QC Tool implementation proposal

Revision 3 by GISAT, 27th February 2018

# Current state

There is a web based tool available (CLCQC Tool at [http://clcqc.gisat.cz](http://clcqc.gisat.cz/)) for checking the quality of the deliveries (Semantic Quality Check) of the Corine Land Cover data products. This tool is at the moment a web application with server-side checks running using the functionalities of the ArcGIS server suite. The quality control is in production use for 2012 and 2018 CLC mapping campaign and it allows the users to verify their deliveries in DIY style before actually delivering them to the EEA for final control and acceptance. The GitHub repository of the old version of the tool is private and available at the URL: <https://github.com/gisat/clcqc>

Originally, there was an idea to extend existing CLCQC Tool for semantic checks needs of other Copernicus products and as such specific profiles (checks, parameters, exceptions) has been prepared for all products based on production specification. Finally, the overall requirements (see below) and related implementation approach for new QC tool is different. While, some elements (e.g. products profiles for Copernicus products) can be re-used, new tools will be based on different open source software framework. The new QC tool will be developed on EEA’s GitHub repository at the URL:

<https://github.com/orgs/eea/teams/copernicus_quality_tools/>

[Branches and tags will be used during the development with a regime of fast and incremental deployment (2 weeks cycle), in order to provide quick feedbacks and avoid monolithic implementation in the wrong direction. The deployed releases labeled by a version number tag will be published on DockerHub together with an installation file for quick deployment.](https://github.com/orgs/eea/teams/copernicus/)

# The requirements

* The web-based tool will be deployed at the EEA infrastructure. Currently it is deployed at Gisat infrastructure
* It will be possible to run the tool locally at SP side with a little effort from the perspective of maintenance
* The checks in the server deployment and at the local user’s work stations deployments will be the same (the local deployment will eventually not have the DB managing user and check history part)
* For easy deployment and maintenance the application will be delivered as Docker container
* The tools for checking the quality will be written using open source solutions.
* The computing resources required for the checks will be reasonably limited

# Proposal

## Architecture

The architecture section discussed four important design questions:

1. Implementation of the checks and their dependencies
2. User interaction
3. Data storage / checking environment
4. Packaging and distribution of the whole solution

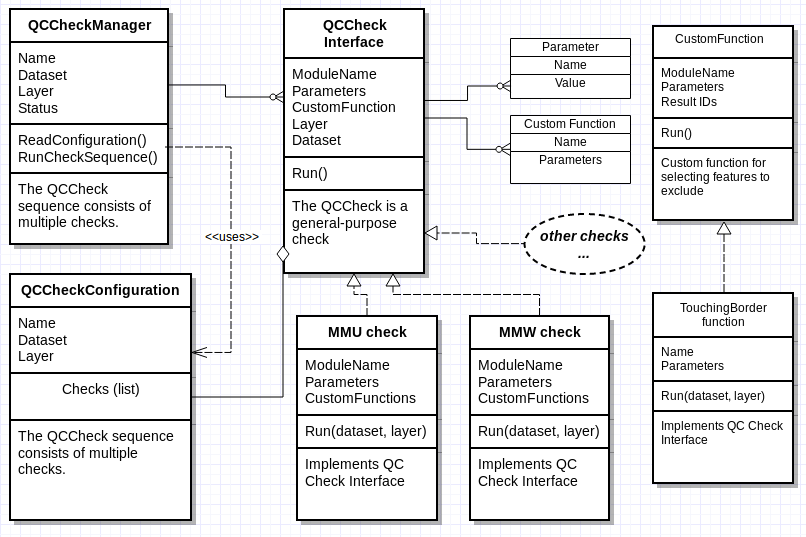
## Checks implementation

The most important question in the checks implementation is how to design a generalized checking infrastructure. We need to consider that each checked dataset may have multiple layers and each QC checking sequence of a dataset consists of multiple generic QC checks. Therefore we have to define generic (but extendable) set of QC checks and then for each dataset, we must specify:

* which QC checks are required for this dataset and its layers?
* What is the sequence (order) of the QC checks?
* What are the dataset-specific parameter values, constraints and exceptions of each QC check in the sequence?

To make the solution flexible we propose a QC Check interface with configurable parameters. Each check module (such as the MMU check or the MMW check) will implement the QC Check interface. An example of a configurable parameter is the area parameter in the MMU check.

