OGC® DOCUMENT: 22-054

External identifier of this OGC® document: http://www.opengis.net/doc/PER/eoep-Hack2018

Open Geospatial Consortium

2022 WEB MAPPING CODE SPRINT SUMMARY ENGINEERING REPORT

ENGINEERING REPORT

DRAFT

Submission Date: 2022-12-20
Approval Date: 2028-07-10
Publication Date: 2028-06-12
Editor: Gobe Hobona, Joana Simoes

Notice: This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is *not an official position* of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.
Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

License Agreement

Use of this document is subject to the license agreement at https://www.ogc.org/license

Copyright notice

Copyright © 2022 Open Geospatial Consortium
To obtain additional rights of use, visit https://www.ogc.org/legal

Note

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

CONTENTS

LIST OF FIGURES

I EXECUTIVE SUMMARY

The mechanisms through which maps are delivered across the Internet have evolved significantly over the past two decades. Advancement of such mechanisms has been driven by a combination of factors, for example, new data formats have emerged, the SWaP-C (size, weight, power, and cost) of the devices has improved, and the capabilities of web browsers have been enhanced by improvements brought by HTML5. This means that some of the functionality that web mapping applications previously could not implement in a standardized way, are now becoming increasingly common.

To support the development of OGC API Standards that standardize many of the new capabilities available to web mapping applications, the Open Geospatial Consortium (OGC) and EuroGeographics hosted the 2022 Web Mapping Code Sprint from November 29th to December 1st, 2022. The event was sponsored by OGC Strategic Member, Ordnance Survey. The code sprint was held as a hybrid event, consisting of a virtual element hosted on the OGC's Discord environment and an in-person element hosted by EuroGeographics in Brussels, Belgium.

Code Sprints experiment with emerging ideas in the context of geospatial Standards, help improve interoperability of existing Standards by experimenting with new extensions or profiles, and are used for building proofs-of-concept to support standards-development activities and the enhancement of software products. Non-coding activities such as testing, working on documentation, or reporting issues are also conducted during a code sprint. In addition, the code sprints' mentor stream provides an excellent opportunity to onboard developers new to the Standards.

The 2022 Web Mapping Code Sprint focused on the following:

- OGC API – Tiles Standard: This Standard describes API building blocks that can enable implementations to serve map tiles, vector tiles (tiled feature data) or tiled coverage data.
- OGC API – Maps candidate standard: This candidate Standard describes API building blocks that can enable implementations to serve spatially referenced and dynamically rendered electronic maps.
- OGC API – Styles candidate Standard: This candidate Standard describes API building blocks that can enable implementations to manage and fetch styles that consist of symbolizing instructions that can be applied by a rendering engine to features and/or coverages.
- and various Styles & Symbology Encodings (e.g. SLD, SymCore, etc)

The mentor stream of the code sprint featured two tutorials, about understanding and using one server side and one client side implementation of OGC API — Tiles. They were both well attended, both in-person and online. The mentor stream also included two onboarding sessions, focused on collaborating in software projects that implement the standards.

The code sprint successfully facilitated the development and testing of prototype implementations of OGC API Standards, including candidate Standards, that relate to web mapping. Further, the code sprint provided a foundation for the development of the next version

of the Symbology Core Standard. Sprint participants were able to provide feedback directly to the editors of the Standards and the editors were able to clarify any issues encountered by the sprint participants. The code sprint also raised awareness about the Standards. The code sprint therefore met all of its objectives.

The OGC is an international consortium of more than 500 businesses, government agencies, research organizations, and universities driven to make geospatial (location) information and services FAIR — Findable, Accessible, Interoperable, and Reusable. The consortium consists of Standards Working Groups (SWGs) that have responsibility for designing a candidate Standard prior to approval as an OGC Standard and for making revisions to an existing OGC Standard. The sprint objectives for the SWGs were to:

- Create awareness about OGC Standards;
- Develop prototype implementations of OGC Standards, including implementations of draft OGC Application Programming Interface (API) Standards;
- Test the prototype implementations;
- Provide feedback to the Editor about what worked and what did not; and
- Provide feedback about the Standards and candidate Standards.

EuroGeographics is a not-for-profit organization that represents many of the National Mapping, Cadastral and Land Registration Authorities across Europe. The organization facilitates access to data, services, and expertise, as well as supporting the sharing of knowledge across the continent. The organization also publishes a product called Open Maps for Europe, which provided a useful resource for sprint participants. For example, within the first day of the code sprint, the sprint participants had implemented an OGC API -Maps façade in front of a Web Map Service (WMS) that was serving maps from the Open Maps for Europe product.

Ordnance Survey (OS) is the National Mapping Agency of Great Britain. OS publishes printed and digital maps, as well as offering access to the maps and data through a variety of APIs. In September 2022, OS launched the OS NGD API suite of products that implement a number of OGC API Standards. The Web Mapping Code Sprint therefore provided an opportunity for OS to directly support the advancement and implementation of the OGC API Standards on which the new OS NGD API products are built. The code sprint also provided an opportunity for OS engineers to directly engage with the editors of the Standards. Such access to editors and SWG members greatly accelerates development of applications.

This engineering report makes the following recommendations for future innovation work items:

- Prototyping and demonstration of support for the querying of coverages in HTML Previews of coverage tiles
- A testbed activity to implement the prototype Symbology Core v2.0 would help to evaluate support for the various use cases identified in this sprint
- A testbed activity to evaluate options for access control across OGC API Standards (covering at least the Features, Records, Tiles, and Styles APIs)

- A testbed activity applying the ISO 19135 standard to the registration and management of styles through OGC API — Styles
- A testbed activity analysing the potential for consistent modification and harmonization of different stylesheets (e.g. SLD, Mapbox styles, ArcGIS, MapInfo)

The sprint participants also made the following recommendations for things that the SWGs should consider:

- A code sprint focused on general security/access control issues, across OGC API and broader OGC Standards
- Design an approach for supporting 'Cancel' requests, so that long-running jobs can be terminated as users zoom or pan across a map
- A user guide for OGC API — Tiles would help to accelerate uptake (e.g. describing how to build static tile servers in the Cloud)
- A future version of the OGC API — Tiles Standard could relax or remove the requirement that the tileset list URL ends with `/tiles`

II KEYWORDS

The following are keywords to be used by search engines and document catalogues.

hackathon, application-to-the-cloud, testbed, docker, web service

III SECURITY CONSIDERATIONS

No security considerations have been made for this document.

IV SUBMITTERS

All questions regarding this document should be directed to the editor or the contributors:

NAME	ORGANIZATION	ROLE
Gobe Hobona	Open Geospatial Consortium	Editor
Joana Simoes	Open Geospatial Consortium	Editor
Andreas Matheus	Secure Dimensions	Contributor
Antonio Cerciello	OSGeo	Contributor
Clemens Portele	interactive instruments GmbH	Contributor
Erwan Bocher	CNRS (National Center for Scientific Research)	Contributor
Francesco Bartoli	OSGeo	Contributor
Iván Sánchez Ortega	OSGeo	Contributor
Jérôme Jacovella-St-Louis	Ecere Corporation	Contributor
Joan Maso	UAB-CREAF	Contributor
Keith Pomakis	MariaDB	Contributor
Michael Gordon	Ordnance Survey	Contributor
Núria Julià Selvas	UAB-CREAF	Contributor
Olivier Ertz	HEIG-VD (School of Management and Engineering Vaud)	Contributor

NAME	ORGANIZATION	ROLE
Oscar Andrés Díaz	GeoSolutions	Contributor
Prajwalita Chavan	IIT Bombay	Contributor
Robin Houtmeyers	Hexagon	Contributor
Tim Schaub	Planet Labs PBC	Contributor
Tom Crauwels	Hexagon	Contributor
Tom Kralidis	Meteorological Service of Canada	Contributor

V ABSTRACT

The subject of this Engineering Report (ER) is a code sprint that was held from November 29th to December 1st, 2022 to advance OGC API Standards that relate to web mapping, and others that relate to styling and symbology encoding standards. The code sprint was hosted by the Open Geospatial Consortium (OGC) and EuroGeographics. The code sprint was sponsored by Ordnance Survey (OS), and was held as a hybrid event with the face-to-face element hosted at the Mundo Madou centre in Brussels, Belgium.

1

SCOPE

1 SCOPE

A Code Sprint is a collaborative and inclusive event driven by innovative and rapid programming with minimal process and organization constraints to support the development of new applications and open standards. Code Sprints experiment with emerging ideas in the context of geospatial standards, help improve interoperability of existing standards by experimenting with new extensions or profiles, and are used for building proofs of concept to support standards development activities and enhancement of software products.

The code sprint described in this engineering report focused on the following approved and candidate Standards:

- OGC API — Tiles Standard
- OGC API — Maps candidate Standard
- OGC API — Styles candidate Standard
- various Styles & Symbology Encodings (e.g. SLD, SE, SymCore, tc)

The code sprint was organized to provide a collaborative environment that enables software developers, users, and architects to work together on open standards that relate to web mapping, styles, and symbology. This engineering report presents the sprint architecture, the results of the prototyping, and a discussion resulting from the prototyping.

