QA Catalogue

Péter Király

2025-02-11

Table of contents

Introduction

QA catalogue is a set of software packages for bibliographical record quality assessment. It reads MARC or PICA files (in different formats), analyses some quality dimensions, and saves the results into CSV files. These CSV files could be used in different context, we provide a lightweight, web-based user interface for that. Some of the functionalities are available as a web service, so the validation could be built into a cataloguing/quality assessment workflow.

Figure 1: QA Catalogue's web user interface

- For more info
  - main project page: Metadata Quality Assessment Framework
  - Validating 126 million MARC records at DATeCH 2019 paper, slides, thesis chapter
  - Empirical evaluation of library catalogues at SWIB 2019 slides, paper in English, paper in Spanish
  - quality assessment of Gent University Library catalogue (a running instance of the dashboard): http://gent.qa-catalogue.eu/
  - new: QA catalogue mailing list https://listserv.gwdg.de/mailman/listinfo/qa-catalogue
- If you would like to play with this project, but you don't have MARC21 please to download some recordsets mentioned in Appendix I: Where can I get MARC records? of this document.

Part I

Installation

1 Quick start guide

1.1 Installation

See `INSTALL.md` for dependencies.

1. `wget https://github.com/pkiraly/metadata-qa-marc/releases/download/v0.6.0/metadata-qa-ma`
2. `unzip metadata-qa-marc-0.6.0-release.zip`
3. `cd metadata-qa-marc-0.6.0/`

1.2 Configuration

Either use the script `qa-catalogue` or create configuration files:

4. `cp setdir.sh.template setdir.sh`

Change the input and output base directories in `setdir.sh`. Local directories `input/` and `output/` will be used by default. Files of each catalogue are in a subdirectory of theses base directories:

5. Create configuration based on some existing config files:

- `cp catalogues/loc.sh catalogues/[abbreviation-of-your-library].sh`
- edit `catalogues/[abbreviation-of-your-library].sh according to` configuration guide

1.3 With Docker

A more detailed instruction how to use qa-catalogue with Docker can be found in the wiki

A Docker image bundling qa-catalogue with all of its dependencies and the web interface qa-catalogue-web is made available:

- continuously via GitHub as `ghcr.io/pkiraly/qa-catalogue`
- and for releases via Docker Hub as `pkiraly/metadata-qa-marc`

To download, configure and start an image in a new container the file docker-compose.yml is needed in the current directory. It can be configured with the following environment variables:

- `IMAGE`: which Docker image to download and run. By default the latest image from Docker Hub is used (`pkiraly/metadata-qa-marc`). Alternatives include
  - `IMAGE=ghcr.io/pkiraly/qa-catalogue:main` for most recent image from GitHub packages
  - `IMAGE=metadata-qa-marc` if you have locally build the Docker image
- `CONTAINER`: the name of the docker container. Default: `metadata-qa-marc`.
- `INPUT`: Base directory to put your bibliographic record files in subdirectories of. Set to `./input` by default, so record files are expected to be in `input/$NAME`.
- `OUTPUT`: Base directory to put result of qa-catalogue in subdirectory of. Set to `./output` by default, so files are put in `output/$NAME`.
- `WEBCONFIG`: directory to expose configuration of qa-catalogue-web. Set to `./web-config` by default. If using non-default configuration for data analysis (for instance PICA instead of MARC) then you likely need to adjust configuration of the web interface as well. This directory should contain a configuration file `configuration.cnf`.
- `WEBPORT`: port to expose the web interface. For instance `WEBPORT=9000` will make it available at http://localhost:9000/ instead of http://localhost/.
- `SOLRPORT`: port to expose Solr to. Default: `8983`.

Environment variables can be set on command line or be put in local file `.env`, e.g.:

```
WEBPORT=9000 docker compose up -d
```

or

```
docker compose --env-file config.env up -d
```

When the application has been started this way, run analyses with script `./docker/qa-catalogue` the same ways as script `./qa-catalogue` is called when not using Docker (see usage for details). The following example uses parameters for Gent university library catalogue:

```
./docker/qa-catalogue \
  --params "--marcVersion GENT --alephseq" \
  --mask "rug01.export" \
  --catalogue gent \
  all
```

Now you can reach the web interface (qa-catalogue-web) at http://localhost:80/ (or at another port as configured with environment variable `WEBPORT`). To further modify appearance of the interface, create templates in your `WEBCONFIG` directory and/or create a file `configuration.cnf` in this directory to extend UI configuration without having to restart the Docker container.

This example works under Linux. Windows users should consult the Docker on Windows wiki page. Other useful Docker commands at QA catalogue's wiki.

Everything else should work the same way as in other environments, so follow the next sections.

1.4 Use

```
catalogues/[abbreviation-of-your-library].sh all-analyses
catalogues/[abbreviation-of-your-library].sh all-solr
```

For a catalogue with around 1 million record the first command will take 5-10 minutes, the later 1-2 hours.

2 Build

Prerequisites: Java 11 (I use OpenJDK), and Maven 3

1. Optional step: clone and build the parent library, metadata-qa-api project:

```
git clone https://github.com/pkiraly/metadata-qa-api.git
cd metadata-qa-api
mvn clean install
cd ..
```

2. Mandatory step: clone and build the current qa-catalogue project

```
git clone https://github.com/pkiraly/qa-catalogue.git
cd metadata-qa-marc
mvn clean install
```

3 Download

The released versions of the software is available from Maven Central repository. The stable releases (currently 0.6.0) is available from all Maven repos, while the developer version (*-SNAPSHOT) is available only from the Sonatype Maven repository. What you need to select is the file `metadata-qa-marc-0.6.0-jar-with-dependencies.jar`.

Be aware that no automation exists for creating a current developer version as nightly build, so there is a chance that the latest features are not available in this version. If you want to use the latest version, do build it.

Since the jar file doesn't contain the helper scripts, you might also consider downloading them from this GitHub repository:

```
wget https://raw.githubusercontent.com/pkiraly/qa-catalogue/master/common-script
wget https://raw.githubusercontent.com/pkiraly/qa-catalogue/master/validator
wget https://raw.githubusercontent.com/pkiraly/qa-catalogue/master/formatter
wget https://raw.githubusercontent.com/pkiraly/qa-catalogue/master/tt-completeness
```

You should adjust `common-script` to point to the jar file you just downloaded.

Part II

Usage

Helper scripts

The tool comes with some bash helper scripts to run all these with default values. The generic scripts locate in the root directory and library specific configuration like scripts exist in the `catalogues` directory. You can find predefined scripts for several library catalogues (if you want to run it, first you have to configure it). All these scrips mainly contain configuration, and then it calls the central `common-script` which contains the functions.

If you do not want to

run

```
catalogues/[your script] [command(s)]
```

or

```
./qa-catalogue --params="[options]" [command(s)]
```

The following commands are supported:

- `validate` – runs validation
- `completeness` – runs completeness analysis
- `classifications` – runs classification analysis
- `authorities` – runs authorities analysis
- `tt-completeness` – runs Thomson-Trail completeness analysis
- `shelf-ready-completeness` – runs shelf-ready completeness analysis
- `serial-score` – calculates the serial scores
- `format` – runs formatting records
- `functional-analysis` – runs functional analysis
- `pareto` – runs pareto analysis
- `marc-history` – generates cataloguing history chart
- `prepare-solr` – prepare Solr index (you should already have Solr running, and index created)
- `index` – runs indexing with Solr
- `sqlite` – import tables to SQLite
- `export-schema-files` – export schema files
- `all-analyses` – run all default analysis tasks
- `all-solr` – run all indexing tasks
- `all` – run all tasks
- `config` – show configuration of selected catalogue

You can find information about these functionalities below this document.

configuration

1. create the configuration file (setdir.sh)

```
cp setdir.sh.template setdir.sh
```

2. edit the file configuration file. Two lines are important here

```
BASE_INPUT_DIR=your/path
BASE_OUTPUT_DIR=your/path
BASE_LOG_DIR==your/path
```

- `BASE_INPUT_DIR` is the parent directory where your MARC records exists
- `BASE_OUTPUT_DIR` is where the analysis results will be stored
- `BASE_LOG_DIR` is where the analysis logs will be stored

3. edit the library specific file

Here is an example file for analysing Library of Congress' MARC records

```
#!/usr/bin/env bash

. ./setdir.sh

NAME=loc
MARC_DIR=${BASE_INPUT_DIR}/loc/marc
MASK=*.mrc

. ./common-script
```

Three variables are important here:

1. `NAME` is a name for the output directory. The analysis result will land under $BASE_{O}UTPUT_{D}IR$/NAME directory
2. `MARC_DIR` is the location of MARC files. All the files should be in the same directory
3. `MASK` is a file mask, such as `*.mrc`, `*.marc` or `*.dat.gz`. Files ending with `.gz` are uncompressed automatically.

You can add here any other parameters this document mentioned at the description of individual command, wrapped in TYPE_PARAMS variable e.g. for the Deutche Nationalbibliothek's config file, one can find this

```
TYPE_PARAMS="--marcVersion DNB --marcxml"
```

This line sets the DNB’s MARC version (to cover fields defined within DNB’s MARC version), and XML as input format.

The following table summarizes the configuration variables. The script `qa-catalogue` can be used to set variables and execute analysis without a library specific configuration file:

variable	`qa-catalogue`	description	default
`ANALYSES`	`-a/--analyses`	which tasks to run with `all-analyses`	`validate, validate_sqlite, completeness, completeness_sqlite, classifications, authorities, tt_completeness, shelf_ready_completeness, serial_score, functional_analysis, pareto, marc_history`
	`-c/--catalogue`	display name of the catalogue	`$NAME`
`NAME`	`-n/--name`	name of the catalogue	qa-catalogue
`BASE_INPUT_DIR`	`-d/--input`	parent directory of input file directories	`./input`
`INPUT_DIR`	`-d/--input-dir`	subdirectory of input directory to read files from	
`BASE_OUTPUT_DIR`	`-o/--output`	parent output directory	`./output`
`BASE_LOG_DIR`	`-l/--logs`	directory of log files	`./logs`
`MASK`	`-m/--mask`	a file mask which input files to process, e.g. `*.mrc`	`*`
`TYPE_PARAMS`	`-p/--params`	parameters to pass to individual tasks (see below)	
`SCHEMA`	`-s/--schema`	record schema	`MARC21`
`UPDATE`	`-u/--update`	optional date of input files	
`VERSION`	`-v/--version`	optional version number/date of the catalogue to compare changes	
`WEB_CONFIG`	`-w/--web-config`	update the specified configuration file of qa-catalogue-web	

variable	qa-catalogue	description	default
	`-f/--env-file`	configuration file to load environment variables from (default: `.env`)	

Detailed instructions

We will use the same jar file in every command, so we save its path into a variable.

```
export JAR=target/metadata-qa-marc-0.7.0-jar-with-dependencies.jar
```