A QC check may also refer to a special custom function parameter in order to exclude some features or raster cells from the checking. This custom function also has configurable parameters. The output of the custom function is a list of feature IDs or a list of raster cell values. The features of the dataset whose IDs are not in the custom function output are excluded from the checking. An example of a custom function could be a function that returns all features touching a predefined boundary feature. The setup is shown in the class diagram. Each concrete check module must implement the QC Check interface (see Figure 1).

Figure 1: Class diagram of the QC check implementation

In figure 1 the CheckManager reads the configuration which specifies a list of checks to be run for a given dataset. For each check in the configuration, the manager uses the ModuleName setting to locate the executable script. The values of the parameters (if any) are also read from the configuration. The QA infrastructure will be designed with scalability in mind. The incoming requests to the CheckManager will be received through an Application Programming Interface (API). The API will contain methods to:

* Start checking a dataset (uploaded dataset files, check configuration file)
* Get Check Status (percent completed)
* Get Check Result (returns the checking result report together with geospatial vector data with locations of errors)

We suggest adopting the Open Geospatial Consortium (OGC) Web processing service as the API as it provides standard methods for starting a long-running process and querying its status. However, a custom API with the methods described above could also be used.

Invoking the CheckManager via a web API will make it possible to have multiple CheckManager instances on multiple servers or cloud-based virtual machines managed by a load balancer. In case of multiple incoming requests a load balancer will manage the request queue, delegate each request to an appropriate CheckManager instance and launch new instances if necessary. The load balancing is described in more detail in the Packaging and Distribution Section.

We propose to store the check configuration of each dataset in a JSON configuration file. The following JSON code illustrates an example of a configuration file:

{

"dataSet":"CLC",

"layer":"CLC\_change",

"checks":[

{

"check":"topology\_check"

}, {

"check":"technical\_changes\_check",

"parameters":[

{

"name":"status\_1",

"value":"Code\_06"

},

{

"name":"status\_2",

"value":"Code\_12"

}

]

}, {

"check":"MMU\_check",

"parameters":[

{

"name":"area\_ha",

"value":5

}

],

"custom\_function":[

{

"name":"touching\_boundary",

"parameters":"[{boundary\_layer: mapping\_area\_boundary}]",

}

]

}

]

}

In the example dataset-specific configuration above, there is a sequence of three QC checks: topology check, technical changes check, and MMU check. The topology check does not have any parameters. The technical changes check has two parameters (status\_1 and status\_2) which can be configured for a specific dataset. The MMU check has one parameter (the area\_ha) and a "touching\_boundary" custom function. The selection parameter uses a spatial query expression to select polygons which do not touch the mapping area boundary.

Note: For more complex selection or subsetting, we may need to define a custom function. The custom function will return a list of feature IDs that are excluded from the check. For example in the MMU case there can be a custom function that returns IDs of features touching the mapping area boundary.

See section ’Checks’ for more information about generic checks defined as well as particular parameters related to individual products.

## Data Storage / checking environment

The second important decision is the file format of the checked datasets. For vector datasets we decided to import the data to a spatial database. Specifically we propose to use the PostgreSQL with PostGIS extension. The main advantage of PostGIS is the ease of the implementation and the availability of plenty of standard functions for working with the spatial vector data as well as high quality spatial indexing. The advantages in the speed due to effective indexing are probably enough to offset resource consumption for the import of the data into the database. The only obvious drawback then is the used disk space. Also handling the case of an import failure (for example in the case of an empty GDB file or a GDB file with rasters instead of vectors) will be required. In this case the tool will reject the delivery and send an error message with reason of data import failure.

For raster datasets (GeoTiff files) we are planning to run the checks directly on the raster files using tools such as GDAL. .

## Front end application

Our proposal is to design a new front end application tailored to the needs of running checks for more products. We conclude that it is more effective to develop a new front end application for the following reasons: (1) The old front end application is built with the Sencha ExtJS library which is not open-source and has a restrictive license for redistribution, (2) The old front end application needs a special MongoDB database which means extra software dependencies. The new front end application will have a front end user interface written in React.js consisting of three main forms: user management form, dataset management form, and QC check management form. In the user management there will be multiple user roles (admin, regular user, guest). The admin will be able to manage user accounts and give user permissions to upload a dataset or run checks on a dataset.