2

NORMATIVE REFERENCES

2 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Joan Masó, Jérôme Jacovella-St-Louis: OGC 17-083r4, *OGC Two Dimensional Tile Matrix Set and Tile Set Metadata*. Open Geospatial Consortium (2022). https://docs.ogc.org/is/17-083r4/17-083r4.html.

Joan Masó, Jérôme Jacovella-St-Louis: OGC 20-057, *OGC API — Tiles — Part 1: Core*. Open Geospatial Consortium (2022). https://docs.ogc.org/is/20-057/20-057.html.

Charles Heazel: OGC API — Common — Part 1: Core (Draft). OGC 19-072, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/19-072.html

Charles Heazel: OGC API — Common — Part 2: Geospatial Data (Draft). OGC 20-024, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/20-024.html

Leonard Daly, Rollin Phillips: OGC 20-058, *Interoperable Simulation and Gaming Sprint Year 2 Engineering Report*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-058.html.

Clemens Portele: OGC API — Styles (Draft). OGC 20-009, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/20-009.html

Markus Lupp: OGC 05-078r4, *OpenGIS Styled Layer Descriptor Profile of the Web Map Service Implementation Specification*. Open Geospatial Consortium (2007). https://portal.ogc.org/files/?artifact id=22364.

WHATWG. HTML, Living Standard [online, viewed 2022-12-08]. Available at https://html.spec.whatwg.org/

3

TERMS, DEFINITIONS AND ABBREVIATED TERMS

3 TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word "shall" (not "must") is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the 'ModSpec'. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

3.1. API

a standard set of documented and supported functions and procedures that expose the capabilities or data of an operating system, application, or service to other applications [adapted from ISO/IEC TR 13066-2:2016]

3.2. coordinate reference system

a coordinate system that is related to the real world by a datum term name [source: ISO 19111]

3.3. dataset

a set of data, published or curated by a single agent, and available for access or download in one or more representations (modified from DCAT: https://www.w3.org/TR/vocab-dcat-2/#dcat-scope).

Note 1 to entry: A Web API implementing OGC API — Common often gives access to a single dataset which may be comprised of one or more *geospatial data resources*.

3.4. geospatial data resource

a web accessible resource that consists of a set of geospatial data

Note 1 to entry: In Web APIs implementing *OGC API — Common — Part 2: Geospatial Data*, geospatial data resources are referred to as collections and are defined in the *collections* conformance class.

Note 2 to entry: *geodata* is sometimes used in this document as an abbreviation of *geospatial data*

3.5. geospatial resource aspect

a web accessible resource that represents a component of geospatial information (metadata, schemas…) or geospatial data accessed using a particular mechanism and data model (e.g., feature items, tiles, maps, coverages,…) of a more generic geospatial data resource (e.g., a collection)

Note 1 to entry: Not to be confused with a web accessible resource representation. While resource representations share the same path and are selected by format negotiation, geospatial aspects use different paths. Commonly a geospatial aspect is a subpath of a geospatial data resource.

3.6. map tile

a *tile* that contains information in a raster form where the values of cells are colors which can be readily displayed on rendering devices

Note 1 to entry: Map tiles are generated in combination with *OGC API — Maps*.

3.7. OpenAPI Document

a document (or set of documents) that defines or describes an API. An OpenAPI definition uses and conforms to the OpenAPI Specification (https://www.openapis.org)

3.8. **geographic information**

information concerning phenomena implicitly or explicitly associated with a location relative to the Earth (source: ISO 19101)

3.9. **map**

portrayal of geographic information as a digital image file suitable for display on a computer screen (source: OGC 06-042)

3.10. **portrayal**

presentation of information to humans (source: ISO 19117)

3.11. **map tile**

tile that contains information in a raster form where the values of cells are colors which can be readily displayed on rendering devices

3.12. **tile**

geometric shape with known properties that may or may not be the result of a tiling (tessellation) process. A tile consists of a single connected “piece” without “holes” or “lines” (topological disc).

In the context of a 2D *tile matrix*, a *tile* is one of the rectangular regions of space, which can be uniquely identified by row and column integer indices, making up the tile matrix.

In the context of a geospatial data *tile set*, a *tile* contains data for such a partition of space as part of an overall set of tiles for that tiled geospatial data.

Note 1 to entry: From OGC 19-014r1: Core Tiling Conceptual and Logical Models for 2D Euclidean Space

Note 2 to entry: From OGC 17-083r4: OGC Two Dimensional Tile Matrix Set and Tile Set Metadata Standard

Note 3 to entry: Tiles are useful to efficiently request, transfer, cache, display, store and process geospatial data for a specific resolution and area of interest, providing deterministic performance and scalability for arbitrarily large datasets.

Note 4 to entry: Tiles can contain a variety of data types, such as grid-based pictorial representations (map tiles), coverage subsets (coverage tiles), or feature-based representations (vector tiles).

3.13. **tile matrix**

tiling grid in a given 2D coordinate reference system, associated to a specific scale and partitioning space into regular conterminous *tiles*, each of which being assigned a unique identifier

Note 1 to entry: From OGC 17-083r4: OGC Two Dimensional Tile Matrix Set and Tile Set Metadata Standard

Note 2 to entry: Each tile of a tile matrix is uniquely identifiable by a row and a column integer index. The number of rows is referred to as the *matrix height*, while the maximum number of columns is referred to as the *matrix width* (the number of columns can vary for different rows in *variable width tile matrices*).

3.14. **tile matrix set**

tiling scheme consisting of a set of *tile matrices* defined at different scales covering approximately the same area and having a common coordinate reference system.

Note 1 to entry: From OGC 17-083r4: OGC Two Dimensional Tile Matrix Set and Tile Set Metadata Standard

3.15. **tile set**

a set of *tiles* resulting from tiling data according to a particular *tiling scheme*

Note 1 to entry: From OGC 19-014r1: Core Tiling Conceptual and Logical Models for 2D Euclidean Space, but adapted to clarify that in the context of this document, a tile set refers specifically to a set of tiles containing data and following a common tiling scheme.

3.16. **Web API**

API using an architectural style that is founded on the technologies of the Web [source: OGC API — Features — Part 1: Core]

3.17. Abbreviated terms

API	Application Programming Interface
CRS	Coordinate Reference System
GIS	Geographic Information System
OGC	Open Geospatial Consortium
OWS	OGC Web Services
REST	Representational State Transfer
WMS	Web Map Service
WMTS	Web Map Tile Service

4

HIGH-LEVEL ARCHITECTURE

4 HIGH-LEVEL ARCHITECTURE

As illustrated in Figure 1, the sprint architecture was designed with the view of enabling client applications to connect to different servers that implement open geospatial standards that relate to web mapping, styles and symbology. Implementations of OGC API — Maps, OGC API — Tiles, and OGC API — Styles were deployed in participants' own infrastructure in order to build a solution with the architecture shown below in Figure 1.

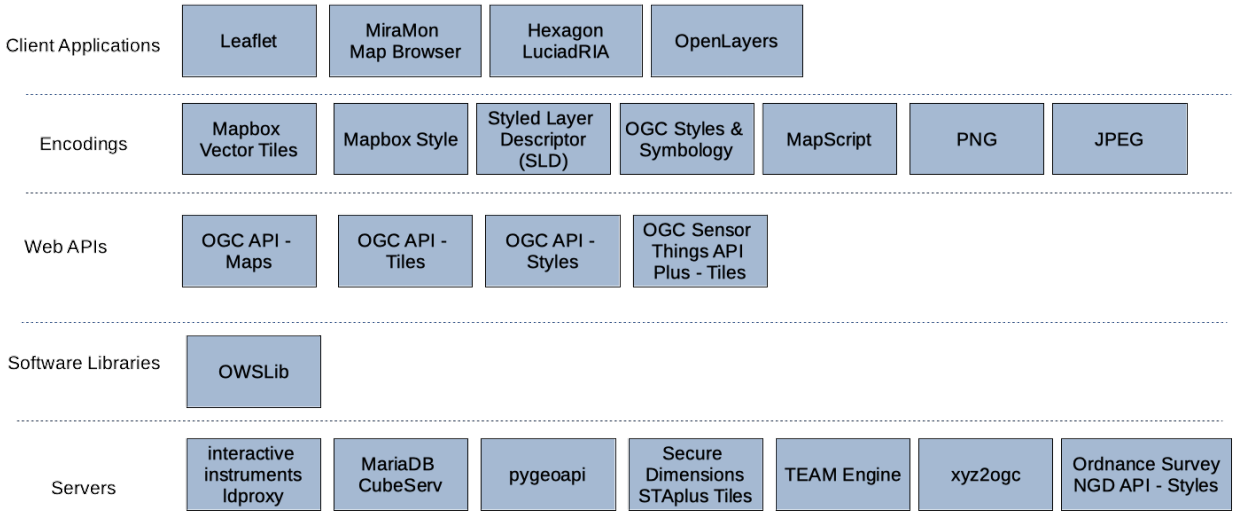


Figure 1 — High Level Overview of the Sprint Architecture

The rest of this section describes the software deployed and standards implemented during the code sprint.