General parameters

Most of the analyses uses the following general parameters

- `--schemaType <type>` metadata schema type. The supported types are:
  - `MARC21`
  - `PICA`
  - `UNIMARC` (assessment of UNIMARC records are not yet supported, this parameter value is only reserved for future usage)
- `-m <version>`, `--marcVersion <version>` specifies a MARC version. Currently, the supported versions are:
  - `MARC21`, Library of Congress MARC21
  - `DNB`, the Deuthche Nationalbibliothek's MARC version
  - `OCLC`, the OCLCMARC
  - `GENT`, fields available in the catalog of Gent University (Belgium)
  - `SZTE`, fields available in the catalog of Szegedi Tudományegyetem (Hungary)
  - `FENNICA`, fields available in the Fennica catalog of Finnish National Library
  - `NKCR`, fields available at the National Library of the Czech Republic
  - `BL`, fields available at the British Library
  - `MARC21NO`, fields available at the MARC21 profile for Norwegian public libraries
  - `UVA`, fields available at the University of Amsterdam Library
  - `B3KAT`, fields available at the B3Kat union catalogue of Bibliotheksverbundes Bayern (BVB) and Kooperativen Bibliotheksverbundes Berlin-Brandenburg (KOBV)
  - `KBR`, fields available at KBR, the national library of Belgium
  - `ZB`, fields available at Zentralbibliothek Zürich
  - `OGYK`, fields available at Országygyűlési Könyvtár, Budapest

- `-n`, `--nolog` do not display log messages
- parameters to limit the validation:
  - `-i [record ID]`, `--id [record ID]` validates only a single record having the specifies identifier (the content of 001)
  - `-l [number]`, `--limit [number]` validates only given number of records
  - `-o [number]`, `--offset [number]` starts validation at the given Nth record
  - `-z [list of tags]`, `--ignorableFields [list of tags]` do NOT validate the selected fields. The list should contain the tags separated by commas (`,`), e.g. `--ignorableFields A02,AQN`
  - `-v [selector]`, `--ignorableRecords [selector]` do NOT validate the records which match the condition denoted by the selector. The selector is a test MARCspec string e.g. `--ignorableRecords STA$a=SUPPRESSED`. It ignores the records which has `STA` field with an `a` subfield with the value `SUPPRESSED`.
- `-d [record type]`, `--defaultRecordType [record type]` the default record type to be used if the record's type is undetectable. The record type is calculated from the combination of Leader/06 (Type of record) and Leader/07 (bibliographic level), however sometimes the combination doesn't fit to the standard. In this case the tool will use the given record type. Possible values of the record type argument:
  - BOOKS
  - CONTINUING_RESOURCES
  - MUSIC
  - MAPS
  - VISUAL_MATERIALS
  - COMPUTER_FILES
  - MIXED_MATERIALS
- parameters to fix known issues before any analyses:
  - `-q`, `--fixAlephseq` sometimes ALEPH export contains '^' characters instead of spaces in control fields (006, 007, 008). This flag replaces them with spaces before the validation. It might occur in any input format.
  - `-a`, `--fixAlma` sometimes Alma export contains '#' characters instead of spaces in control fields (006, 007, 008). This flag replaces them with spaces before the validation. It might occur in any input format.
  - `-b`, `--fixKbr` KBR's export contains '#' characters instead spaces in control fields (006, 007, 008). This flag replaces them with spaces before the validation. It might occur in any input format.
- `-f <format>`, `--marcFormat <format>` The input format. Possible values are
  - `ISO`: Binary (ISO 2709)
  - `XML`: MARCXML (shortcuts: `-x`, `--marcxml`)
  - `ALEPHSEQ`: Alephseq (shortcuts: `-p`, `--alephseq`)

  - `LINE_SEPARATED`: Line separated binary MARC where each line contains one record) (shortcuts: `-y`, `--lineSeparated`)
  - `MARC_LINE`: MARC Line is a line-separated format i.e. it is a text file, where each line is a distinct field, the same way as MARC records are usually displayed in the MARC21 standard documentation.
  - `MARCMAKER`: MARCMaker format
  - `PICA_PLAIN`: PICA plain (https://format.gbv.de/pica/plain) is a serialization format, that contains each fields in distinct row.
  - `PICA_NORMALIZED`: normalized PICA (https://format.gbv.de/pica/normalized) is a serialization format where each line is a separate record (by bytecode `0A`). Fields are terminated by bytecode 1E, and subfields are introduced by bytecode `1F`.

- `-t <directory>`, `--outputDir <directory>` specifies the output directory where the files will be created
- `-r`, `--trimId` remove spaces from the end of record IDs in the output files (some library system add padding spaces around field value 001 in exported files)
- `-g <encoding>`, `--defaultEncoding <encoding>` specify a default encoding of the records. Possible values:
  - `ISO-8859-1` or `ISO8859_1` or `ISO_8859_1`
  - `UTF8` or `UTF-8`
  - `MARC-8` or `MARC8`
- `-s <datasource>`, `--dataSource <datasource>` specify the type of data source. Possible values:
  - `FILE`: reading from file
  - `STREAM`: reading from a Java data stream. It is not usable if you use the tool from the command line, only if you use it with its API.
- `-c <configuration>`, `--allowableRecords <configuration>` if set, criteria which allows analysis of records. If the record does not met the criteria, it will be excluded. An individual criterium should be formed as a MarcSpec (for MARC21 records) or PicaFilter (for PICA records). Multiple criteria might be concatenated with logical operations: `&&` for AND, `||` for OR and `!` for not. One can use parentheses to group logical expressions. An example: `'002@.0 !~ "^L" && 002@.0 !~ "^..[iktN]" && (002@.0 !~ "^.v" || 021A.a?)'`. Since the criteria might form a complex phase containing spaces, the passing of which is problematic among multiple scripts, one can apply Base64 encoding. In this case add `base64:` prefix to the parameters, such as `base64:"$(echo '002@.0 !~ "^L" && 002@.0 !~ "^..[iktN]" && (002@.0 !~ "^.v" || 021A.a?)' | base64 -w 0)`.
- `-1 <type>`, `--alephseqLineType <type>`, true, "Alephseq line type. The `type` could be
  - `WITH_L`: the records' AlephSeq lines contain an `L` string (e.g. `000000002 008   L 780804s1977^^^^enk||||||b||||001^0|eng||`)

  - `WITHOUT_L`: the records' AlephSeq lines do not contai an `L` string (e.g. `000000002 008   780804s1977^^^^enk||||||b||||001^0|eng||`)

- PICA related parameters

  - `-2 <path>`, `--picaIdField <path>` the record identifier
  - `-u <char>`, `--picaSubfieldSeparator <char>` the PICA subfield separator. subfield of PICA records. Default is `003@$0`. Default is `$`.
  - `-j <file>`, `--picaSchemaFile <file>` an Avram schema file, which describes the structure of PICA records
  - `-k <path>`, `--picaRecordType <path>` The PICA subfield which stores the record type information. Default is `002@$0`.

- Parameters for grouping analyses

  - `-e <path>`, `--groupBy <path>` group the results by the value of this data element (e.g. the ILN of libraries holding the item). An example: `--groupBy 001@$0` where `001@$0` is the subfield containing the comma separated list of library ILN codes.
  - `-3 <file>`, `--groupListFile <file>` the file which contains a list of ILN codes

The last argument of the commands are a list of files. It might contain any wildcard the operating system supports ('*','?', etc.).

4 Validating MARC records

It validates each records against the MARC21 standard, including those local defined field, which are selected by the MARC version parameter.

The issues are classified into the following categories: record, control field, data field, indicator, subfield and their subtypes.

There is an uncertainty in the issue detection. Almost all library catalogues have fields, which are not part of the MARC standard, neither that of their documentation about the locally defined fields (these documents are rarely available publicly, and even if they are available sometimes they do not cover all fields). So if the tool meets a field which are undefined, it is impossible to decide whether it is valid or invalid in a particular context. So in some places the tool reflects this uncertainty and provides two calculations, one which handles these fields as error, and another which handles these as valid fields.

The tool detects the following issues:

machine name

explanation

record level issues

undetectableType

the document type is not detectable

invalidLinkage

the linkage in field 880 is invalid

ambiguousLinkage

the linkage in field 880 is ambiguous

control field position issues

obsoleteControlPosition

the code in the position is obsolete (it was valid in a previous version of MARC, but it is not valid now)

controlValueContainsInvalidCode

the code in the position is invalid

invalidValue

the position value is invalid

data field issues

missingSubfield

missing reference subfield (880$6)

nonrepeatableField

repetition of a non-repeatable field

undefinedField

the field is not defined in the specified MARC version(s)

indicator issues

obsoleteIndicator

the indicator value is obsolete (it was valid in a previous version of MARC, but not in the current version)

nonEmptyIndicator

indicator that should be empty is non-empty

invalidValue

the indicator value is invalid

subfield issues

undefinedSubfield

the subfield is undefined in the specified MARC version(s)

invalidLength

the length of the value is invalid

invalidReference

the reference to the classification vocabulary is invalid

patternMismatch

content does not match the patterns specified by the standard

nonrepeatableSubfield

repetition of a non-repeatable subfield

invalidISBN

invalid ISBN value

invalidISSN

invalid ISSN value

unparsableContent

the value of the subfield is not well-formed according to its specification

nullCode

null subfield code

invalidValue

invalid subfield value

Usage:

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.Validator [options] <file>
```

or with a bash script

```
./validator [options] <file>
```

or

```
catalogues/<catalogue>.sh validate
```

or

```
./qa-catalogue --params="[options]" validate
```

options:

- general parameters
- granularity of the report
  - `-S`, `--summary`: creating a summary report instead of record level reports
  - `-H`, `--details`: provides record level details of the issues
- output parameters:

  - `-G <file>`, `--summaryFileName <file>`: the name of summary report the program produces. The file provides a summary of issues, such as the number of instance and number of records having the particular issue.
  - `-F <file>`, `--detailsFileName <file>`: the name of report the program produces. Default is `validation-report.txt`. If you use "stdout", it won't create file, but put results into the standard output.
  - `-R <format>`, `--format <format>`: format specification of the output. Possible values:
    - `text` (default),
    - `tab-separated` or `tsv`,
    - `comma-separated` or `csv`

- `-W`, `--emptyLargeCollectors`: the output files are created during the process and not only at the end of it. It helps in memory management if the input is large, and it has lots of errors, on the other hand the output file will be segmented, which should be handled after the process.
- `-T`, `--collectAllErrors`: collect all errors (useful only for validating small number of records). Default is turned off.
- `-I <types>`, `--ignorableIssueTypes <types>`: comma separated list of issue types not to collect. The valid values are (for details see the issue types table):
  - `undetectableType`: undetectable type
  - `invalidLinkage`: invalid linkage
  - `ambiguousLinkage`: ambiguous linkage
  - `obsoleteControlPosition`: obsolete code
  - `controlValueContainsInvalidCode`: invalid code
  - `invalidValue`: invalid value
  - `missingSubfield`: missing reference subfield (880$6)
  - `nonrepeatableField`: repetition of non-repeatable field
  - `undefinedField`: undefined field
  - `obsoleteIndicator`: obsolete value
  - `nonEmptyIndicator`: non-empty indicator
  - `invalidValue`: invalid value
  - `undefinedSubfield`: undefined subfield
  - `invalidLength`: invalid length
  - `invalidReference`: invalid classification reference
  - `patternMismatch`: content does not match any patterns
  - `nonrepeatableSubfield`: repetition of non-repeatable subfield
  - `invalidISBN`: invalid ISBN
  - `invalidISSN`: invalid ISSN
  - `unparsableContent`: content is not well-formatted
  - `nullCode`: null subfield code
  - `invalidValue`: invalid value

Outputs: * `count.csv`: the count of bibliographic records in the source dataset

```
total
1192536
```

- `issue-by-category.csv`: the counts of issues by categories. Columns:
  - `id` the identifier of error category
  - `category` the name of the category
  - `instances` the number of instances of errors within the category (one record might have multiple instances of the same error)
  - `records` the number of records having at least one of the errors within the category

```
id,category,instances,records
2,control field,994241,313960
3,data field,12,12
4,indicator,5990,5041
5,subfield,571,555
```

- `issue-by-type.csv`: the count of issues by types (subcategories).