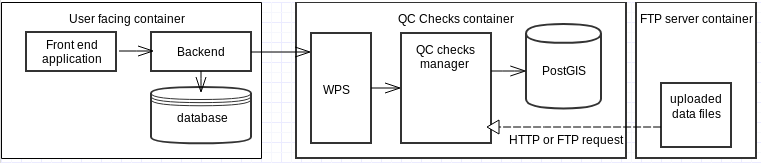
The dataset management page form will allow the user to upload a dataset into the system. The upload will be implemented in the background as FTPS file upload. In case of no ssl available (deployment on a local user work station), a self signed certificate will be provided. The dataset management page will also show an overview of previously uploaded datasets.

The QC check management form will allow the authorized user to launch a series of QC checks on a selected dataset. Once a series of QC checks is completed, the status of each check will be shown. There will also be a table showing the history of past checks and their outcomes to manage past sessions.

The user facing application will have a Node.js backend communicating with a Postgres database. This database will store the user information, current session information, and the history of past checks. The results of the checks will be saved in machine readable format (json or xml) in the database. For some of the checks a spatial vector result (Geojson) with locations of non-compliant polygons will also be saved. The database will also contain tables with supported dataset product types (such as CLC dataset product type, UA dataset product type) and the dataset-specific QC check configurations (order of checks and values of their parameters) as described in the previous checks implementation section. These configurations will be saved in the database when the application is installed (they will be loaded from a special Docker image containing the configuration data)

## Packaging and distribution

For packaging and distribution of the solution two key factors need to be considered: (1) ease of installation on the local user PC and (2) scalability and load balancing on the production servers. To ensure effective maintenance and scalability, we propose to distribute the application as a Docker-stack of containers, using separate docker containers for the user-facing application and for the QC checking service.



The first container contains the client-side front end web application, the backend application, and the application database.

The second docker container is a load balancer. The load balancer accepts incoming web requests for running QC checks. It manages a queue of the requests and delegates the requests to one or more QC checking containers.

The QC checking docker container contains the QC CheckManager and the PostGIS database for imported vector datasets. The QC Check container runs as a web service. We propose having an Application Programming Interface (API) of the web service with a standard protocol such as the Web Processing Service (WPS). The WPS API allows launching a sequence of checks, monitoring the checking process status, and obtaining the check result for a dataset via simple HTTP requests. In the HTTP request two parameters are passed to the WPS: The URL of the dataset file and the checking configuration for the dataset. The advantage of using a WPS service is a well-defined communication interface documented by the Open Geospatial Consortium (OGC). Alternatives to using WPS include (a) defining a custom API (REST API) and (b) using docker environment communication channels such as docker ip and docker swarm. However, the docker swarm alternative is not as well suited to deploying the solution on multiple servers in the cloud environment.

Another separate container is a FTP / FTPS server connected to a docker volume. The GDB and GeoTiff data files are uploaded to the FTP server from the user-facing application.

## In addition to software containers, data containers will also be deployed. The first data container will contain the dataset-specific QC check configuration files. The second data container will contain licensed data files such as AOI files.

An example deployment with multiple docker containers is shown in Figure 3. There are four composed containers: One user-facing container, two QC checks containers, and one FTP server container. The user-facing container contains the web application. The QC checks containers are responsible for running the sequences of QC checks. This is where the heavy lifting takes place. It is possible to deploy multiple QC check containers if there is a Load Balancer in front of them to decide which of them will be actually used. On technological level the docker containers will be provided via the Docker Hub and there will be Docker Compose scripts for the full deployment on a server or on a local PC. The docker containers with licensed data files (AOI files) will be published on a password-protected private Docker hub repository.

# Checks

As mentioned proposed approach builds on experience from semantic check activities carried out for particular Copernicus products delivery in previous years. Delivery workflows and semantic check arrangements are currently (historically) implemented differently for different products. Nevertheless, there is no reason behind and there is an opportunity for overall standardization, harmonization and streamlining. Focus is on checks which can be automated. Look-and-feel or plausibility type of checks are out of scope of this document.

Proposed approach is based on (extendable) set of generic checks which will be implemented in generic way with the least possible dependencies (see previous chapter) and then used for semantic checks of particular product from Copernicus portfolio (again extendable in future) with specific product-related parameters and exceptions definition.

## Generic checks

Tables 1 and 2 below includes proposed list of generic semantic checks applicable for all datasets. Checks are divided as applicable for vector and raster based products. For now some checks between vector and raster datasets may overlap, but in general they will be implemented in different OSS environment. Particular product relevant application is discussed later.