4.1. Approved and Draft Standards

This section describes the approved and draft standards implemented during the code sprint.

4.1.1. OGC API — Maps

The OGC API — Maps candidate Standard describes an API that can serve spatially referenced and dynamically rendered electronic maps. The specification describes the discovery and query operations of an API that provides access to electronic maps in a manner independent of the underlying data store. The query operations allow dynamically rendered maps to be retrieved from the underlying data store based upon simple selection criteria, defined by the client.

4.1.2. OGC API — Styles

OGC API — Styles specifies building blocks for Web APIs that enable map servers and clients as well as visual style editors to manage and fetch styles that consist of symbolizing instructions that can be applied by a rendering engine on features and/or coverages. The API implements the conceptual model for style encodings and style metadata. The model defines three main concepts, namely the style, stylesheet, and style metadata. The concepts are explained below:

- The **style** is the main resource.
- Each style is available in one or more **stylesheets** — the representation of a style in an encoding like OGC SLD 1.0 or Mapbox Style. Clients can use the stylesheet of a style that fits the user's needs best, based on the capabilities of available tools and their preferences.
- For each style there is **style metadata** available, with general descriptive information about the style, structural information (e.g., layers and attributes), and so forth to allow users to discover and select existing styles for their data.

4.1.3. OGC API — Tiles

OGC API — Tiles specifies a Standard for Web APIs that provide tiles of geospatial information. The Standard supports different forms of geospatial information, such as tiles of vector features ("vector tiles"), coverages, maps (or imagery) and potentially eventually additional types of tiles of geospatial information.

Vector data represents geospatial objects such as points, lines, and polygons. Tiles of vector feature data (i.e. Vector Tiles) represent partitions of vector data covering a large area (e.g. lines representing rivers in a country).

In this context, a map is essentially an image representing at least one type of geospatial information. Tiles of maps (i.e. Map Tiles) represent subsets of maps covering a large area (e.g. a satellite image).

4.1.4. OGC STAplus — an extension of the OGC SensorThings API

STAplus is an extension to the OGC SensorThings data model. Inspired by Citizen Science applications, STAplus supports the 'ownership concept' whereby observations collected by sensors are owned by (different) users that may express the license for re-use. In addition to the ownership and license abilities, the extension specified by STAplus supports the expression of explicit relations between observations and the creation of group(s) of observations.

4.1.5. OGC Styled Layer Descriptor

The Styled Layer Descriptor (SLD) encoding standard defines an encoding that supports user-defined symbolization and coloring of geographic feature and coverage data. SLD addresses

the need for users and software to be able to control the visual portrayal of the geospatial data. The ability to define styling rules requires a styling language that the client and server can both understand. SLD is often used in combination with the OGC Symbology Encoding Standard (SE).

4.1.6. OGC Styles and Symbology

The OGC Symbology Conceptual Model: Core Part Standard (OGC 18-067r3), also known as OGC SymCore, specifies the conceptual basis to define symbology rules for the portrayal of geographical data. It is modular and extensible (one core model, many extensions), also encoding agnostic (one symbology model, many encodings). It contains a minimal set of abstract classes representing explicit extension points of the model.

OGC Styles and Symbology is a vision for a standard that implements the OGC Symbology Conceptual Model: Core Part Standard (OGC 18-067r3). The Web Mapping Code Sprint was a key moment to consolidate the roadmap to achieve this vision. The plan towards such a candidate SymCore 2.0 Standard consists of the definition of a conceptual and logical model, Cascading Style Sheets (CSS) and JavaScript Object Notation (JSON) encodings, as well as a mapping to SLD/SE (eventually with existing well known vendor options). The requirements described will be supported by illustrated and encoded cartographic use cases.

4.2. Open Source Software Projects

This section describes open source software products that were deployed during the code sprint.

4.2.1. Leaflet

Leaflet is a popular JavaScript library for displaying maps in web pages, created in 2011 by Volodymir Agafonkin and released under a BSD-2 license. It has a minimalistic architectural design in order to provide a small code footprint. As a result, any advanced or specific functionality is implemented by plugins.

Leaflet.ImageOverlay.OGCAPI is a Leaflet plugin implementing an OGC API — Maps client, and was created during a previous code sprint.

4.2.2. Idproxy

Idproxy is an implementation of the OGC API suite of Standards, inspired by the W3C/OGC Spatial Data on the Web Best Practices. Idproxy is developed by interactive instruments GmbH, written in Java (Source Code), and is typically deployed using docker (DockerHub). In addition to supporting commonly used data formats for geospatial data, an emphasis is placed on an intuitive HTML representation.

The current version supports PostgreSQL/PostGIS databases, GeoPackages and WFS 2.0 instances as backends for feature data. MBTiles is supported for tilesets.

ldproxy implements all conformance classes and recommendations of "OGC API — Features — Part 1: Core" and "OGC API — Features — Part 2: Coordinate Reference Systems By Reference" and is an OGC Reference Implementation for the two standards. ldproxy also supports the OGC API — Records candidate Standard as well as the draft extensions of OGC API — Features (that is Part 3, CQL2, Part 4 and most of the current proposals discussed by the Features API Standards Working Group). ldproxy supports GeoJSON, JSON-FG, HTML, FlatGeoBuf, CityJSON, glTF 2.0 and the Geography Markup Language (GML) as feature encodings.

ldproxy also has implementations for additional resource types: Vector and Map Tiles, Styles, Routes, and 3D Tilesets.

ldproxy is distributed under the Mozilla Public License 2.0.

4.2.3. Miramon Map Browser

The Centre for Ecological Research and Forestry Applications (CREAF) at the Autonomous University of Barcelona (UAB) deployed an instance of the MiraMon Map Browser which is available on the CREAF website and on GitHub. The MiraMon Map Browser is a web application for creating visualization and analysis solutions that run in modern web browsers. The browser uses HTML5, JavaScript, and supports OGC Standards such as OGC API — Tiles, Web Map Service (WMS), and Web Map Tile Service (WMTS). The JavaScript client is able to combine AJAX, binary arrays, the WMS/WMTS protocols, the OGC API — Maps candidate Standard and the OGC API — Tiles Standard. The MiraMon Map Browser has also been extended to support STAplus, an extension of the OGC SensorThings API Standard.

4.2.4. OSGeo OWSLib

OWSLib is a Python client for OGC Web Services (OWS) and their related content models. The project is an OSGeo Community project and is released under a BSD 3-Clause License.

OWSLib supports numerous OGC Standards, including increasing support for the OGC API suite of Standards. The official documentation provides an overview of all supported standards.

4.2.5. OSGeo pygeoapi

pygeoapi is a Python server implementation of the OGC API suite of Standards. The project emerged as part of the next generation OGC API efforts in 2018 and provides the capability for organizations to deploy a RESTful OGC API endpoint using OpenAPI, GeoJSON, and HTML. pygeoapi is open source and released under an MIT license. pygeoapi is an official OSGeo Project as well as an OGC Reference Implementation.

pygeoapi supports numerous OGC API Standards. The official documentation provides an overview of all supported standards.

4.2.6. STAplus Viewer App by Secure Dimensions

The STAplus Viewer App is a proof-of-concept implementation of a web-browser application based on JavaScript (JS) and Leaflet. The implementation further leverages JS libraries from STAM (SensorThings API Map) developed by Datacove for INSPIRE.

The STAplus Viewer App is based on the Leaflet JavaScript library. It is however possible to base the application on OpenLayers. This alternative implementation is available from the COS4Cloud project. Even though it looks and feels slightly different, both implementations provide the same functionality.

4.2.7. TEAM Engine

The Test, Evaluation, And Measurement (TEAM) Engine is a testing facility that executes test suites developed using the TestNG framework or the OGC Compliance Test Language (CTL). It is typically used to verify specification compliance and is the official test harness of the OGC Compliance Testing Program (CITE).

4.2.8. OpenLayers

OpenLayers is a library for developing browser-based mapping applications. It works with vector and raster data from a variety of sources and is able to reproject data for rendering. The library has broad support for OGC protocols and formats, including WMS, WMTS, Web Feature Service (WFS), Well Known Text (WKT), Well Known Binary (WKB), Geography Markup Language (GML), GeoTIFF/COG, and KML. In addition, it has support for community and non-OGC standards such as GeoJSON, XYZ tiles, TileJSON, and more. OpenLayers is written in JavaScript and is available under the BSD 2-Clause license.

The `ol/source/OGCMapTile` module provides a source for map tiles from an OGC API – Tiles implementation. The `ol/source/OGCVectorTile` module provides a source for vector tiles from an OGC API — Tiles implementation.

4.2.9. xyz2ogc

The xyz2ogc program supports the generation of OGC API – Tiles metadata from existing XYZ tilesets. The program is implemented in the Go programming language and the published module can be used in Go-based services that implement the OGC API – Tiles Standard.

4.3. Proprietary products

This section describes proprietary software products that were deployed during the code sprint.