```
id,categoryId,category,type,instances,records
5,2,control field,"invalid code",951,541
6,2,control field,"invalid value",993290,313733
8,3,data field,"repetition of non-repeatable field",12,12
10,4,indicator,"obsolete value",1,1
11,4,indicator,"non-empty indicator",33,32
12,4,indicator,"invalid value",5956,5018
13,5,subfield,"undefined subfield",48,48
14,5,subfield,"invalid length",2,2
15,5,subfield,"invalid classification reference",2,2
16,5,subfield,"content does not match any patterns",286,275
17,5,subfield,"repetition of non-repeatable subfield",123,120
18,5,subfield,"invalid ISBN",5,3
19,5,subfield,"invalid ISSN",105,105
```

- `issue-summary.csv`: details of individual issues including basic statistics

```
id,MarcPath,categoryId,typeId,type,message,url,instances,records
53,008/33-34 (008map33),2,5,invalid code,'b' in 'b ',https://www.loc.gov/marc/bibliographic/
70,008/00-05 (008all00),2,5,invalid code,Invalid content: '2023  '. Text '2023  ' could not
28,008/22-23 (008map22),2,6,invalid value,| ,https://www.loc.gov/marc/bibliographic/bd008p.h
19,008/31 (008book31),2,6,invalid value, ,https://www.loc.gov/marc/bibliographic/bd008b.html
17,008/29 (008book29),2,6,invalid value, ,https://www.loc.gov/marc/bibliographic/bd008b.html
```

- `issue-details.csv`: list of issues by record identifiers. It has two columns, the record identifier, and a complex string, which contains the number of occurrences of each individual issue concatenated by semicolon.

```
recordId,errors
99117335059205508,1:2;2:1;3:1
99117335059305508,1:1
99117335059405508,2:2
99117335059505508,3:1
```

`1:2;2:1;3:1` means that 3 different types of issues are occurred in the record, the firs issue which has issue ID 1 occurred twice, issue ID 2 which occurred once and issue ID 3, which occurred once. The issue IDs can be resolved from the `issue-summary.csv` file's firs column.

- `issue-details-normalized.csv`: the normalized version of the previous file

```
id,errorId,instances
99117335059205508,1,2
99117335059205508,2,1
99117335059205508,3,1
99117335059305508,1,1
99117335059405508,2,2
99117335059505508,3,1
```

- `issue-total.csv`: the number of issue free records, and number of record having issues

```
type,instances,records
0,0,251
1,1711,848
2,413,275
```

where types are - 0: records without errors - 1: records with any kinds of errors - 2: records with errors excluding invalid field errors

- `issue-collector.csv`: non normalized file of record ids per issues. This is the "inverse" of `issue-details.csv`, it tells you in which records a particular issue occurred.

```
errorId,recordIds
1,99117329355705508;99117328948305508;99117334968905508;99117335067705508;99117335176005508;
```

- `validation.params.json`: the list of the actual parameters during the running of the validation

An example with parameters used for analysing a PICA dataset. When the input is a complex expression it is displayed here in a parsed format. It also contains some metadata such as the versions of MQFA API and QA catalogue.

```
{
  "args":["/path/to/input.dat"],
  "marcVersion":"MARC21",
  "marcFormat":"PICA_NORMALIZED",
  "dataSource":"FILE",
  "limit":-1,
  "offset":-1,
  "id":null,
  "defaultRecordType":"BOOKS",
  "alephseq":false,
  "marcxml":false,
  "lineSeparated":false,
  "trimId":true,
  "outputDir":"/path/to/_output/k10plus_pica",
  "recordIgnorator":{
    "criteria":[],
    "booleanCriteria":null,
    "empty":true
  },
  "recordFilter":{
    "criteria":[],
    "booleanCriteria":{
      "op":"AND",
      "children":[
        {
          "op":null,
          "children":[],
          "value":{
            "path":{
              "path":"002@.0",
              "tag":"002@",
              "xtag":null,
              "occurrence":null,
              "subfields":{"type":"SINGLE","input":"0","codes":["0"]},
              "subfieldCodes":["0"]
            },
            "operator":"NOT_MATCH",
            "value":"^L"
          }
        },
        {"op":null,"children":[],"value":{"path":{"path":"002@.0","tag":"002@","xtag":null,"c
        {"op":"OR","children":[{"op":null,"children":[],"value":{"path":{"path":"002@.0","tag
      ],
      "value":null
    },
    "empty":false
  },
  "ignorableFields":{
    "fields":["001@","001E","001L","001U","001U","001X","001X","002V","003C","003G","003Z","0
    "empty":false
  },
  "stream":null,
  "defaultEncoding":null,
  "alephseqLineType":null,
  "picaIdField":"003@$0",
  "picaSubfieldSeparator":"$",
  "picaSchemaFile":null,
  "picaRecordTypeField":"002@$0",
  "schemaType":"PICA",
  "groupBy":null,
  "detailsFileName":"issue-details.csv",
  "summaryFileName":"issue-summary.csv",
  "format":"COMMA_SEPARATED",
  "ignorableIssueTypes":["FIELD_UNDEFINED"],
  "pica":true,
  "replacementInControlFields":null,
  "marc21":false,
  "mqaf.version":"0.9.2",
  "qa-catalogue.version":"0.7.0-SNAPSHOT"
}
```

- `id-groupid.csv`: the pairs of record identifiers - group identifiers.

```
id,groupId
010000011,0
010000011,77
010000011,2035
010000011,70
010000011,20
```

Currently, validation detects the following errors:

Leader specific errors:

- Leader/[position] has an invalid value: ‘[value]’ (e.g. `Leader/19 (leader19) has an invalid value: '4'`)

Control field specific errors:

- 006/[position] ([name]) contains an invalid code: ‘[code]’ in ‘[value]’ (e.g. `006/01-05 (tag006book01) contains an invalid code: 'n' in '  n '`)
- 006/[position] ([name]) has an invalid value: ‘[value]’ (e.g. `006/13 (tag006book13) has an invalid value: ' '`)
- 007/[position] ([name]) contains an invalid code: ‘[code]’ in ‘[value]’
- 007/[position] ([name]) has an invalid value: ‘[value]’ (e.g. `007/01 (tag007microform01) has an invalid value: ' '`)
- 008/[position] ([name]) contains an invalid code: ‘[code]’ in ‘[value]’ (e.g. `008/18-22 (tag008book18) contains an invalid code: 'u' in 'u   '`)
- 008/[position] ([name]) has an invalid value: ‘[value]’ (e.g. `008/06 (tag008all06) has an invalid value: ' '`)

Data field specific errors

- Unhandled tag(s): [tags] (e.g. `Unhandled tag: 265`)
- [tag] is not repeatable, however there are [number] instances
- [tag] has invalid subfield(s): [subfield codes] (e.g. `110 has invalid subfield: s`)
- [tag][*indicator*]*hasinvalidcode* :′ [*code*]′(*e.g.*‵110ind1 has invalid code: ‘2’‘)
- [tag][*indicator*]*shouldbeempty*, *ithas*′[*code*]′(*e.g.*‵110ind2 should be empty, it has ‘0’‘)
- [tag][*subfieldcode*]*isnotrepeatable*, *howeverthereare*[*number*]*instances*(*e.g.*‵072a is not repeatable, however there are 2 instances‘)
- [tag][*subfieldcode*]*hasaninvalidvalue* : [*value*](*e.g.*‵046a has an invalid value: ‘fb——’‘)

Errors of specific fields:

- 045*aerrorin*′[*value*]′ : *lengthisnot*4*char*(*e.g.*‵045a error in ‘2209668’: length is not 4 char‘)
- 045$a error in ‘[value]’: ‘[part]’ does not match any patterns
- 880 should have subfield $a
- 880 refers to field [tag], which is not defined (e.g. `880 refers to field 590, which is not defined`)

An example:

```
Error in '   00000034 ':
  110$ind1 has invalid code: '2'
Error in '   00000056 ':
  110$ind1 has invalid code: '2'
```

```
Error in '   00000057 ':
  082$ind1 has invalid code: ' '
Error in '   00000086 ':
  110$ind1 has invalid code: '2'
Error in '   00000119 ':
  700$ind1 has invalid code: '2'
Error in '   00000234 ':
  082$ind1 has invalid code: ' '
Errors in '   00000294 ':
  050$ind2 has invalid code: ' '
  260$ind1 has invalid code: '0'
  710$ind2 has invalid code: '0'
  710$ind2 has invalid code: '0'
  710$ind2 has invalid code: '0'
  740$ind2 has invalid code: '1'
Error in '   00000322 ':
  110$ind1 has invalid code: '2'
Error in '   00000328 ':
  082$ind1 has invalid code: ' '
Error in '   00000374 ':
  082$ind1 has invalid code: ' '
Error in '   00000395 ':
  082$ind1 has invalid code: ' '
Error in '   00000514 ':
  082$ind1 has invalid code: ' '
Errors in '   00000547 ':
  100$ind2 should be empty, it has '0'
  260$ind1 has invalid code: '0'
Errors in '   00000571 ':
  050$ind2 has invalid code: ' '
  100$ind2 should be empty, it has '0'
  260$ind1 has invalid code: '0'
...
```

4.0.0.1 post processing validation result (validate-sqlite)

Usage:

```
catalogues/<catalogue>.sh validate-sqlite
```

or

```
./qa-catalogue --params="[options]" validate-sqlite
```

or

```
./common-script [options] validate-sqlite
```

[options] are the same as for validation

4.0.0.1.1 Catalogue for a single library

If the data is *not* grouped by libraries (no `--groupBy <path>` parameter), it creates the following SQLite3 database structure and import some of the CSV files into it:

`issue_summary` table for the `issue-summary.csv`:

It represents a particular type of error

```
id          INTEGER,  -- identifier of the error
MarcPath    TEXT,     -- the location of the error in the bibliographic record
categoryId  INTEGER,  -- the identifier of the category of the error
typeId      INTEGER,  -- the identifier of the type of the error
type        TEXT,     -- the description of the type
message     TEXT,     -- extra contextual information
url         TEXT,     -- the url of the definition of the data element
instances   INTEGER,  -- the number of instances this error occured
records     INTEGER   -- the number of records this error occured in
```

`issue_details` table for the `issue-details.csv`:

Each row represents how many instances of an error occur in a particular bibliographic record

```
id          TEXT,    -- the record identifier
errorId     INTEGER, -- the error identifier (-> issue_summary.id)
instances   INTEGER  -- the number of instances of an error in the record
```

4.0.0.1.2 Union catalogue for multiple libraries

If the dataset is a union catalogue, and the record contains a subfield for the libraries holding the item (there is `--groupBy <path>` parameter), it creates the following SQLite3 database structure and import some of the CSV files into it:

`issue_summary` table for the `issue-summary.csv` (it is similar to the other issue_summary table, but it has an extra `groupId` column)

```
groupId    INTEGER,
id         INTEGER,
MarcPath   TEXT,
categoryId INTEGER,
typeId     INTEGER,
type       TEXT,
message    TEXT,
url        TEXT,
instances  INTEGER,
records    INTEGER
```

`issue_details` table (same as the other `issue_details` table)

```
id         TEXT,
errorId    INTEGER,
instances  INTEGER
```

`id_groupid` table for `id-groupid.csv`:

```
id         TEXT,
groupId    INTEGER
```

`issue_group_types` table contains statistics for the error types per groups.

