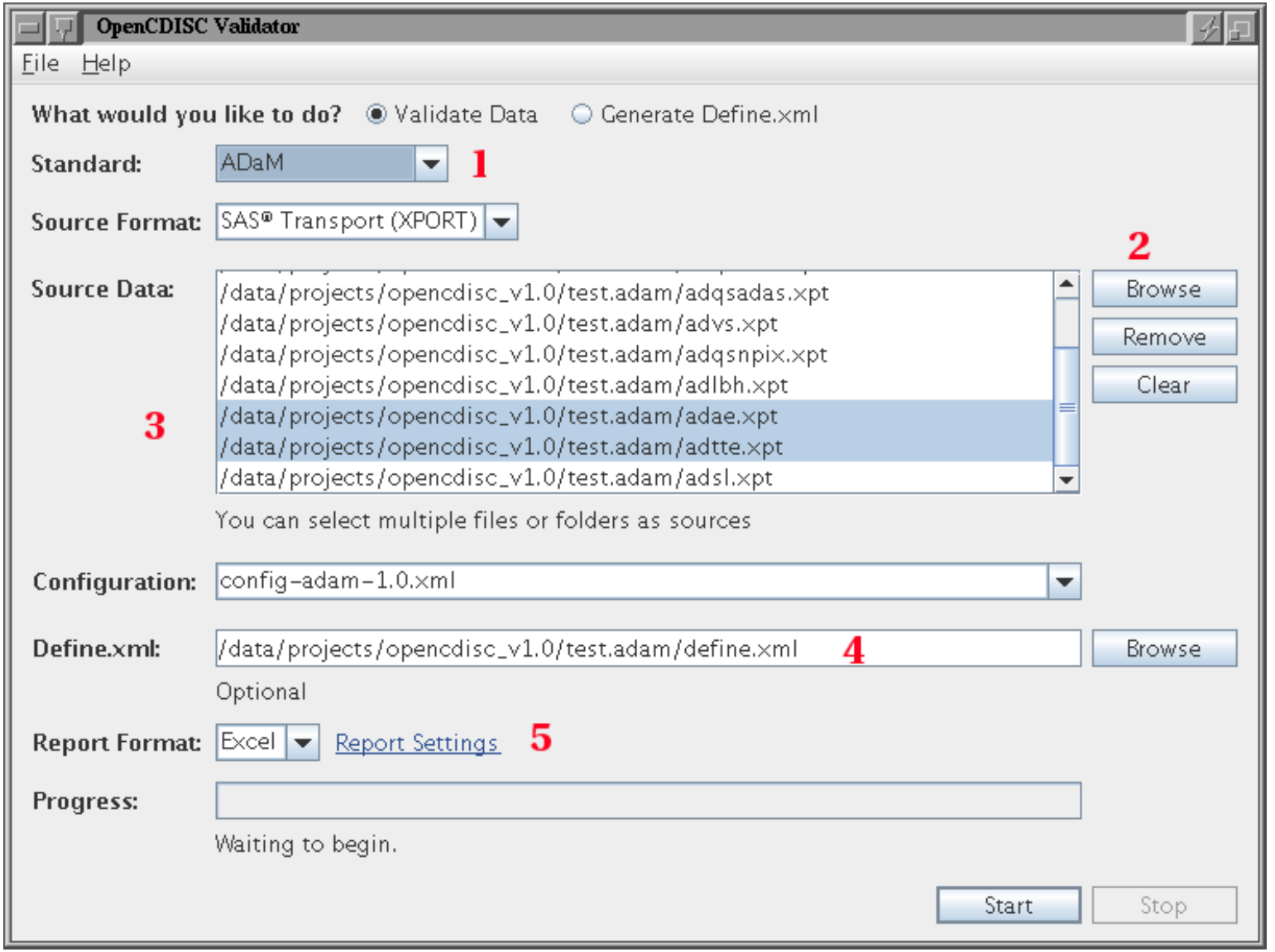
The Command Line Interface (CLI) allows control of:

- 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



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

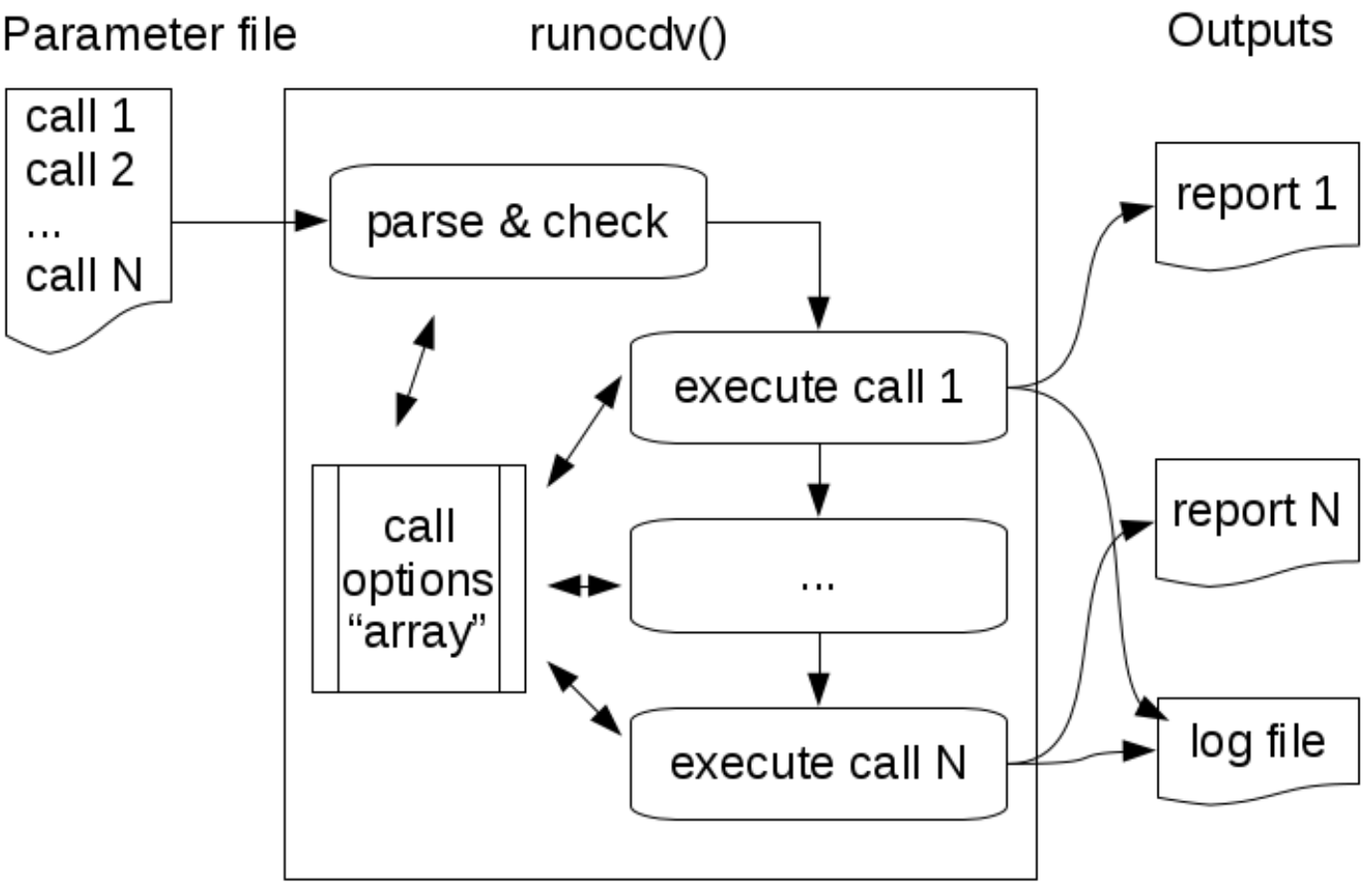
Specify calls in a parameter file

Sample parameter file

```
# 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



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