4.3.1. MariaDB CubeWerx CubeServ

The CubeWerx server ("cubeserv") is implemented in C and currently implements the following OGC Standards and draft specifications.

- Multiple conformance classes and recommendations of the OGC API — Tiles — Part 1: Core Standard.
- Multiple conformance classes and recommendations of the OGC API — Maps — Part 1: Core candidate Standard.
- All conformance classes and recommendations of the OGC API — Features — Part 1: Core Standard.
- Multiple conformance classes and recommendations of the OGC API — Records — Part 1: Core candidate Standard.
- Multiple conformance classes and recommendations of the OGC API — Coverages — Part 1: Core candidate Standard.
- Multiple conformance classes and recommendations of the OGC API — Processes — Part 1: Core Standard.
- Multiple versions of the Web Map Service (WMS), Web Processing Service (WPS), Web Map Tile Service (WMTS) and Web Feature Service (WFS) Standards.
- A number of other "un-adopted" OGC Web Service draft specifications including the Testbed-12 Web Integration Service, OWS-7 Engineering Report — GeoSynchronization Service, and the Web Object Service prototype.

The cubeserv executable supports a wide variety of back ends including Oracle, MariaDB, SHAPE files, etc. It also supports a wide array of service-dependent output formats (e.g., GML, GeoJSON, Mapbox Vector Tiles, MapMP, etc.) and coordinate reference systems.

4.3.2. Ordnance Survey Implementation of OGC API — Styles

Ordnance Survey (OS) works with a wide array of customers and partners across many industries in the United Kingdom and beyond. A key part of the current OS Roadmap is the suite of products to deliver the OS National Geographic Database (OS NGD). At the end of September 2022, OS launched the OS NGD API – Features product which implements Parts 1, 2, and 3 of OGC API — Features to provide access to data along with filtering capabilities to enable customers to retrieve just the data they need. During the code sprint, developers from OS focuses on implementation of the OGC API — Styles candidate Standard. The work at the code sprint complemented other OS activities that are implementing OGC API — Tiles.

4.3.3. GNOSIS Map Server

The GNOSIS Map Server is written in the eC programming language and supports multiple OGC API Standards. GNOSIS Map Server supports multiple encodings including GNOSIS Map Tiles (which can contain either vector data, gridded coverages, imagery, point clouds or 3D meshes), Mapbox Vector Tiles, GeoJSON, GeoECON, GML and MapML. An experimental server is available online at https://maps.ecere.com/ogcapi and has been used in multiple OGC Innovation Program initiatives.

4.3.4. Hexagon Luciad RIA

LuciadRIA is a product that enables applications running in a web browser to offer 2D, 3D, or 4D visualization of satellite and other imagery, vector-based data and dynamic content, such as tracks. LuciadRIA can connect to implementations of OGC API Standards, as well as implementations of OGC Web Service standards.

5

RESULTS

5 RESULTS

The code sprint included multiple software products and implementations of OGC and ISO Standards. This section presents some of the results from the code sprint.

5.1. MariaDB CubeWerx CubeSERV

Developers from MariaDB configured an instance of their CubeSERV product to offer an OGC API — Maps interface in front of a WMS that was serving EuroGeographics's EuroRegionalMap. A screenshot of the CubeSERV user interface is shown in Figure 2. The user interface lists the coordinate reference systems that are supported by the collection that is being retrieved.

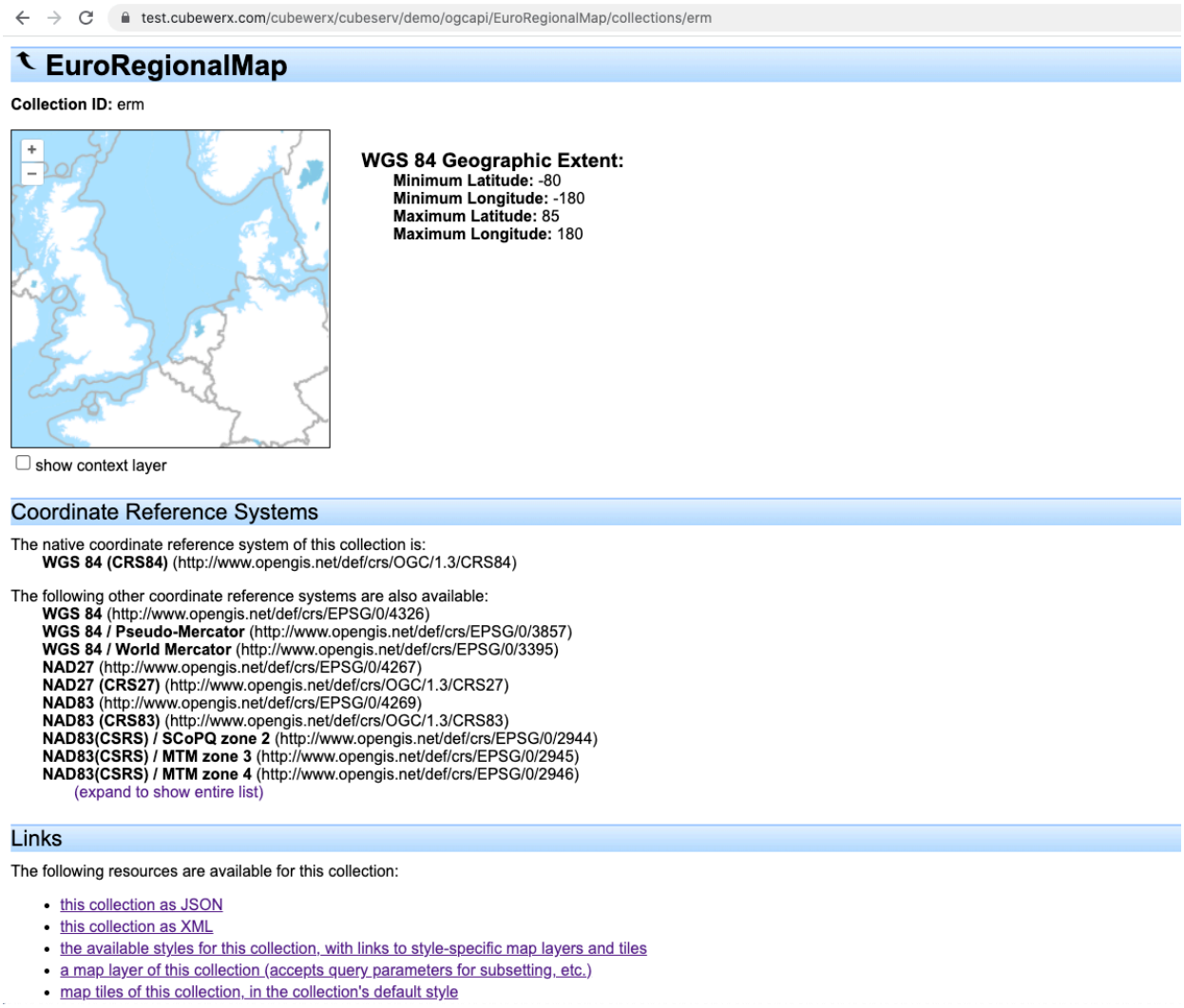


Figure 2 — Screenshot of CubeSERV describing a collection

The styles that are supported by the collection can also be accessed, as demonstrated in Figure 3.

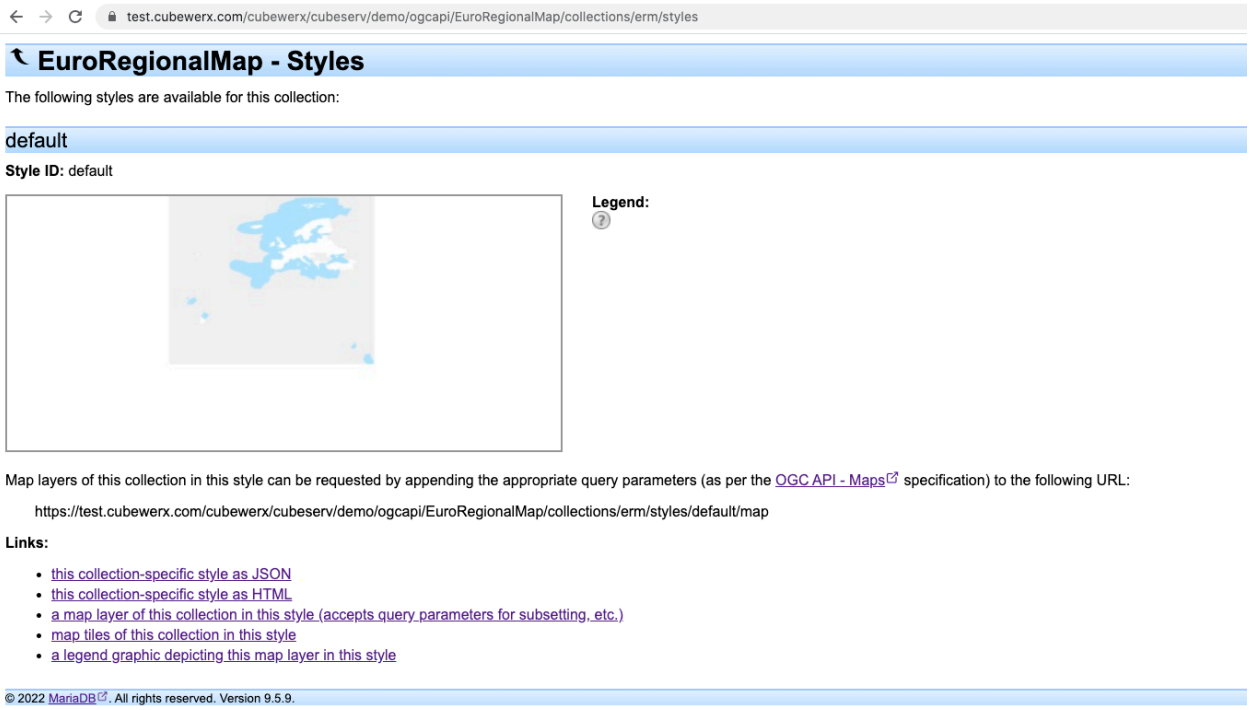


Figure 3 — Screenshot of CubeSERV describing styles supported by the collection

The supported map tiles can also be retrieved. A screenshot describing part of a supported tile matrix set is shown in Figure 4.