```
groupId    INTEGER,
typeId     INTEGER,
records    INTEGER,
instances  INTEGER
```

`issue_group_categories` table contains statistics for the error categories per groups

```
groupId    INTEGER,
categoryId INTEGER,
records    INTEGER,
instances  INTEGER
```

`issue_group_paths` table contains statistics for the error types per paths per groups

```
groupId    INTEGER,
typeId     INTEGER,
path       TEXT,
records    INTEGER,
instances  INTEGER
```

For union catalogues it also creates an extra Solr index with the suffix `_validation`. It contains one Solr document for each bibliographic record with three fields: the record identifier, the list of group identifiers and the list of error identifiers (if any). This Solr index is needed for populating the `issue_group_types`, `issue_group_categories` and `issue_group_paths` tables. This index will be ingested into the main Solr index.

5 Display one MARC record, or extract data elements from MARC records

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.Formatter [options] <file>
```

or with a bash script

```
./formatter [options] <file>
```

options:

- general parameters
- `-f`, `--format`: the MARC output format
  - if not set, the output format follows the examples in the MARC21 documentation (see the example below)
  - `xml`: the output will be MARCXML
- `-c <number>`, `-countNr <number>`: count number of the record (e.g. 1 means the first record)
- `-s [path=query]`, `-search [path=query]`: print records matching the query. The query part is the content of the element. The path should be one of the following types:
  - control field tag (e.g. `001`, `002`, `003`)
  - control field position (e.g. `Leader/0`, `008/1-2`)
  - data field (`655\$2`, `655\$ind1`)
  - named control field position (`tag006book01`)
- `-l <selector>`, `--selector <selector>`: one or more MarcSpec or PICA Filter selectors, separated by ';' (semicolon) character
- `-w`, `--withId`: the generated CSV should contain record ID as first field (default is turned off)
- `-p <separator>`, `--separator <separator>`: separator between the parts (default: TAB)
- `-e <file>`, `--fileName <file>`: the name of report the program produces (default: `extracted.csv`)

- `-A <identifiers>`, `--ids <identifiers>`: a comma separated list of record identifiers

The output of displaying a single MARC record is something like this one:

```
LEADER 01697pam a2200433 c 4500
001 1023012219
003 DE-101
005 20160912065830.0
007 tu
008 120604s2012    gw ||||| |||| 00||||ger
015   $a14,B04$z12,N24$2dnb
016 7 $2DE-101$a1023012219
020   $a9783860124352$cPp. : EUR 19.50 (DE), EUR 20.10 (AT)$9978-3-86012-435-2
024 3 $a9783860124352
035   $a(DE-599)DNB1023012219
035   $a(OCoLC)864553265
035   $a(OCoLC)864553328
040   $a1145$bger$cDE-101$d1140
041   $ager
044   $cXA-DE-SN
082 04$81\u$a622.0943216$qDE-101$222/ger
083 7 $a620$a660$qDE-101$222sdnb
084   $a620$a660$qDE-101$2sdnb
085   $81\u$b622
085   $81\u$z2$s43216
090   $ab
110 1 $0(DE-588)4665669-8$0http://d-nb.info/gnd/4665669-8$0(DE-101)963486896$aHalsbrücke$4au
245 00$aHalsbrücke$bzur Geschichte von Gemeinde, Bergbau und Hütten$chrsg. von der Gemeinde 
264  1$a[Freiberg]$b[Techn. Univ. Bergakad.]$c2012
300   $a151 S.$bIll., Kt.$c31 cm, 1000 g
653   $a(Produktform)Hardback
653   $aGemeinde Halsbrücke
653   $aHüttengeschichte
653   $aFreiberger Bergbau
653   $a(VLB-WN)1943: Hardcover, Softcover / Sachbücher/Geschichte/Regionalgeschichte, Lände
700 1 $0(DE-588)1113208554$0http://d-nb.info/gnd/1113208554$0(DE-101)1113208554$aThiel, Ulri
850   $aDE-101a$aDE-101b
856 42$mB:DE-101$qapplication/pdf$uhttp://d-nb.info/1023012219/04$3Inhaltsverzeichnis
925 r $arb
```

An example for extracting values:

```
./formatter --selector "008~7-10;008~0-5" \
            --defaultRecordType BOOKS \
            --separator "," \
            --outputDir ${OUTPUT_DIR} \
            --fileName marc-history.csv \
             ${MARC_DIR}/*.mrc
```

It will put the output into `${OUTPUT_DIR}/marc-history.csv`.

6 Completeness

Counts basic statistics about the data elements available in the catalogue.

Usage:

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.Completeness [options] <file>
```

or with a bash script

```
./completeness [options] <file>
```

or

```
catalogues/<catalogue>.sh completeness
```

or

```
./qa-catalogue --params="[options]" completeness
```

options:

- general parameters
- `-R <format>`, `--format <format>`: format specification of the output. Possible values are:
  - `tab-separated` or `tsv`,
  - `comma-separated` or `csv`,
  - `text` or `txt`
  - `json`
- `-V`, `--advanced`: advanced mode (not yet implemented)
- `-P`, `--onlyPackages`: only packages (not yet implemented)

6.1 Output files:

6.1.1 marc-elements.csv

is list of MARC elements (field$subfield) and their occurrences in two ways as number or records, and number of instances. The columns in the file are:

- `documenttype`: the document types found in the dataset. There is an extra document type: `all` representing all records
- `path`: the notation of the data element
- `packageid` and `package`: each path belongs to one package, such as `Control Fields`, and each package has an internal identifier.
- `tag`: the label of tag
- `subfield`: the label of subfield
- `number-of-record`: means how many records they are available,
- `number-of-instances`: means how many instances are there in total (some records might contain more than one instances, while others don't have them at all)
- `min`, `max`, `mean`, `stddev` the minimum, maximum, mean and standard deviation of the number of instances per record (as floating point numbers)
- `histogram`: the histogram of the instances (`1=1; 2=1` means: a single instance is available in one record, two instances are available in one record)

documenttype	path	packageid	package	tag	subfield	number-of-record	number-of-instances	min	max	mean	stddev	histogram
all	leader23	0	Control Fields	Leader	Undefined	1099	1099	1	1	1.0	0.0	1=1099
all	leader22	0	Control Fields	Leader	Length of the implementation-defined portion	1099	1099	1	1	1.0	0.0	1=1099
all	leader21	0	Control Fields	Leader	Length of the starting-character-position portion	1099	1099	1	1	1.0	0.0	1=1099

documenttype	path	packageid	package	tag	subfield	number-of-record	number-of-instances	min	max	mean	stddev	histogram
all	110$a	2	Main Entry	Main Entry - Corporate Name	Corporate name or jurisdiction name as entry element	4	4	1	1	1.0	0.0	1=4
all	340$b	5	Physical Description	Physical Medium	Dimensions	2	3	1	2	1.5	0.3535533905932738	1=1, 2=1
all	363$a	5	Physical Description	Normalized Date and Sequential Designation	First level of enumeration	1	1	1	1	1.0	0.0	1=1
all	340$a	5	Physical Description	Physical Medium	Material base and configuration	2	3	1	2	1.5	0.3535533905932738	1=1, 2=1

6.1.2 packages.csv

The completeness of packages (packages are groups of tags)

Its columns:

- `documenttype`: the document type of the record
- `packageid`: the identifier of the package
- `name`: name of the package
- `label`: label of the package
- `iscoretag`: does the package belong to the Library of Congress MARC standard
- `count`: the number of records having at least one data element from this package

documenttype	packageid	name	label	iscoretag	count
all	1	01X-09X	Numbers and Code	true	1099
all	2	1XX	Main Entry	true	816
all	6	4XX	Series Statement	true	358
all	5	3XX	Physical Description	true	715
all	8	6XX	Subject Access	true	514
all	4	25X-28X	Edition, Imprint	true	1096
all	7	5XX	Note	true	354
all	0	00X	Control Fields	true	1099
all	99	unknown	unknown origin	false	778

6.1.3 libraries.csv

Lists the content of the 852$a (it is useful only if the catalog is an aggregated catalog). Its columns are:

- `library`: the code of a library
- `count`: the number of records having a particular library code

library	count
“00Mf”	713
“British Library”	525
“Inserted article about the fires from the Courant after the title page.”	1
“National Library of Scotland”	310
“StEdNL”	1
“UkOxU”	33

6.1.4 libraries003.csv

List the content of the 003 (it is useful only if the catalog is an aggregated catalog). Its columns are:

- `library`: the code of a library
- `count`: the number of records having a particular library code

library	count
"103861"	1
"BA-SaUP"	143
"BoCbLA"	25
"CStRLIN"	110
"DLC"	3

6.1.5 completeness.params.json

The list of the actual parameters in analysis.

An example with parameters used for analysing a MARC dataset. When the input is a complex expression it is displayed here in a parsed format. It also contains some metadata such as the versions of MQFA API and QA catalogue.

