

# QCS command line options

## Introduction

When running the QCS from the command line (DOS prompt, Linux shell etc.) some run time **arguments** can be specified. Since the application can be started in different ways (e.g., execution of a Java command, double click on a BAT file, custom run configuration within an IDE etc.), this document intentionally ignores *how* the application is run. We will adopt following notation:

- **RUN**: the literal "RUN" represents the generic command triggering execution of the application, which depends by the user platform, methodology or IDE.

*Example*: an actual "RUN command" could be something like

### Example of run command

```
java -jar -Xmx2g jrc-qcs-2.3.jar %1 %2
```

or

### Example of run command

```
call mvn exec:java -Dexec.args="%1 %2"
```

Where the "%1 %2" are just placeholders for the arguments (see below), that is: they are **not** part of the "RUN" command.

## Basic commands

The QCS can be run in different modes. For the time being two main execution modes are available:

- **GUI** (standard execution): Opens the main window and waits for user's actions
- **Background**: Runs in background and validates the dataset passed as argument. This usually means running the application from the command line

Execution mode can be selected by passing a proper argument to the RUN command:

- **RUN -help**: Lists all commands line options
- **RUN -gui**: Executes the application in normal mode (main GUI)
- **RUN -v <protocol ID> <dataset path>**: Validates the dataset identified by the path argument against the target protocol

```
Windows PowerShell
14:24:48 DatabaseDAOFactory - INFO : init() - Initialization successful
usage: help
-d,--daemon <protocolId path_in path_out time test> Perform validation
on the dataset
located in target
path. Can be run
also in bulk mode
-ef,--export_files <protocolId> Export the list of
all configuration
files
-fi,--field_info <fieldId> Shows all
information about
target field
-g,--gui <comment> Open user GUI
-h,--help Print available
options
-p,--protocol <protocolId> Shows all rules in
target protocol
-rf,--rules_by_field <fieldId protocolId> Shows all rules
addressing a target
field (for a given
protocol)
-s,--schema <schemaID> Load target schema
data
-t,--test <target> Executes embedded
test units.
Use "-t -a" to list
all test commands
(or --test --aid)
-v,--validate <protocolId file> Perform validation
on the target file.
Default values are
defined in the
"application.properties" file
```

Figure 1: Example of the "-help" command

The **protocol ID** argument specifies the protocol to be used to validate the input dataset. For example, these are the protocol supporting the ENCR 2022 data call.

Protocol ID	ENCR protocol
11	Incidence protocol (39 var)
13	Mortality protocol "Age Unit" (7 var)
14	Mortality protocol "Age Range" (7 var)
15	Population protocol "Age Unit" (6 var)
16	Population protocol "Age Range" (6 var)
17	Life Table protocol (6 var)

Examples:

- RUN -v 11: Validates the default dataset against the ENCR 2022 Incidence protocol (specified in the *application.properties* file)
- RUN -v 11 C:/Users/Home/input/encr/incidence/my\_datataset.csv: Validates the file *my\_datataset.csv* against the ENCR 2022 *Incidence* protocol (ID = 11)
- RUN -v 17 C:/Users/Home/input/encr/incidence/my\_lifetable.csv: Validates the file *my\_lifetable.csv* against the ENCR 2022 *Life Table* protocol (ID = 17)

## Advanced commands

Some advanced commands allows to run the QCS in background, query the application's inner configuration (introspection) and trigger some basic unit tests.

### Running in background

#### Running QCS in background

RUN -d <protocol ID> <path in> <path out> <time> <test suite>: Run QCS in background (daemon or service)

where:

- **protocol ID**: The ID of the protocol to be applied (same as above)
- **path in**: The absolute path where dataset can be "dropped" (that is: placed in the validation queue)
- **path out**: The absolute path where output reports are generated
- **time**: The time used when polling the path in directory (that is: the latency between dropping a file and starting validation)
- **test**: If *true*, the output report are created with respect of the name convention required by the test suite (default = *false*)

Example:

- **RUN -d 11 /home/user/my\_input /home/user/my\_output 10**: Executes the QCS in background mode, checking the *my\_input* folder every 10 seconds, and placing outreports in *my\_output* folder (Incidence protocol)

**Remark**: When running as a background service, the QCS uses a set of *traffic light signals* on the file system. For further details, see the "Running in background" document.

## Introspection

Some advanced commands allows to retrieve information about configuration of schemas, variables, rules and protocols.