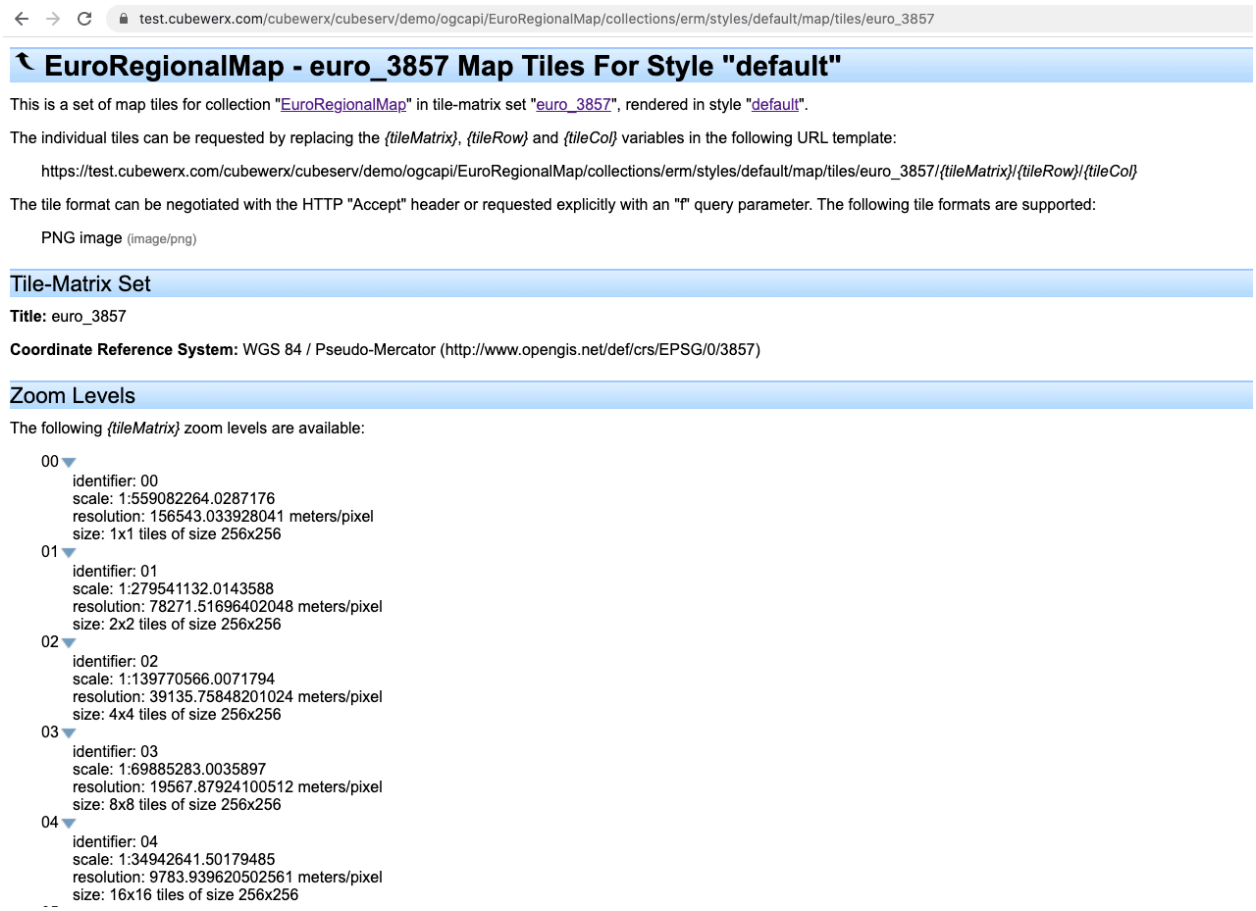


Figure 4 — Screenshot of CubeSERV describing a supported tile matrix set

5.2. GNOSIS Map Server

A demonstration instance of GNOSIS Map Server was used in the code sprint to support implementation of various client applications, including for example Leaflet which is described in Clause 5.3. GNOSIS Map Server was also used to support the implementation of an Executable Test Suite for OGC API — Tiles, which is described in Clause 5.12.

Figure 5 shows a screenshot of the GNOSIS web client (left hand-side) and the responses it is receiving from an instance of GNOSIS Map Server (right hand-side). The interface between the client and server conforms to the OGC API - Tiles Standard. Notice that the responses are individual tiles encoded in the Mapbox Vector Tiles (MVT) format. MVT is an optional encoding supported by OGC API — Tiles.

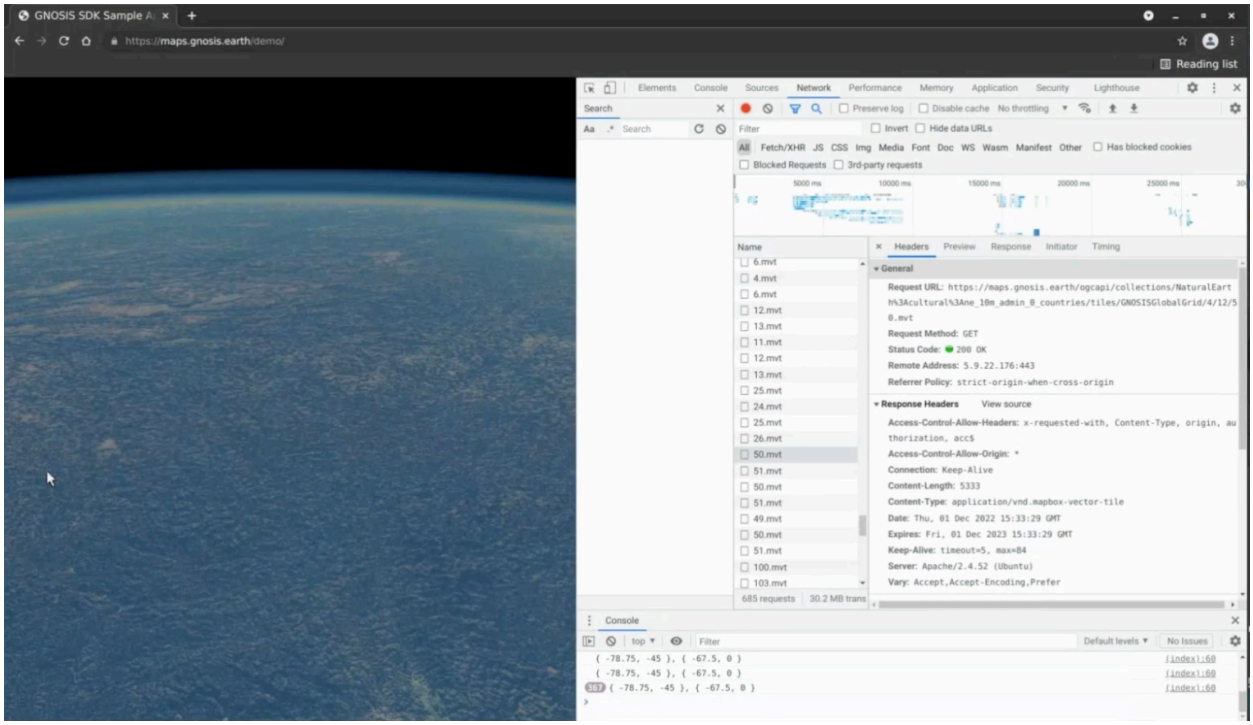


Figure 5 — Screenshot of the GNOSIS web client (left) and responses from an instance of GNOSIS Map Server (right)

5.3. Leaflet

5.3.1. OGC API — Tiles implementation

Leaflet has built-in support for map tiles (or "TileLayer"s), but its internal implementation has significant constrains: All TileLayers must share the same coordinate system, must be aligned to the same origin of coordinates, and all tiles in any given TileLayer must have the same pixel size.