```
{
  "args":["/path/to/input.xml.gz"],
  "marcVersion":"MARC21",
  "marcFormat":"XML",
  "dataSource":"FILE",
  "limit":-1,
  "offset":-1,
  "id":null,
  "defaultRecordType":"BOOKS",
  "alephseq":false,
  "marcxml":true,
  "lineSeparated":false,
  "trimId":false,
  "outputDir":"/path/to/_output/",
  "recordIgnorator":{
    "conditions":null,
    "empty":true
  },
  "recordFilter":{
    "conditions":null,
    "empty":true
  },
  "ignorableFields":{
    "fields":null,
    "empty":true
  },
  "stream":null,
  "defaultEncoding":null,
  "alephseqLineType":null,
  "picaIdField":"003@$0",
  "picaSubfieldSeparator":"$",
  "picaSchemaFile":null,
  "picaRecordTypeField":"002@$0",
  "schemaType":"MARC21",
  "groupBy":null,
  "groupListFile":null,
  "format":"COMMA_SEPARATED",
  "advanced":false,
  "onlyPackages":false,
  "replacementInControlFields":"#",
  "marc21":true,
  "pica":false,
  "mqaf.version":"0.9.2",
  "qa-catalogue.version":"0.7.0"
}
```

6.2 Output files for union catalogues

For union catalogues the `marc-elements.csv` and `packages.csv` have a special version.

6.2.1 completeness-grouped-marc-elements.csv

The same as `marc-elements.csv` but with an extra element `groupId`

- `groupId`: the library identifier available in the data element specified by the `--groupBy` parameter. `0` has a special meaning: all libraries

groupId	documenttype	path	packageid	package	tag	subfield	number-of-record	number-of-instances	min	max	mean	stddev	histogram
350	all	044K$9	50	PICA+ bibliographic description	"Schlagwortfolgen (GBV, SWB, K10plus)"	PPN	1	1	1	1	1.0	0.0	1=1
350	all	044K$7	50	PICA+ bibliographic description	"Schlagwortfolgen (GBV, SWB, K10plus)"	Vorläufiger Link	1	1	1	1	1.0	0.0	1=1

6.2.2 completeness-grouped-packages.csv

The same as `packages.csv` but with an extra element `group`

- `group`: the library identifier available in the data element specified by the `--groupBy` parameter. `0` has a special meaning: all libraries

group	documenttype	packageid	name	label	iscoretag	count
0	Druckschriften (einschließlich Bildbänden)	50	0...	PICA+ bibliographic description	false	987
0	Druckschriften (einschließlich Bildbänden)	99	unknown	unknown origin	false	3

group	documenttype	packageid	name	label	iscoretag	count
0	Medienkombination	50	0...	PICA+ bib- lio- graphic de- scrip- tion	false	1
0	Mikroform	50	0...	PICA+ bib- lio- graphic de- scrip- tion	false	11
0	Tonträger, Videodatenträger, Bildliche Darstellungen	50	0...	PICA+ bib- lio- graphic de- scrip- tion	false	1
0	all	50	0...	PICA+ bib- lio- graphic de- scrip- tion	false	1000
0	all	99	unknown	unknown ori- gin	false	3
100	Druckschriften (einschließlich Bildbänden)	50	0...	PICA+ bib- lio- graphic de- scrip- tion	false	20

group	documenttype	packageid	name	label	iscoretag	count
100	Medienkombination	50	0…	PICA+ bibliographic description	false	1

6.2.3 completeness-groups.csv

This is available for union catalogues, containing the groups

- `id`: the group identifier
- `group`: the name of the library
- `count`: the number of records from the particular library

id	group	count
0	all	1000
100	Otto-von-Guericke-Universität, Universitätsbibliothek Magdeburg [DE-Ma9]	21
1003	Kreisarchäologie Rotenburg [DE-MUS-125322…]	1
101	Otto-von-Guericke-Universität, Universitätsbibliothek, Medizinische Zentralbibliothek (MZB), Magdeburg [DE-Ma14…]	6
1012	Mariengymnasium Jever [DE-Je1]	19

6.2.4 id-groupid.csv

This is the very same file what validation creates. Completeness creates it only if it is not yet available.

6.3 post processing completeness result (completeness-sqlite)

The `completeness-sqlite` step (which is launched by the `completeness` step, but could be launched independently as well) imports `marc-elements.csv` or `completeness-grouped-marc-elements.csv` file into `marc_elements` table. For the catalogues without the `--groupBy` parameter the `groupId` column will be filled by `0`. Its columns are:

```
groupId             INTEGER,
documenttype        TEXT,
path                TEXT,
packageid           INTEGER,
package             TEXT,
tag                 TEXT,
subfield            TEXT,
number-of-record    INTEGER,
number-of-instances INTEGER,
min                 INTEGER,
max                 INTEGER,
mean                REAL,
stddev              REAL,
histogram           TEXT
```

7 Thompson-Traill completeness

Kelly Thompson and Stacie Traill recently published their approach to calculate the quality of ebook records coming from different data sources. Their article is *Implementation of the scoring algorithm described in Leveraging Python to improve ebook metadata selection, ingest, and management.* In Code4Lib Journal, Issue 38, 2017-10-18. http://journal.code4lib.org/articles/12828

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.ThompsonTraillCompleteness [options] <file>
```

or with a bash script

```
./tt-completeness [options] <file>
```

or

```
catalogues/[catalogue].sh tt-completeness
```

or

```
./qa-catalogue --params="[options]" tt-completeness
```

options:

- general parameters
- `-F <file>`, `--fileName <file>`: the name of report the program produces. Default is `tt-completeness.csv`.

It produces a CSV file like this:

```
id,ISBN,Authors,Alternative Titles,Edition,Contributors,Series,TOC,Date 008,Date 26X,LC/NLM,
LoC,Mesh,Fast,GND,Other,Online,Language of Resource,Country of Publication,noLanguageOrEngli
RDA,total
"010002197",0,0,0,0,0,0,0,1,2,0,0,0,0,0,0,0,1,0,0,0,4
"01000288X",0,0,1,0,0,1,0,1,2,0,0,0,0,0,0,0,0,0,0,0,5
```

```
"010004483",0,0,1,0,0,0,0,1,2,0,0,0,0,0,0,0,1,0,0,0,5
"010018883",0,0,0,0,1,0,0,1,2,0,0,0,0,0,0,0,1,1,0,0,6
"010023623",0,0,3,0,0,0,0,1,2,0,0,0,0,0,0,0,1,0,0,0,7
"010027734",0,0,3,0,1,2,0,1,2,0,0,0,0,0,0,0,1,0,0,0,10
```

8 Shelf-ready completeness

This analysis is the implementation of the following paper:

Emma Booth (2020) *Quality of Shelf-Ready Metadata. Analysis of survey responses and recommendations for suppliers* Pontefract (UK): National Acquisitions Group, 2020. p 31. https://nag.org.uk/wp-content/uploads/2020/06/NAG-Quality-of-Shelf-Ready-Metadata-Survey-Analysis-and-Recommendations_FINAL_June2020.pdf

The main purpose of the report is to highlight which fields of the printed and electronic book records are important when the records are coming from different suppliers. 50 libraries participated in the survey, each selected which fields are important to them. The report listed those fields which gets the highest scores.

The current calculation based on this list of essential fields. If all data elements specified are available in the record it gets the full score, if only some of them, it gets a proportional score. E.g. under 250 (edition statement) there are two subfields. If both are available, it gets score 44. If only one of them, it gets the half of it, 22, and if none, it gets 0. For 1XX, 6XX, 7XX and 8XX the record gets the full scores if at least one of those fields (with subfield $a) is available. The total score became the average. The theoretical maximum score would be 28.44, which could be accessed if all the data elements are available in the record.

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.ShelfReadyCompleteness [options] <file>
```

with a bash script

```
./shelf-ready-completeness [options] <file>
```

or

```
catalogues/[catalogue].sh shelf-ready-completeness
```

or

```
./qa-catalogue --params="[options]" shelf-ready-completeness
```

options:

- general parameters
- `-F <file>`, `--fileName <file>`: the report file name (default is `shelf-ready-completeness.csv`)

9 Serial score analysis

These scores are calculated for each continuing resources (type of record (LDR/6) is language material ('a') and bibliographic level (LDR/7) is serial component part ('b'), integrating resource ('i') or serial ('s')).

The calculation is based on a slightly modified version of the method published by Jamie Carlstone in the following paper:

Jamie Carlstone (2017) *Scoring the Quality of E-Serials MARC Records Using Java*, Serials Review, 43:3-4, pp. 271-277, DOI: 10.1080/00987913.2017.1350525

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.SerialScore [options] <file>
```

with a bash script

```
./serial-score [options] <file>
```

or

```
catalogues/[catalogue].sh serial-score
```

or

```
./qa-catalogue --params="[options]" serial-score
```

options:

- general parameters
- `-F <file>`, `--fileName <file>`: the report file name. Default is `shelf-ready-completeness.csv`.

10 FRBR functional requirement analysis

The Functional Requirements for Bibliographic Records (FRBR) document's main part defines the primary and secondary entities which became famous as FRBR models. Years later Tom Delsey created a mapping between the 12 functions and the individual MARC elements.

Tom Delsey (2002) *Functional analysis of the MARC 21 bibliographic and holdings formats. Tech. report.* Library of Congress, 2002. Prepared for the Network Development and MARC Standards Office Library of Congress. Second Revision: September 17, 2003. https://www.loc.gov/marc/marc-functional-analysis/original_source/analysis.pdf.

This analysis shows how these functions are supported by the records. Low support means that only small portion of the fields support a function are available in the records, strong support on the contrary means lots of fields are available. The analyses calculate the support of 12 functions for each record, and returns summary statistics.

It is an experimental feature because it turned out, that the mapping covers about 2000 elements (fields, subfields, indicators etc.), however on an average record there are max several hundred elements, which results that even in the best record has about 10-15% of the totality of the elements supporting a given function. So the tool doesn't show you exact numbers, and the scale is not 0-100 but 0-[best score] which is different for every catalogue.

The 12 functions:

10.1 Discovery functions

- search (DiscoverySearch): Search for a resource corresponding to stated criteria (i.e., to search either a single entity or a set of entities using an attribute or relationship of the entity as the search criteria).
- identify (DiscoveryIdentify): Identify a resource (i.e., to confirm that the entity described or located corresponds to the entity sought, or to distinguish between two or more entities with similar characteristics).
- select (DiscoverySelect): Select a resource that is appropriate to the user's needs (i.e., to choose an entity that meets the user's requirements with respect to content, physical format, etc., or to reject an entity as being inappropriate to the user's needs)
- obtain (DiscoveryObtain): Access a resource either physically or electronically through an online connection to a remote computer, and/or acquire a resource through purchase, licence, loan, etc.

10.2 Usage functions

- restrict (UseRestrict): Control access to or use of a resource (i.e., to restrict access to and/or use of an entity on the basis of proprietary rights, administrative policy, etc.).
- manage (UseManage): Manage a resource in the course of acquisition, circulation, preservation, etc.
- operate (UseOperate): Operate a resource (i.e., to open, display, play, activate, run, etc. an entity that requires specialized equipment, software, etc. for its operation).
- interpret (UseInterpret): Interpret or assess the information contained in a resource.

10.3 Management functions

- identify (ManagementIdentify): Identify a record, segment, field, or data element (i.e., to differentiate one logical data component from another).
- process (ManagementProcess): Process a record, segment, field, or data element (i.e., to add, delete, replace, output, etc. a logical data component by means of an automated process).
- sort (ManagementSort): Sort a field for purposes of alphabetic or numeric arrangement.
- display (ManagementDisplay): Display a field or data element (i.e., to display a field or data element with the appropriate print constant or as a tracing).

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.FunctionalAnalysis [options] <file>
```

with a bash script

```
./functional-analysis [options] <file>
```

or

```
catalogues/<catalogue>.sh functional-analysis
```

or

```
./qa-catalogue --params="[options]" functional-analysis
```

options: * general parameters

Output files: * `functional-analysis.csv`: the list of the 12 functions and their average count (number of support fields), and average score (percentage of all supporting fields available in the record) * `functional-analysis-mapping.csv`: the mapping of functions and data elements * `functional-analysis-histogram.csv`: the histogram of scores and count of records for each function (e.g. there are x number of records which has j score for function a)

11 Classification analysis

It analyses the coverage of subject indexing/classification in the catalogue. It checks specific fields, which might have subject indexing information, and provides details about how and which subject indexing schemes have been applied.

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.ClassificationAnalysis [options] <file>
Rscript scripts/classifications/classifications-type.R <output directory>
```

with a bash script

```
./classifications [options] <file>
Rscript scripts/classifications/classifications-type.R <output directory>
```

or

```
catalogues/[catalogue].sh classifications
```

or

```
./qa-catalogue --params="[options]" classifications
```

options:

- general parameters
- `-w`, `--emptyLargeCollectors`: empty large collectors periodically. It is a memory optimization parameter, turn it on if you run into a memory problem.