## Vector data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Check** | | **Open source tool or software** | **Remarks** | |
| V1 | File format |  | | Accepted format > currently shape, file gdb, personal gdb |
| V2 | File naming convention | Python | | Standard file name |
| V3 | Attribute table structure | ogrinfo | | The control of the existence of the attribute and the format |
| V4 | CRS | ogrinfo | | CRS definition. Control via the EPSG code |
| V5 | Unique identifier | SQL in PostGIS | | The database must contain field which is unique. |
| V6 | Valid codes | SQL in PostGIS | | Checks against a group of codes. This generic check can be used for various check lists and/or combined with exceptions as e.g.   * List of valid codes * List of non-probable code e.g. currently applicable in context of changes. * List of technical changes e.g. currently applicable in context of changes. |
|  |  |  | |  |
| V8 | Multipart polygons | SQL in PostGIS | | There can be no multipart geometry in the dataset. |
|  |  |  | |  |
|  |  |  | |  |
| V11 | MMU | SQL in PostGIS | | Class dependent minimum mapping size threshold. Exceptions can apply as e.g. boundary polygons |
| V12 | MMW | PostGIS (ApproximateMedialAxis) | | Minimum mapping width threshold. Can be solved by calculating skeleton of polygon or inner buffer. |
| V13 | Topology | SQL in Postgiss | | Topology is being controlled inside the layer not throughout the layers. Several sub-checks included (can be decomposed) as  - no overlapping features  - no gaps in data  - etc. |
| V14 | Neighbouring polygons with the same code | SQL in Postgis, | | Neighbouring polygons with the same code cannot exist |
| V15 | Metadata | Web service API http request | | INSPIRE Geoportal Metadata Validator Web Service: <http://inspire-geoportal.ec.europa.eu/validator2/html/usingaswebservice.html> |
| V16 | Completeness | PostGIS | | Complete products defined AOI (or AOI with buffer) must be covered by data. It requires AOI definition file availability and accessibility (may be subject of IPR). |

*Table 1 – Generic vector data checks*

## Raster data

|  |  |  |  |
| --- | --- | --- | --- |
| **Check** | | **Open source tool or software** | **Remarks** |
| R1 | File format | gdalinfo | Accepted format > currently GeoTIFF |
| R2 | File naming convention | Python | Standard file name |
| R3 | Attribute table structure | gdalinfo | Control the existence of the attribute table and its structure |
| R4 | CRS | gdalinfo | Control of the EPSG code |
| R5 | Raster resolution | gdalinfo | Pixel size |
| R6 | Raster origin | gdalinfo | It must be consistent with the European standard grids, divided or multiplied based on the resolution. |
| R7 | Bit depth | gdalinfo | Accepted bit depth > currently 8 bit |
| R8 | Compression type | gdalinfo | Accepted lossless compression (LZW or PackBits, others) > currently PackBits in the product specifications. |
| R9 | Pixel values (valid codes) | gdal | The file must contain only valid values. |
| R10 | No data pixels | gdal | No data pixels in the mapped area. |
| R11 | MMU | Customized Python code (NumPy) | Minimum mapping size threshold. Exceptions can apply as e.g. boundary polygons, next-to-cloud areas polygons etc. |
| R12 | Metadata | Web service API | INSPIRE Geoportal Metadata Validator Web Service |
| R13 | Completeness | Gdal, Customized Python code | Complete products defined AOI (or AOI with buffer) must be covered by data. It requires AOI definition file availability and accessibility (may be subject of IPR). |
| R14 | Colour table existence | gdalinfo | The file must have associated colour table. |
| R15 | Colours in the colour table | gdalinfo | Only valid colours (defined by RGB values) must be used |

*Table 2 – Generic raster data checks*

Specific product-related parameters and exceptions

This paragraph specifies articular particular product-related parameters and exceptions to be used for generic checks defined above. The setup for each product is based on the product specification sheets as defined for each product. Listings of checks relevant for each product (dataset) together with individual parameters are shown in tables 3 (vector datasets) and 4 (raster dataset).