The flexibility of OGC API - Tiles in regard to tile size and origin of coordinates presented a challenge for the Leaflet client implementation. The sprint participants discussed this challenge and reached consensus to ignore TileMatrixSets which do not have a constant tile size, or that do not have a common origin of coordinates for the (0,0) tiles.

5.3.2. OGC API — Maps implementation

An instance of leaflet was configured to access an OGC API — Maps implementation from a MariaDB CubeSERV instance. The CubeSERV instance had been configured to offer an OGC API — Maps façade in front of a WMS that offered layers of the EuroRegionalMap of EuroGeographics. A screenshot of the leaflet instance is shown in Figure 6.

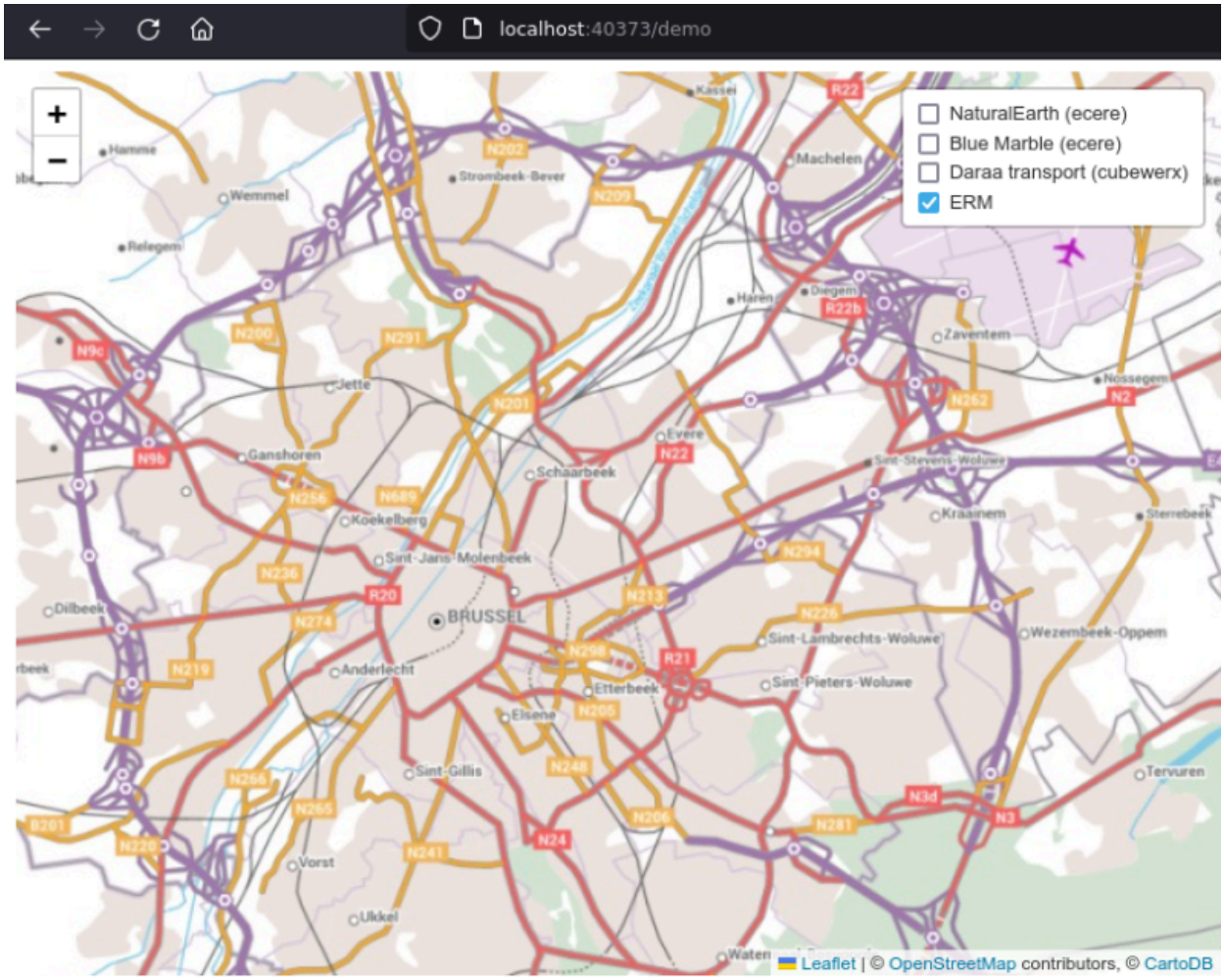


Figure 6 — Screenshot of leaflet accessing implementations of OGC API - Maps and OGC API - Tiles.

5.4. ldproxy

Sprint participants from interactive instruments GmbH updated ldproxy's implementation of *OGC API — Tiles* and *OGC API — Styles*.

5.4.1. OGC API — Tiles

The participants from interactive instruments GmbH updated the ldproxy implementation of *OGC API — Tiles — Part 1: Core* to support the released version 1.0.0 of the Standard. Specifically, the main updates are:

- updated conformance class URIs;

- removed `tileMatrixSetDefinition` member from the tile set metadata;
- added `tiling-scheme` links to the tile set metadata;
- added the `Link-Template` header for the tile URI template;
- added `operationId` to all OpenAPI operations (in addition to setting the `operationId` for all Tiles operations, `operationIds` have been added to all other API operations, too);
- the member `tileMatrixSetURI` in a tile matrix set was changed to `uri`.

Work was also started during the Code Sprint to support the optional `OATiles-hint` header, but this requires more complex changes in various layers of the software and was not finalized during the Code Sprint.

In addition, several improvements to the Tiles building blocks were implemented and tested before and during the Code Sprint:

- For tiles from an MBTiles provider, additional information from the MBTiles container is used for the tile set metadata, including the vector layer and bounding box information;
- harmonize the behavior of the map with tiles in the HTML representation of a tile set, independent of how the representation is configured (using a Mapbox Style, using feature data with a wireframe style, or using tiles from an MBTiles container).

All changes are included in ldproxy v3.3.0, which was released on December 16, 2022.

The ldproxy demonstration deployment at https://demo.ldproxy.net/ was updated with the changes. The deployment includes an API that serves an Ordnance Survey Zoomstack dataset through features, tiles and styles resources.

For example, below is the (shortened) tile set metadata for the Zoomstack vector tiles, including the HTTP headers of the response:

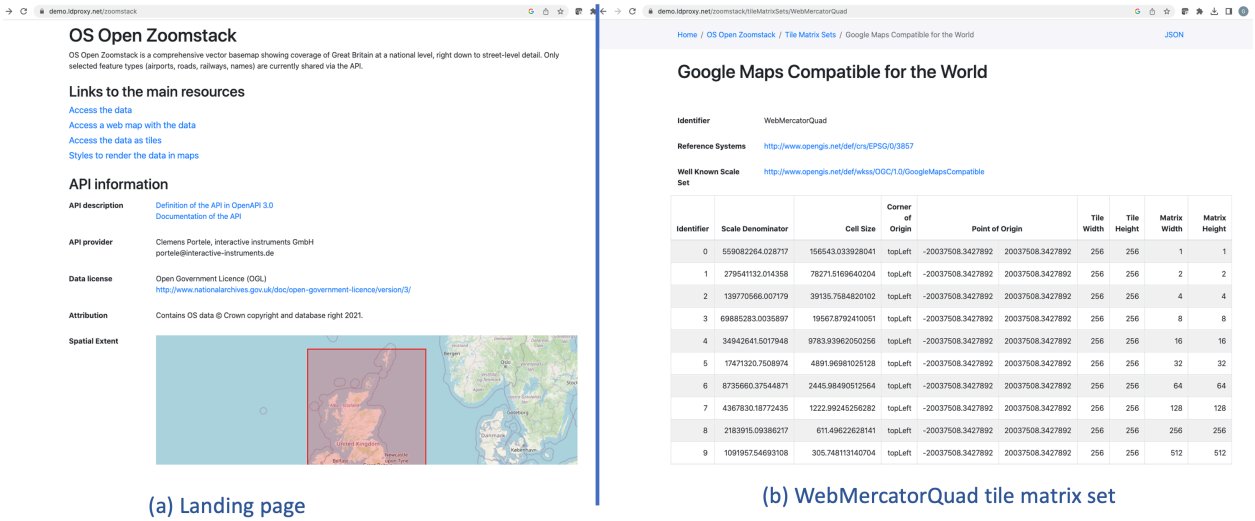
```
HTTP/2 200
content-disposition: inline; filename="WebMercatorQuad.json"
content-language: en
content-type: application/json
date: Fri, 16 Dec 2022 16:26:03 GMT
etag: W/"ed1c7a5750df7c526823056406274c6e--gzip"
link: <https://demo.ldproxy.net/zoomstack/tiles/WebMercatorQuad?f=tilejson>
; rel="alternate"; title="This document as TileJSON"; type="application/vnd.
mapbox.tile+json"
link: <https://demo.ldproxy.net/zoomstack/tileMatrixSets/WebMercatorQuad>; rel=
"http://www.opengis.net/def/rel/ogc/1.0/tiling-scheme"; title="Definition of
the tiling scheme"
link: <https://demo.ldproxy.net/zoomstack/tiles/WebMercatorQuad?f=json>; rel=
"self"; title="This document"; type="application/json"
link-template: <https://demo.ldproxy.net/zoomstack/tiles/WebMercatorQuad/{tileM
atrix}/{tileRow}/{tileCol}?f=mvt>; rel="item"; title="Mapbox vector tiles; the
link is a URI template where {tileMatrix}/{tileRow}/{tileCol} is the tile in
the tiling scheme '{{tileMatrixSetId}}'"; type="application/vnd.mapbox-vector-
tile"
vary: Accept,Accept-Language,Accept-Encoding
x-request-id: 72c93630-2a72-4ec0-a58d-e883e5a93bf5
```

```
{
  "links" : [ {
    "rel" : "self",
    "type" : "application/json",
    "title" : "Dieses Dokument",
    "href" : "https://demo.ldproxy.net/zoomstack/tiles/WebMercatorQuad?f=json"
  }, {
    "rel" : "alternate",
    "type" : "application/vnd.mapbox.tile+json",
    "title" : "Dieses Dokument als TileJSON",
    "href" : "https://demo.ldproxy.net/zoomstack/tiles/WebMercatorQuad?f=
tilejson"
  }, {
    "rel" : "http://www.opengis.net/def/rel/ogc/1.0/tiling-scheme",
    "title" : "Beschreibung des Kachelschemas",
    "href" : "https://demo.ldproxy.net/zoomstack/tileMatrixSets/WebMercatorQuad
"
  }, {
    "rel" : "item",
    "type" : "application/vnd.mapbox-vector-tile",
    "title" : "Mapbox Vector Tiles; der Link ist ein URI-Template wobei {tile
Matrix}/{tileRow}/{tileCol} die Kachel im Kachelschema '{{tileMatrixSetId}}'
identifiziert",
    "href" : "https://demo.ldproxy.net/zoomstack/tiles/WebMercatorQuad/{tileMat
rix}/{tileRow}/{tileCol}?f=mvt",
    "templated" : true
  } ],
  "dataType" : "vector",
  "tileMatrixSetId" : "WebMercatorQuad",
  "tileMatrixSetURI" : "http://www.opengis.net/def/tilematrixset/OGC/1.0/
WebMercatorQuad",
  "tileMatrixSetLimits" : [ {
    "tileMatrix" : "0",
    "minTileRow" : 0,
    "maxTileRow" : 0,
    "minTileCol" : 0,
    "maxTileCol" : 0
  }, ..., {
    "tileMatrix" : "13",
    "minTileRow" : 2343,
    "maxTileRow" : 2784,
    "minTileCol" : 3886,
    "maxTileCol" : 4157
  }, {
    "tileMatrix" : "14",
    "minTileRow" : 4686,
    "maxTileRow" : 5569,
    "minTileCol" : 7772,
    "maxTileCol" : 8314
  } ],
  "boundingBox" : {
    "lowerLeft" : [ -9.2230281, 49.8211737 ],
    "upperRight" : [ 2.6891998, 60.7781954 ],
    "crs" : "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
  },
  "centerPoint" : {
    "coordinates" : [ -3.2835, 58.6359 ],
    "tileMatrix" : "10"
  },
  "layers" : [ {
    "title" : "sea",
    "id" : "sea",
    "dataType" : "vector"
```

```
}, {
  "title" : "names",
  "id" : "names",
  "dataType" : "vector",
  "propertiesSchema" : {
    "properties" : {
      "name2language" : {
        "type" : "string"
      },
      "type" : {
        "type" : "string"
      },
      "name2" : {
        "type" : "string"
      },
      "name1" : {
        "type" : "string"
      },
      "name1language" : {
        "type" : "string"
      }
    },
    "type" : "object"
  }
}, {
  "title" : "rail",
  "id" : "rail",
  "dataType" : "vector",
  "propertiesSchema" : {
    "properties" : {
      "type" : {
        "type" : "string"
      }
    },
    "type" : "object"
  }
}, ..., {
  "title" : "airports",
  "id" : "airports",
  "dataType" : "vector",
  "propertiesSchema" : {
    "properties" : {
      "name" : {
        "type" : "string"
      }
    },
    "type" : "object"
  }
}, {
  "title" : "woodland",
  "id" : "woodland",
  "dataType" : "vector"
}, {
  "title" : "national_parks",
  "id" : "national_parks",
  "dataType" : "vector"
}, {
  "title" : "urban_areas",
  "id" : "urban_areas",
  "dataType" : "vector"
}, {
  "title" : "surfacewater",
  "id" : "surfacewater",
```

```
    "dataType" : "vector"
  } ]
}
```

Screenshots of the ldproxy landing page (a) and its WebMercatorQuad tile matrix set definition (b) are shown in Figure 7.



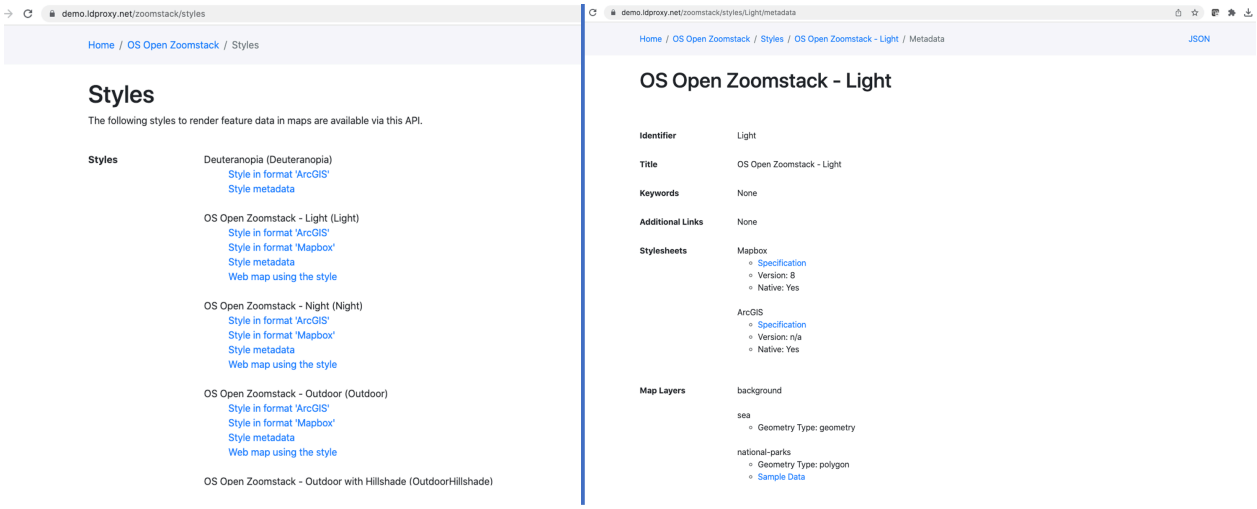
(a) Landing page

(b) WebMercatorQuad tile matrix set

Figure 7 — Screenshot of the ldproxy landing page (a) and its WebMercatorQuad tile matrix set definition (b)

5.4.2. OGC API — Styles

The ldproxy demonstration implementation of OGC API — Styles was updated with additional styles during the code sprint. The demo server provided a reference for other participants to validate their implementations by. A screenshot of a listing of styles offered by the ldproxy demo is shown in Figure 8 (a) and metadata about a single style, derived from the information in the style, is demonstrated in Figure 8 (b).



(a) Listing of styles

(b) Style metadata

Figure 8 — Screenshot of the listing of styles offered by the ldproxy demo (a) and style metadata (b)

Figure 9 shows a map using the MapLibre JS library with a style from the API which accesses the Zoomstack vector tiles from the Tiles resources of the same API and hillshade coverage tiles from an Ordnance Survey server.