The output is a set of files:

- `classifications-by-records.csv`: general overview of how many records has any subject indexing
- `classifications-by-schema.csv`: which subject indexing schemas are available in the catalogues (such as DDC, UDC, MESH etc.) and where they are referred

- `classifications-histogram.csv`: a frequency distribution of the number of subjects available in records (x records have 0 subjects, y records have 1 subjects, z records have 2 subjects etc.)
- `classifications-frequency-examples.csv`: examples for particular distributions (one record ID which has 0 subject, one which has 1 subject, etc.)
- `classifications-by-schema-subfields.csv`: the distribution of subfields of those fields, which contains subject indexing information. It gives you a background that what other contextual information behind the subject term are available (such as the version of the subject indexing scheme)
- `classifications-collocations.csv`: how many record has a particular set of subject indexing schemes
- `classifications-by-type.csv`: returns the subject indexing schemes and their types in order of the number of records. The types are TERM_LIST (subtypes: DICTIONARY, GLOSSARY, SYNONYM_RING), METADATA_LIKE_MODEL (NAME_AUTHORITY_LIST, GAZETTEER), CLASSIFICATION (SUBJECT_HEADING, CATEGORIZATION, TAXONOMY, CLASSIFICATION_SCHEME), RELATIONSHIP_MODEL (THESAURUS, SEMANTIC_NETWORK, ONTOLOGY).

12 Authority name analysis

It analyses the coverage of authority names (persons, organisations, events, uniform titles) in the catalogue. It checks specific fields, which might have authority names, and provides details about how and which schemes have been applied.

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.AuthorityAnalysis [options] <file>
```

with a bash script

```
./authorities [options] <file>
```

or

```
catalogues/<catalogue>.sh authorities
```

or

```
./qa-catalogue --params="[options]" authorities
```

options:

- general parameters
- `-w`, `--emptyLargeCollectors`: empty large collectors periodically. It is a memory optimization parameter, turn it on if you run into a memory problem

The output is a set of files:

- `authorities-by-records.csv`: general overview of how many records has any authority names
- `authorities-by-schema.csv`: which authority names schemas are available in the catalogues (such as ISNI, Gemeinsame Normdatei etc.) and where they are referred
- `authorities-histogram.csv`: a frequency distribution of the number of authority names available in records (x records have 0 authority names, y records have 1 authority name, z records have 2 authority names etc.)

- `authorities-frequency-examples.csv`: examples for particular distributions (one record ID which has 0 authority name, one which has 1 authority name, etc.)
- `authorities-by-schema-subfields.csv`: the distribution of subfields of those fields, which contains authority names information. It gives you a background that what other contextual information behind the authority names are available (such as the version of the authority name scheme)

13 Field frequency distribution

This analysis reveals the relative importance of some fields. Pareto's distribution is a kind of power law distribution, and Pareto-rule of 80-20 rules states that 80% of outcomes are due to 20% of causes. In catalogue outcome is the total occurrences of the data element, causes are individual data elements. In catalogues some data elements occurs much more frequently then others. This analyses highlights the distribution of the data elements: whether it is similar to Pareto's distribution or not.

It produces charts for each document type and one for the whole catalogue showing the field frequency patterns. Each chart shows a line which is the function of field frequency: on the X-axis you can see the subfields ordered by the frequency (how many times a given subfield occurred in the whole catalogue). They are ordered by frequency from the most frequent top 1% to the least frequent 1% subfields. The Y-axis represents the cumulative occurrence (from 0% to 100%).

Before running it you should first run the completeness calculation.

With a bash script

```
catalogues/[catalogue].sh pareto
```

or

```
./qa-catalogue --params="[options]" pareto
```

options:

- general parameters

14 Generating cataloguing history chart

This analysis is based on Benjamin Schmidt's blog post A brief visual history of MARC cataloging at the Library of Congress. (Tuesday, May 16, 2017).

It produces a chart where the Y-axis is based on the "date entered on file" data element that indicates the date the MARC record was created (008/00-05), the X-axis is based on "Date 1" element (008/07-10).

Usage:

```
catalogues/[catalogue].sh marc-history
```

or

```
./qa-catalogue --params="[options]" marc-history
```

options:

- general parameters

15 Import tables to SQLite

This is just a helper function which imports the results of validation into SQLite3 database.

The prerequisite of this step is to run validation first, since it uses the files produced there. If you run validation with `catalogues/<catalogue>.sh` or `./qa-catalogue` scripts, this importing step is already covered there.

Usage:

```
catalogues/[catalogue].sh sqlite
```

or

```
./qa-catalogue --params="[options]" sqlite
```

options:

- general parameters

Output:

- `qa_catalogue.sqlite`: the SQLite3 database with 3 tables: `issue_details`, `issue_groups`, and `issue_summary`.

16 Indexing bibliographic records with Solr

Run indexer:

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.MarcToSolr [options] [file]
```

With script:

```
catalogues/[catalogue].sh all-solr
```

or

```
./qa-catalogue --params="[options]" all-solr
```

options:

- general parameters
- `-S <URL>`, `--solrUrl <URL>`: the URL of Solr server including the core (e.g. http://localhost:8983/solr/loc
- `-A`, `--doCommit`: send commits to Solr regularly (not needed if you set up Solr as described below)
- `-T <type>`, `--solrFieldType <type>`: a Solr field type, one of the predefined values. See examples below.
  - `marc-tags` - the field names are MARC codes
  - `human-readable` - the field names are Self Descriptive MARC code
  - `mixed` - the field names are mixed of the above (e.g. `245a_Title_mainTitle`)
- `-C`, `--indexWithTokenizedField`: index data elements as tokenized field as well (each bibliographical data elements will be indexed twice: once as a phrase (fields suffixed with `_ss`), and once as a bag of words (fields suffixed with `_txt`). [This parameter is available from v0.8.0]
- `-D <int>`, `--commitAt <int>`: commit index after this number of records [This parameter is available from v0.8.0]
- `-E`, `--indexFieldCounts`: index the count of field instances [This parameter is available from v0.8.0]
- `-G`, `--indexSubfieldCounts`: index the count of subfield instances [This parameter is available from v0.8.0]

- `-F`, `--fieldPrefix <arg>`: field prefix

The `./index` file (which is used by `catalogues/[catalogue].sh` and `./qa-catalogue` scripts) has additional parameters: * `-Z <core>`, `--core <core>`: The index name (core). If not set it will be extracted from the `solrUrl` parameter * `-Y <path>`, `--file-path <path>`: File path * `-X <mask>`, `--file-mask <mask>`: File mask * `-W`, `--purge`: Purge index and exit * `-V`, `--status`: Show the status of index(es) and exit * `-U`, `--no-delete`: Do not delete documents in index before starting indexing (be default the script clears the index)

16.1 Solr field names

QA catalogue builds a Solr index which contains a) a set of fixed Solr fields that are the same for all bibliographic input, and b) Solr fields that depend on the field names of the metadata schema (MARC, PICA, UNIMARC etc.) - these fields should be mapped from metadata schema to dynamic Solr fields by an algorithm.

16.1.1 Fixed fields

- `id`: the record ID. This comes from the identifier of the bibliographic record, so 001 for MARC21
- `record_sni`: the JSON representation of the bibliographic record
- `groupId_is`: the list of group IDs. The content comes from the data element specified by the `--groupBy` parameter split by commas (‘,’).
- `errorId_is`: the list of error IDs that come from the result of the validation.

16.1.2 Mapped fields

The mapped fields are Solr fields that depend on the field names of the metadata schema. The final Solr field follows the pattern:

Field prefix:

With `--fieldPrefix` parameter you can set a prefix that is applied to the variable fields. This might be needed because Solr has a limitation: field names start with a number can not be used in some Solr parameter, such as `fl` (field list selected to be retrieved from the index). Unfortunately bibliographic schemas use field names start with numbers. You can change a mapping parameter that produces a mapped value that resembles the BIBFRAME mapping of the MARC21 field, but not all field has such a human readable association.

Field suffixes:

- `*_sni`: not indexed, stored string fields – good for storing fields used for displaying information
- `*_ss`: not parsed, stored, indexed string fields – good for display and facets
- `*_tt`: parsed, not stored, indexed string fields – good for term searches (these fields will be availabe if `--indexWithTokenizedField` parameter is applied)
- `*_is`: parsed, not stored, indexed integer fields – good for searching for numbers, such as error or group identifiers (these fields will be availabe if `--indexFieldCounts` parameter is applied)

The mapped value

With `--solrFieldType` you can select the algorithm that generates the mapped value. Right now there are three formats: * `marc-tags` - the field names are MARC codes (`245$a` → `245a`) * `human-readable` - the field names are Self Descriptive MARC code (`245$a` → `Title_mainTitle`) * `mixed` - the field names are mixed of the above (e.g. `245a_Title_mainTitle`)

16.1.2.1 "marc-tags" format

```
"100a_ss":["Jung-Baek, Myong Ja"],
"100ind1_ss":["Surname"],
"245c_ss":["Vorgelegt von Myong Ja Jung-Baek."],
"245ind2_ss":["No nonfiling characters"],
"245a_ss":["S. Tret'jakov und China /"],
"245ind1_ss":["Added entry"],
"260c_ss":["1987."],
"260b_ss":["Georg-August-Universität Göttingen,"],
"260a_ss":["Göttingen :"],
"260ind1_ss":["Not applicable/No information provided/Earliest available publisher"],
"300a_ss":["141 p."],
```

16.1.2.2 "human-readable" format

```
"MainPersonalName_type_ss":["Surname"],
"MainPersonalName_personalName_ss":["Jung-Baek, Myong Ja"],
"Title_responsibilityStatement_ss":["Vorgelegt von Myong Ja Jung-Baek."],
"Title_mainTitle_ss":["S. Tret'jakov und China /"],
"Title_titleAddedEntry_ss":["Added entry"],
"Title_nonfilingCharacters_ss":["No nonfiling characters"],
"Publication_sequenceOfPublishingStatements_ss":["Not applicable/No information provided/Ear
"Publication_agent_ss":["Georg-August-Universität Göttingen,"],
"Publication_place_ss":["Göttingen :"],
"Publication_date_ss":["1987."],
"PhysicalDescription_extent_ss":["141 p."],
```

16.1.2.3 "mixed" format

```
"100a_MainPersonalName_personalName_ss":["Jung-Baek, Myong Ja"],
"100ind1_MainPersonalName_type_ss":["Surname"],
"245a_Title_mainTitle_ss":["S. Tret'jakov und China /"],
"245ind1_Title_titleAddedEntry_ss":["Added entry"],
"245ind2_Title_nonfilingCharacters_ss":["No nonfiling characters"],
"245c_Title_responsibilityStatement_ss":["Vorgelegt von Myong Ja Jung-Baek."],
"260b_Publication_agent_ss":["Georg-August-Universität Göttingen,"],
"260a_Publication_place_ss":["Göttingen :"],
"260ind1_Publication_sequenceOfPublishingStatements_ss":["Not applicable/No information prov
"260c_Publication_date_ss":["1987."],
"300a_PhysicalDescription_extent_ss":["141 p."],
```

A distinct project metadata-qa-marc-web, provides a web application that utilizes to build this type of Solr index in number of ways (a facetted search interface, term lists, search for validation errors etc.)