Vector products

Following vector Copernicus Land service products are considered in this description:

• CLC status / CLC change

• Urban Atlas / Urban Atlas change

• Urban Atlas Street Tree Layers

• Natura 2000 LU/LC datasets

• Riparian zones LC/LU datasets

Specific parameters and exceptions for particular product are in table 3 below.

|  |  |  |
| --- | --- | --- |
| **Dataset** | **Checks relevant for the dataset** | **Individual parameters for the dataset** |
| CLC  CLC change | V1, V2, V3, V4, V5, V6, V8, V10, V11, V12, V13, V14, V15, V16 | V6: also with list of non-probable change codes performed only for the change layer  V6: also with technical change exception performed only for the change layer. The change layer can contain polygons with the same code “from” and “to” but then the attribute CHType must contain value “T”  V11: 25 Ha for the status layer, 5 Ha for the change layer  Exception for border polygons for both status and change layer  Exception for the complex change applied for change layer  V12: 100 m for both status and change layers  V16: AOI includes 1 km width buffer along land/land borders, 25 km width buffer along land/sea borders |
| Urban Atlas | V1, V2, V3, V4, V5, V6, V8, V10, V11, V13, V14, V15 | V6: also with list of non-probable change codes performed only for the change layer  V11: 0.25 Ha for artificial classes (code 1.x)  1 Ha for all non-artificial classes.  Comment 1 for V11 check:  Due to the integration of transport network information these MMU thresholds might be underrun. In this case as well as for features located at the outer boundary of the mapping area exceptional MMU thresholds have been specified : “*Exception of MMU 0.25 / 1 ha: in case of an homogeneous area > MMU, but divided in 2 or more polygons by the road network, each part can be smaller to preserve the land cover information. However, no polygon can be smaller than 500 m² (e.g. a 1 ha forest divided in 4 polygons by the road network has to be mapped) except for polygons at the border of the FUA (>100 m²).” (source: product specification guide)*  For this reason a MMU of 250m2 has been applied for polygons not touching the border and 50m2 for those polygons touching the border of the FUA. These values correspond to a half of the MMU specified in the mapping guide and are based on a compromise which should allow to find all sliver polygons without being too strict to valuable information inside polygons that are below the MMU.  Comment 2 for V11 check:  For better data handling transport network polygons (classes 1.2.2.x) are split during production into 1 km by 1 km tiles. This means that polygons of the transport network can fall far below the MMU, while still remaining valid. For this reason polygons pertaining to any of the 1.2.2.x classes are excluded from the MMU analysis.  V12: 10m. Exception of minimum width 10 m of a mapping unit: to maintain continuity of linear structures, they can be mapped smaller than 10 m over a distance of up to 50m. |
| Urban Atlas Street Tree Layers | V1, V2, V3, V4, V5, V6, V8, V10, V11, V13, V14, V15 | V11: 0.05 Ha |
| Natura 2000 LU/LC datasets | V1, V2, V3, V4, V5, V6, V8, V10, V13, V14, V15 | V11: 0.5 Ha  V12: 10 m |
| Riparian zones LC/LU datasets | V1, V2, V3, V4, V5, V6, V8, V10, V11, V12, V13, V14, V15 | V11: 0.5 Ha  V12: 10 m |

*Table 3 – Vector datasets included in the QC tool proposal and relevant checks and specs overview for these datasets*

Raster products

Following vector Copernicus Land service products are considered in this description:

* HRL Degree of Imperviousness
* HRL Impervious density change
* HRL Imperviousness classified change
* HRL Tree cover density
* HRL Tree cover density change
* HRL Dominant leaf type
* HRL Dominant leaf type change
* HRL Forest type
* HRL Forest type change
* HRL Grassland
* HRL Water Wetness
* HRL Small Woody Featureswith vector, Raster 5m and Raster 100m

The products described above consist of: Vector, Raster 5m, Raster 100m.

Specific parameters and exceptions for particular product are in table 4 below.

|  |  |  |
| --- | --- | --- |
| **Dataset** | **Checks relevant for the dataset** | **Individual parameters for the dataset** |
| HRL Degree of Imperviousness | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R12, R13, R14, R15 | R5:20m and 100m |
| HRL Impervious density change | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R12, R13, R14, R15 | R5:20m and 100m |
| HRL Tree cover density | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R12, R13, R14, R15 | R5:20m and 100m |
| HRL Forest type | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15 | R5:20m and 100m  R11:0.5 Ha (only for 20m resolution product) |
| HRL Natural and semi-natural grassland | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15 | R5:20m and 100m  R11: 1 Ha (only for 20m resolution product) |
| HRL Wetland | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R12, R13, R14, R15 | R5:20m and 100m |
| HRL Water bodies | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R12, R13, R14, R15 | R5:20m and 100m |

*Table 4 – Raster datasets included in the QC tool proposal and relevant checks and specs overview for these datasets*

Note: Overall approach for checks V16/R13 – ‘Completeness’ has to be discussed separately as final solution for implementation depends on agreed strategy for standard AOI data provision, which may be subject of IPR.