- **RUN -p <protocol ID>**: Prints the list of rules in target **protocol**
- **RUN -s <schema ID>**: Prints the list of variables in target **schema**
- **RUN -fi <field ID>**: Prints the table providing **field** information about a single variable, such as: *name*, *description*, *datatype*, *maximum size*, *range ID* etc.
- **RUN -rf <field ID><protocol ID>**: Prints the list of rules addressing the target **field** (*field ID*) in the target **protocol** (*protocol ID*)
- **RUN -ef**: Exports relative path and name of all configuration files (i.e. the files containing configuration of all validation rules)

Examples:

- **RUN -s 11**: Print on screen the list of *variables* contained in the ENCR Incidence protocol for the 2022 data call (*schema ID* = 11)
- **RUN -p 11**: Print on screen the list of *rules* contained in the ENCR Incidence protocol for the 2022 data call (*protocol ID* = 11)
- **RUN -fi 80**: Print on screen information about the *Topo* variable (aka Topography), since it has *field ID* = 80
- **RUN -rf 81 11**: Print on screen the list rules checking the Morphology variable (field ID = 81) in the Incidence 2022 protocol (*protocol ID* = 11)

List of VALID rules in protocol (46 rules)

NICE CLASS	RID	CODE	SID	TYPE (LEVEL)	SCOPE
1 -> FileSizeRule	20	C-SIZE	11	FILE_FORMAT (0)	file_format
2 -> BunchDuplicatesRule	31	E-DUPL	11	DUPLICATES (0)	all_records
3 -> PositionalHeaderRule	40	E-HEAD	11	HEADER (1)	header
4 -> RecordFormatRule	50	E-RECO	11	RECORD_FORMAT (2)	record
4 -> DataTypeRule	60	E-FORM	11	FIELD_FORMAT (3)	record
4 -> MandatoryRule	65	E-MISS	11	FIELD_FORMAT (3)	record
4 -> MaxSizeRule	66	E-FORM	11	FIELD_FORMAT (3)	record
4 -> UnknownValueRule	67	W-UNKN	11	FIELD_FORMAT (3)	record
-> 87, 20 -> 88, 4 -> 73					
4 -> ImportantRule	68	W-MISS	11	FIELD_FORMAT (3)	record
5 -> RangeRule	80	E-OUTR	11	RANGE (4)	record
6 -> TNMEditionRule2	319	W-TNME	11	CROSS_FIELD (5)	record

Figure 2: Example of the "-p 11" command (Incidence 2022 protocol)

## Unit test

The **-t** argument allows to perform specific low leve unit tests. This feature is meant to be used only when developing and/or testing, and should **not** be used without a prior knowledge of the application.

Executing the *test* option using **--aid** as second argument provides the list of all available test features:

- **RUN -t --check <schemalD>**: Checks coherence of configuration for validation schemas. If *schemalD* is missing or not found, then all schemas are checked (see example below)
- **RUN -t --engine <protocolID>**: Executes the main validation engine on the default dataset (defined in the configuration files)
- **RUN -t --file**: Checks the low-level file handlers (used to read/write CSV and TXT files)
- **RUN -t --read <name>**: Reads data from a generic DAO concrete class, where *name* can be: *all*, *logger*, *properties*, *i18n*, *options*, *group*, *schema*, *field*, *schematofield*, *range*, *rangedata*, *schemaview*, *data*, *protocol*, *rule*, *protocoltorule*, *ruletofield*, *message*, *protocolview*, *run*, *output*, *detail*, *reference*
- **RUN -t --range <from to>**: Tests the *RangeProducer* class, which produces a alphanumeric range from the range's ends (*from* and *to*). Usage example: *RUN -t -range 1 10* should print all values in the the range [1..10]
- **RUN -t --sorter**: Checks the external sorter utility (used by the Primary Duplicates rules)
- **RUN -t --try\_rule <ruleID>**: Tests a single validation rule. If *ruleID* is omitted, the default rule (hard-coded in the EngineTester class) is executed. If *ruleID* = 0, then the *myRule.Java* class is tested (as hard-coded in the *MainTester* class)

- `RUN -t --write <name>`: Writes some test data using a legacy DAO, where *name* can be: *data*, *run*, *output*, *detail*, *custom*

**Warning:** Some of these features may be **deprecated**, because they were used to browse the QCS legacy configuration, which exploited as persistence layer a set of flat files. In 2025 the persistence layer was migrated to a SQLite database, therefore some of these features are not necessary anymore and /or could provide incorrect information. However, since implementation of these features contains information about how to validate the persistence layer, they are still provided in order to allow migration also of these features.

*Example:*

The *check coherence of validation schemas* feature was used to verify that the definition of all variables is coherent. For instance, if *my\_variable* is a variable defined with *max\_size* = 5 (maximum allowed number of chars: 5), and the range of allowed values *my\_variable* contains the string "123456", then the QCS cannot provide reliable results. This because the *OutOfRange* rule would tell that the value "123456" is correct, while the *MaxSize* rule would provide an E-FORM error, since it contains more than 5 chars.

This is not a bad design nor a but, but it's the price implied by a flexible architecture: since it's possible to update the QCS's configuration to update variables and rules without changing the code, the application must validate the configuration *before* validating the dataset. With the migration from the flat file architecture to the database oriented architecture many of this "coherence check" should be not necessary anymore, but completing this task is still a work in progress.