```
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
        )
```

R Script call.runocdv.r

- 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:

    ```
    #! /usr/local/bin/Rscript
    ```

    and so can be called as

    ```
    call.runocdv.r file
    ```
  - On Microsoft Windows it may be necessary to call *Rscript* explicitly:

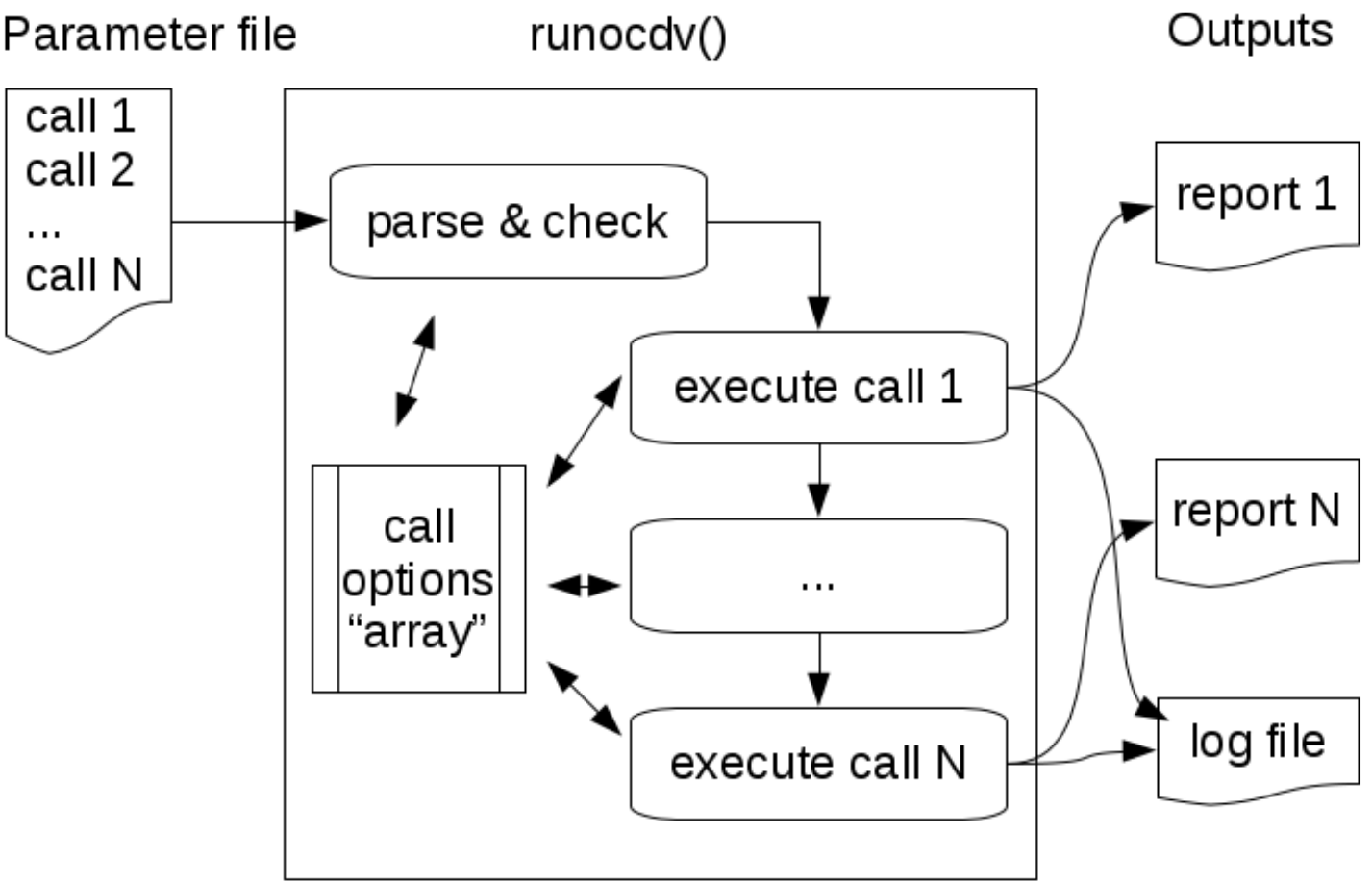
    ```
    \path-to-R\bin\Rscript call.runocdv.r file
    ```

runocdv() log 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



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

```
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
```

Reading the parameter file

We want the input data as a list of vectors:

Vector element	List element		
	Call 1	Call 2	Call 3
source	sdtm/ae.xpt	sdtm/dm.xpt	adam/*.xpt
config	sdtm-3.1.2	sdtm-3.1.2	adam-1.0
report	report/sdtm-ae.xls	report/sdtm-dm.xls	report/adam.xls
report type	Excel	Excel	Excel
version	1.3	1.3	1.2.1

```
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:

```
if (length(Sys.glob(ocd.param[[i]][1]))==0) {
  # write error message
} else {...}
```

- the configuration file exists

- the report directory exists:

```
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:

```
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)
```

Conclusion

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