16.2 Index preparation

The tool uses different Solr indices (aka cores) to store information. In the following example we use `loc` as the name of our catalogue. There are two main indices: `loc` and `loc_dev`. `loc_dev` is the target of the index process, it will create it from scratch. During the proess `loc` is available and searchable. When the indexing has been successfully finished these two indices will be swaped, so the previous `loc` will become `loc_dev`, and the new index will be `loc`. The web user interface will always use the latest version (not the dev).

Besides these two indices there is a third index that contains different kind of results of the analyses. At the time of writing it contains only the results of validation, but later it will cover other information as well. It can be set by the following parameter:

- `-4`, `--solrForScoresUrl <arg>`: the URL of the Solr server used to store scores (it is populated in the `validate-sqlite` process which runs after validation)

During the indexing process the content of this index is meged into the `_dev` index, so after a successfull end of the process this index is not needed anymore.

In order to make the automation easier and still flexible there are some an auxilary commands:

- `./qa-catalogue prepare-solr`: created these two indices, makes sure that their schemas contain the necessary fields
- `./qa-catalogue index`: runs the indexing process
- `./qa-catalogue postprocess-solr`: swap the two Solr cores (and _dev)
- `./qa-catalogue all-solr`: runs all the three steps

If you would like to maintain the Solr index yourself (e.g. because the Solr instance wuns in a cloud environment), you should skip `prepare-solr` and `postprocess-solr`, and run only `index`. For maintaining the schema you can find a minimal viable schema among the test resources

You can set autocommit the following way in `solrconfig.xml` (inside Solr):

```
<autoCommit>
  <maxTime>${solr.autoCommit.maxTime:15000}</maxTime>
  <maxDocs>5000</maxDocs>
  <openSearcher>true</openSearcher>
</autoCommit>
...
<autoSoftCommit>
  <maxTime>${solr.autoSoftCommit.maxTime:-1}</maxTime>
</autoSoftCommit>
```

It needs if you choose to disable QA catalogue to issue commit messages (see `--commitAt` parameter), which makes indexing faster.

In schema.xml (or in Solr web interface) you should be sure that you have the following dynamic fields:

```
<dynamicField name="*_ss" type="strings" indexed="true" stored="true"/>
<dynamicField name="*_tt" type="text_general" indexed="true" stored="false"/>
<dynamicField name="*_is" type="pints" indexed="true" stored="true" />
<dynamicField name="*_sni" type="string_big" docValues="false" multiValued="false" indexed="

<copyField source="*_ss" dest="_text_"/>
```

or use Solr API:

```
NAME=dnb
SOLR=http://localhost:8983/solr/$NAME/schema

// add copy field
curl -X POST -H 'Content-type:application/json' --data-binary '{
  "add-dynamic-field":{
     "name":"*_sni",
     "type":"string",
     "indexed":false,
     "stored":true}
}' $SOLR

curl -X POST -H 'Content-type:application/json' --data-binary '{
  "add-copy-field":{
     "source":"*_ss",
     "dest":["_text_"]}
}' $SOLR
...
```

See the solr-functions file for full code.

QA catalogue has a helper scipt to get information about the status of Solr index (Solr URL, location, the list of cores, number of documents, size in the disk, and last modification):

```
$ ./index --status
Solr index status at http://localhost:8983
Solr directory: /opt/solr-9.3.0/server/solr

core                 | location        | nr of docs |       size |       last modified
.................... | ............... | .......... | .......... | ...................
nls                  | nls_1           |     403946 | 1002.22 MB | 2023-11-25 21:59:39
nls_dev              | nls_2           |     403943 |  987.22 MB | 2023-11-11 15:59:49
nls_validation       | nls_validation  |     403946 |   17.89 MB | 2023-11-25 21:35:44
yale                 | yale_2          |    2346976 |    9.51 GB | 2023-11-11 13:12:35
yale_dev             | yale_1          |    2346976 |    9.27 GB | 2023-11-11 10:58:08
```

17 Export mapping table

17.1 to Avram JSON

Some background info: MARC21 structure in JSON.

Usage:

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.utils.MappingToJson [options] > avram-schema.json
```

or

```
./qa-catalogue --params="[options]" export-schema-files
```

options:

- general parameters
- `-c`, `--withSubfieldCodelists`: with subfield codelists
- `-s`, `--withSelfDescriptiveCode`: with self-descriptive codes
- `-t <type>`, `--solrFieldType <type>`: type of Solr fields, could be one of `marc-tags`, `human-readable`, or `mixed`
- `-f`, `--withFrbrFunctions`: with FRBR functions (see Tom Delsey: Functional analysis of the MARC 21 bibliographic and holdings formats. Tech. report, 2nd revision. Library of Congress, 2003.)
- `-l`, `--withComplianceLevel`: with compliance levels (national, minimal) (see National Level Full and Minimal Requirements. Library of Congress, 1999.)

An example output:

```
...
"010":{
  "tag":"010",
  "label":"Library of Congress Control Number",
  "url":"https:\/\/www.loc.gov\/marc\/bibliographic\/bd010.html",
  "repeatable":false,
  "compilance-level":{
    "national":"Mandatory if applicable",
    "minimal":"Mandatory if applicable"
  },
  "indicator1":null,
  "indicator2":null,
  "subfields":{
    "a":{
      "label":"LC control number",
      "repeatable":false,
      "frbr-functions":[
        "Data Management\/Identify",
        "Data Management\/Process"
      ],
      "compilance-level":{
        "national":"Mandatory if applicable",
        "minimal":"Mandatory if applicable"
      }
    },
    ...
  }
},
"013":{
  "tag":"013",
  "label":"Patent Control Information",
  "url":"https:\/\/www.loc.gov\/marc\/bibliographic\/bd013.html",
  "repeatable":true,
  "compilance-level":{"national":"Optional"},
  "indicator1":null,
  "indicator2":null,
  "subfields":{
    ...
    "b":{
      "label":"Country",
      "repeatable":false,
      "codelist":{
        "name":"MARC Code List for Countries",
        "url":"http:\/\/www.loc.gov\/marc\/countries\/countries_code.html",
        "codes":{
          "aa":{"label":"Albania"},
          "abc":{"label":"Alberta"},
          "-ac":{"label":"Ashmore and Cartier Islands"},
          "aca":{"label":"Australian Capital Territory"},
          ...
        },
        ...
      },
    },
    ...
  }
},
...
```

The script version generates 3 files, with different details: * `avram-schemas/marc-schema.json` * `avram-schemas/marc-schema-with-solr.json` * `avram-schemas/marc-schema-with-solr-and-extensions`

To validate these files install the Avram reference implementation in Node with `npm ci` and run:

```
./avram-schemas/validate-schemas
```

17.2 to HTML

To export the HTML table described at Self Descriptive MARC code

```
java -cp $JAR de.gwdg.metadataqa.marc.cli.utils.MappingToHtml > mapping.html
```

18 Shacl4Bib

since v0.7.0. Note: This is an experimental feature.

The Shapes Constraint Language (SHACL) is a formal language for validating Resource Description Framework (RDF) graphs against a set of conditions (expressed also in RDF). Following this idea and implementing a subset of the language, the Metadata Quality Assessment Framework provides a mechanism to define SHACL-like rules for data sources in non-RDF based formats, such as XML, CSV and JSON (SHACL validates only RDF graphs). Shacl4Bib is the extension enabling the validation of bibliographic records. The rules can be defined either with YAML or JSON configuration files or with Java code. SCHACL uses RDF notation to specify or "address" the data element about which the constraints are set. Shacl4Bib supports Carsten Klee's MARCspec for MARC records, and PICApath for PICA. You can find more information and full definition of the implemented subset of SHACL here: Defining schema with a configuration file

Parameters:

- `-C <file>`, `--shaclConfigurationFile <file>`: specify the SHACL like configuration file
- `-O <file>`, `--shaclOutputFile <file>`: output file (default: `shacl4bib.csv`)
- `-P <type>`, `--shaclOutputType <type>`: specify what the output files should contain. Possible values:
  - `STATUS`: status only (default), where the following values appear:
    - `1` the criteria met,
    - `0` the criteria have not met,
    - `NA`: the data element is not available in the record),
  - `SCORE`: score only. Its value is calculated the following way:
    - if the criteria met it returns the value of `successScore` property (or 0 if no such property)
    - if the criteria have not met, it returns the value of `failureScore` property (or 0 if no such property)
  - `BOTH`: both status and score

Here is a simple example for setting up rules against a MARC subfield:

```yaml
format: MARC
fields:
- name: 040$a
  path: 040$a
  rules:
  - id: 040$a.minCount
    minCount: 1
  - id: 040$a.pattern
    pattern: ^BE-KBR00
```

- `format` represents the format of the input data. It can be either `MARC` or `PICA`
- `fields`: the list of fields we would like to investigate. Since it is a YAMPL example, the `-` and indentation denotes child elements. Here there is only one child, so we analyse here a single subfield.
- `name` is how the data element is called within the rule set. It could be a machine or a human readable string.
- `path` is the "address" of the metadata element. It should be expressed in an addressing language such as MARCSpec or PICAPath (040$a contains the original cataloging agency)
- `rules`: the parent element of the set of rules. Here we have two rules.
- `id`: the identifier of the rule. This will be the header of the column in CSV, and it could be references elsewhere in the SHACL configuration file.
- `mintCount`: this specify the minimum number of instances of the data element in the record
- `pattern`: a regular expression which should match the values of all instances of the data element

The output contains an extra column, the record identifier, so it looks like something like this:

```
id,040$a.minCount,040$a.pattern
17529680,1,1
18212975,1,1
18216050,1,1
18184955,1,1
18184431,1,1
9550740,NA,NA
19551181,NA,NA
118592844,1,1
18592704,1,1
18592557,1,1
```

A Where can I get MARC records?

Here is a list of data sources I am aware of so far:

A.1 United States of America

- Library of Congress — https://www.loc.gov/cds/products/marcDist.php. MARC21 (UTF-8 and MARC8 encoding), MARCXML formats, open access. Alternative access point: https://www.loc.gov/collections/selected-datasets/?fa=contributor:library+of+congress.+cataloging+distribution+service.
- Harvard University Library — https://library.harvard.edu/open-metadata. MARC21 format, CC0. Institution specific features are documented here
- Columbia University Library — https://library.columbia.edu/bts/clio-data.html. 10M records, MARC21 and MARCXML format, CC0.
- University of Michigan Library — https://www.lib.umich.edu/open-access-bibliographic-records. 1,3M records, MARC21 and MARCXML formats, CC0.
- University of Pennsylvania Libraries — https://www.library.upenn.edu/collections/digital-projects/open-data-penn-libraries. Two datasets are available:

  1. Catalog records created by Penn Libraries 572K records, MARCXML format, CC0,
  2. Catalog records derived from other sources, 6.5M records, MARCXML format, Open Data Commons ODC-BY, use in accordance with the OCLC community norms.

- Yale University — https://guides.library.yale.edu/c.php?g=923429. Three datasets are available:

  1. Yale-originated records: 1.47M records, MARC21 format, CC0
  2. WorldCat-derived records: 6.16M records, MARC21 format, ODC-BY
  3. Other records (MARC21), independent of Yale and WorldCat, where sharing is permitted. 404K records, MARC21 format.

- National Library of Medicine (NLM) catalogue records — https://www.nlm.nih.gov/databases/download/catalog.html. 4.2 million records, NLMXML, MARCXML and MARC21 formats. NLM Terms and Conditions

A.2 Germany

- Deutsche Nationalbibliothek — https://www.dnb.de/DE/Professionell/Metadatendienste/Datenbezug/Gesamtabzuege/gesamtabzuege_node.html (note: the records are provided in utf-8 decomposed). 23.9M records, MARC21 and MARCXML format, CC0.
- Bibliotheksverbundes Bayern — https://www.bib-bvb.de/web/b3kat/open-data. 27M records, MARCXML format, CC0.
- Leibniz-Informationszentrum Technik und Naturwissenschaften Universitätsbibliothek (TIB) — https://www.tib.eu/de/forschung-entwicklung/entwicklung/open-data. (no download link, use OAI-PMH instead) Dublin Core, MARC21, MARCXML, CC0.
- K10plus-Verbunddatenbank (K10plus union catalogue of Bibliotheksservice-Zentrum Baden Würtemberg (BSZ) and Gemensamer Bibliotheksverbund (GBV)) — https://swblod.bsz-bw.de/od/. 87M records, MARCXML format, CC0.

A.3 Others

- Universiteitsbibliotheek Gent — https://lib.ugent.be/info/exports. Weekly data dump in Aleph Sequential format. It contains some Aleph fields above the standard MARC21 fields. ODC ODbL.
- Toronto Public Library — https://opendata.tplcs.ca/. 2.5 million MARC21 records, Open Data Policy
- Répertoire International des Sources Musicales — https://opac.rism.info/index.php?id=8&id=8&L=1. 800K records, MARCXML, RDF/XML, CC-BY.
- ETH-Bibliothek (Swiss Federal Institute of Technology in Zurich) — http://www.library.ethz.ch/ms/Open-Data-an-der-ETH-Bibliothek/Downloads. 2.5M records, MARCXML format.
- British library — http://www.bl.uk/bibliographic/datafree.html#m21z3950 (no download link, use z39.50 instead after asking for permission). MARC21, usage will be strictly for non-commercial purposes.
- Talis — https://archive.org/details/talis_openlibrary_contribution. 5.5 million MARC21 records contributed by Talis to Open Library under the ODC PDDL.
- Oxford Medicine Online (the catalogue of medicine books published by Oxford University Press) — https://oxfordmedicine.com/page/67/. 1790 MARC21 records.
- Fennica — the Finnish National Bibliography provided by the Finnish National Library — http://data.nationallibrary.fi/download/. 1 million records, MARCXML, CC0.
- Biblioteka Narodawa (Polish National Library) — https://data.bn.org.pl/databases. 6.5 million MARC21 records.
- Magyar Nemzeti Múzeum (Hungarian National Library) — https://mnm.hu/hu/kozponti-konyvtar/nyilt-bibliografiai-adatok, 67K records, MARC21, HUNMARC, BIBFRAME, CC0

- University of Amsterdam Library — https://uba.uva.nl/en/support/open-data/data-sets-and-publication-channels/data-sets-and-publication-channels.html, 2.7 million records, MARCXML, PDDL/ODC-BY. Note: the record for books are not downloadable, only other document types. One should request them via the website.
- Portugal National Library — https://opendata.bnportugal.gov.pt/downloads.htm. 1.13 million UNIMARC records in MARCXML, RDF XML, JSON, TURTLE and CSV formats. CC0
- National Library of Latvia National bibliography (2017–2020) — https://dati.lnb.lv/. 11K MARCXML records.

Thanks, Johann Rolschewski, Phú, and Hugh Paterson III for their help in collecting this list! Do you know some more data sources? Please let me know.

There are two more datasource worth mention, however they do not provide MARC records, but derivatives:

- Linked Open British National Bibliography 3.2M book records in N-Triplets and RDF/XML format, CC0 license
- Linked data of Bibliothèque nationale de France. N3, NT and RDF/XML formats, Licence Ouverte/Open Licence

B Handling MARC versions

The tool provides two levels of customization:

- project specific tags can be defined in their own Java package, such as these classes for Gent data: de.gwdg.metadataqa.marc.definition.tags.genttags
- for existing tags one can use the API described below

The different MARC versions has an identifier. This is defined in the code as an enumeration:

```
public enum MarcVersion {
  MARC21("MARC21", "MARC21"),
  DNB("DNB", "Deutsche Nationalbibliothek"),
  OCLC("OCLC", "OCLC"),
  GENT("GENT", "Universiteitsbibliotheek Gent"),
  SZTE("SZTE", "Szegedi Tudományegyetem"),
  FENNICA("FENNICA", "National Library of Finland")
  ;
  ...
}
```

When you add version specific modification, you have to use one of these values.

1. Defining version specific indicator codes:

```
Indicator::putVersionSpecificCodes(MarcVersion, List<Code>);
```

Code is a simple object, it has two property: code and label.

example:

```
public class Tag024 extends DataFieldDefinition {
   ...
   ind1 = new Indicator("Type of standard number or code")
             .setCodes(...)
             .putVersionSpecificCodes(
                 MarcVersion.SZTE,
                 Arrays.asList(
                    new Code(" ", "Not specified")
                 )
             )
   ...
}
```

2. Defining version specific subfields:

```
DataFieldDefinition::putVersionSpecificSubfields(MarcVersion, List<SubfieldDefinition>)
```

SubfieldDefinition contains a definition of a subfield. You can construct it with three String parameters: a code, a label and a cardinality code which denotes whether the subfield can be repeatable ("R") or not ("NR").

example:

```
public class Tag024 extends DataFieldDefinition {
   ...
   putVersionSpecificSubfields(
      MarcVersion.DNB,
      Arrays.asList(
         new SubfieldDefinition("9", "Standardnummer (mit Bindestrichen)", "NR")
      )
   );
}
```

3. Marking indicator codes as obsolete:

```
Indicator::setHistoricalCodes(List<String>)
```

The list should be pairs of code and description.

```
public class Tag082 extends DataFieldDefinition {
   ...
   ind1 = new Indicator("Type of edition")
              .setCodes(...)
              .setHistoricalCodes(
                 " ", "No edition information recorded (BK, MU, VM, SE) [OBSOLETE]",
                 "2", "Abridged NST version (BK, MU, VM, SE) [OBSOLETE]"
              )
   ...
}
```

4. Marking subfields as obsolete:

```
DataFieldDefinition::setHistoricalSubfields(List<String>);
```

The list should be pairs of code and description.

[Java} public class Tag020 extends DataFieldDefinition { ... setHistoricalSubfields("b", "Binding information (BK, MP, MU) [OBSOLETE]"); }

If you create new a package for the new MArc version, you should register it to several places:

a. add a case into `src/main/java/de/gwdg/metadataqa/marc/Utils.java`:

```
case "zbtags":      version = MarcVersion.ZB;      break;
```

b. add an item into enumeration at `src/main/java/de/gwdg/metadataqa/marc/definition/tags/TagCate`

```
ZB(23, "zbtags", "ZB", "Locally defined tags of the Zentralbibliothek Zürich", false),
```

c. modify the expected number of data elements at `src/test/java/de/gwdg/metadataqa/marc/utils/Data`

```
assertEquals( 215, statistics.get(DataElementType.localFields));
```

d. … and a `src/test/java/de/gwdg/metadataqa/marc/utils/MarcTagListerTest.java`:

```
assertEquals( 2, (int) versionCounter2.get(MarcVersion.ZB));
assertEquals( 2, (int) versionCounter.get("zbtags"));
```

C Institutions which reportedly use this tool

- Universiteitsbibliotheek Gent, Gent, Belgium
- Biblioteksentralen, Oslo, Norway
- Deutsche Digitale Bibliothek, Frankfurt am Main/Berlin, Germany
- British Library, London/Boston Spa, United Kingdom
- Országgyűlési Könyvtár, Budapest, Hungary
- Studijní a vědecká knihovna Plzeňského kraje, Plzeň, Czech Republic
- Royal Library of Belgium (KBR), Brussels, Belgium
- Gemeinsamer Bibliotheksverbund (GBV), Göttingen, Germany
- Binghampton University Libraries, Binghampton, NY, USA
- Zentralbibliothek Zürich, Zürich, Switzerland

If you use this tool as well, please contact me: pkiraly (at) gwdg (dot) de. I really like to hear about your use case and ideas.

D supporters-and-sponsors

- Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG): Hardware, time for research
- Gemeinsamer Bibliotheksverbund (GBV): contracting for feature development
- Royal Library of Belgium (KBR): contracting for feature development
- JetBrains s.r.o.: IntelliJ IDEA development tool community licence

E Special build process

“deployment” build (when deploying artifacts to Maven Central)

```
mvn clean deploy -Pdeploy
```

F Build Docker image

Build and test

```
# create the Java library
mvn clean install
# create the docker base image
docker compose -f docker/build.yml build app
```

The `docker compose build` command has multiple `--build-arg` arguments to override defaults:

- `QA_CATALOGUE_VERSION`: the QA catalogue version (default: `0.7.0`, current development version is `0.8.0-SNAPSHOT`)
- `QA_CATALOGUE_WEB_VERSION`: it might be a released version such as `0.7.0`, or `main` (default) to use the main branch, or `develop` to use the develop branch.
- `SOLR_VERSION`: the Apache Solr version you would like to use (default: `8.11.1`)
- `SOLR_INSTALL_SOURCE`: if its value is `remote` docker will download it from http://archive.apache.org/. If its value is a local path points to a previously downloaded package (named as `solr-${SOLR_VERSION}.zip` up to version 8.x.x or `solr-${SOLR_VERSION}.tgz` from version 9.x.x) the process will copy it from the host to the image file. Depending on the internet connection, download might take a long time, using a previously downloaded package speeds the building process. (Note: it is not possible to specify files outside the current directory, not using symbolic links, but you can create hard links - see an example below.)

Using the current developer version:

```
docker compose -f docker/build.yml build app \
  --build-arg QA_CATALOGUE_VERSION=0.8.0-SNAPSHOT \
  --build-arg QA_CATALOGUE_WEB_VERSION=develop \
  --build-arg SOLR_VERSION=8.11.3
```

Using a downloaded Solr package:

```
# create link temporary
mkdir download
ln ~/Downloads/solr/solr-8.11.3.zip download/solr-8.11.3.zip
# run docker
docker compose -f docker/build.yml build app \
  --build-arg QA_CATALOGUE_VERSION=0.8.0-SNAPSHOT \
  --build-arg QA_CATALOGUE_WEB_VERSION=develop \
  --build-arg SOLR_VERSION=8.11.3 \
  --build-arg SOLR_INSTALL_SOURCE=download/solr-8.11.3.zip
# delete the temporary link
rm -rf download
```

Then start the container with environment variable `IMAGE` set to `metadata-qa-marc` and run analyses as described above.

For maintainers only:

Upload to Docker Hub:

```
docker tag metadata-qa-marc:latest pkiraly/metadata-qa-marc:latest
docker login
docker push pkiraly/metadata-qa-marc:latest
```

Cleaning before and after:

```
# stop running container
docker stop $(docker ps --filter name=metadata-qa-marc -q)
# remove container
docker rm $(docker ps -a --filter name=metadata-qa-marc -q)
# remove image
docker rmi $(docker images metadata-qa-marc -q)
# clear build cache
docker builder prune -a -f
```