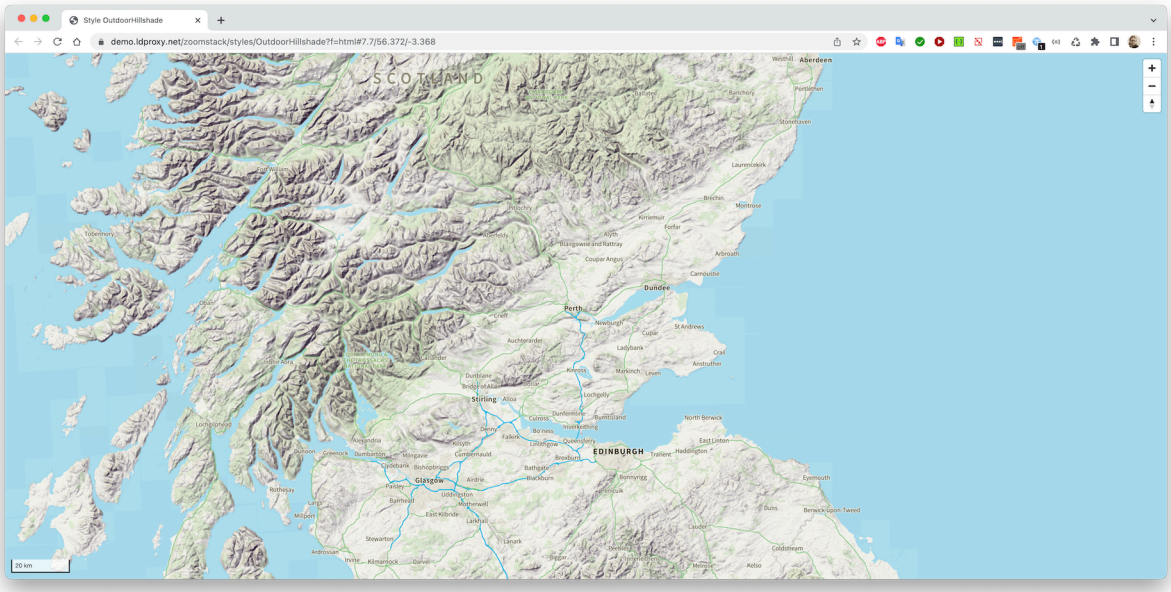


Figure 9 — Screenshot of a map with the Zoomstack style "Outdoor" with additional hillshades

5.5. Hexagon — LuciadFusion/LuciadRIA

5.5.1. Contribution

The sprint participants from Hexagon, Robin Houtmeyers and Tom Crauwels, divided their efforts between the server side and the client side. On the server side they integrated *OGC API — Tiles* as a possible service option in LuciadFusion Studio. On the client side they created a prototype connector for the *OGC API — Maps* candidate Standard and the *OGC API — Tiles* Standard in LuciadRIA and integrated the connector into the LuciadRIA Data Formats sample. More developer-focused information such as documentation, API reference, samples, videos, etc can be found on https://dev.luciad.com/ .

5.5.2. Server side with LuciadFusion

LuciadFusion provides an all-in-one server solution for geospatial data management. At its core is an API that can be used to develop custom server solutions. It ships with LuciadFusion Studio, an out-of-the-box web frontend application developed on top of that API, that lets an end-user manage data intelligently, store and process a multitude of data formats and feed data to multiple applications. A demonstration video can be found here.

The sprint participants from Hexagon started from an earlier prototype that could offer data through an *OGC API — Tiles* interface. The goal of the earlier prototype was a proof of concept, a document that explains how to add support for *OGC API — Tiles* and a rough estimate of the workload. Since *OGC API — Tiles* is defined using the OpenAPI specification, the OpenAPI specification and the tool OpenAPI Generator could be used to generate domain models and REST controllers (for example: Spring controllers).

During this sprint, that earlier prototype was integrated into LuciadFusion Studio as a proof of concept. This means that after adding or crawling data, it could be published as a service that implements OGC API — Tiles, as illustrated in Figure 10, next to existing services such as WMS, WMTS, etc. This functionality was demonstrated in a short video with sample data downloaded from Ordnance Survey's OS Data Hub.

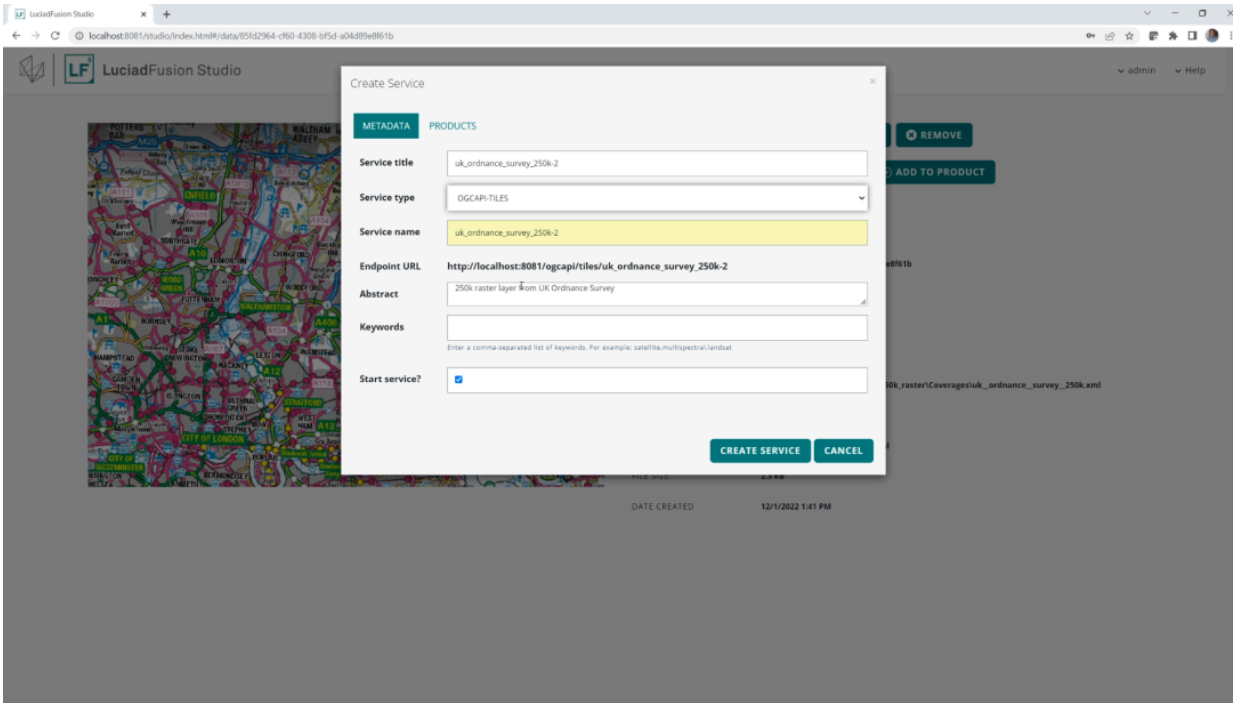


Figure 10 — Dialog for adding a layer from an OGC API - Tiles instance to LuciadFusion

Note: this is still work in progress; there is no out-of-the-box LuciadFusion prototype with OGC API support available yet.

5.5.3. Client side with LuciadRIA

5.5.3.1. Overview

LuciadRIA is a JavaScript API that uses WebGL and HTML5 to deliver desktop-like performance in geospatial applications on the web. A video demonstrating it in action can be found here. More developer oriented information, including live samples, can be found on the Hexagon Luciad RIA Developer Platform.

The sprint participants from Hexagon decided to extend the LuciadRIA Data Formats sample and worked towards integrating the *OGC API — Tiles* & *OGC API — Maps* functionality into its data loading functionality, similar to the existing WMS and WMTS capabilities parsing. Once a connection could be made to one of the servers made available by the other participants, the team from Hexagon implemented a model that could retrieve data and visualize it in a WebGLMap. The team also looked into implementing the vector tiling, but after some trials decided to defer that to future development and to use the remaining time in this sprint to make sure the demo highlights the capability to visualize data on-the-fly in any projection. The demo is available here.

5.5.3.2. Capability parsing

Many of the new OGC API Standards offer a central overview of all data sets available on a server in JSON (or xml, html) at a “collections” URL. This central overview offers links to specific collections (with a unique ID) which in turn can offer links to maps, tiles, styles, etc. An overview from the GNOSIS Map Server:

```
/ogcapi/collections/{collectionID}(collection description)
/ogcapi/collections/{collectionID}/tiles(vector tiles)
/ogcapi/collections/{collectionID}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRo
w}/{tileCol}.{format}(individual vector tile)
/ogcapi/collections/{collectionID}/map/tiles(map tiles)
/ogcapi/collections/{collectionID}/map/tiles/{tileMatrixSetId}/{tileMatrix}/{ti
leRow}/{tileCol}.{format}(individual map tile)
/ogcapi/collections/{collectionID}/coverage/tiles(map tiles)
/ogcapi/collections/{collectionID}/coverage/tiles/{tileMatrixSetId}/{tileMatrix
}/{tileRow}/{tileCol}.{format}(individual coverage tile)
/ogcapi/collections/{collectionID}/items(vector features)
/ogcapi/collections/{collectionID}/coverage(coverage)
/ogcapi/collections/{collectionID}/map(map)
/ogcapi/collections/{collectionID}/styles/{styleId}/map(styled layer map)
/ogcapi/map(dataset map)
/ogcapi/styles/{styleId}/map(styled dataset map)
/ogcapi/tileMatrixSets/{tileMatrixSetId}(individual tiling scheme)
```

In the Data Formats sample, the team created a `ConnectToOGCAPIMapsWizard` and added it to the `ConnectToModal`. This wizard creates an `OGCAPIMapsCapabilities` object after connecting to a valid URL, which contains a collection of `OGCAPIMapsCapabilitiesLayer` objects, one for each collection available. Figure 11 shows a list of collections retrieved by a LuciadRIA web application from an OGC API — Maps instance. Finally, an `OGCAPIMapsLoader` is used to create a model and layer that LuciadRIA can use to load and visualize the data.

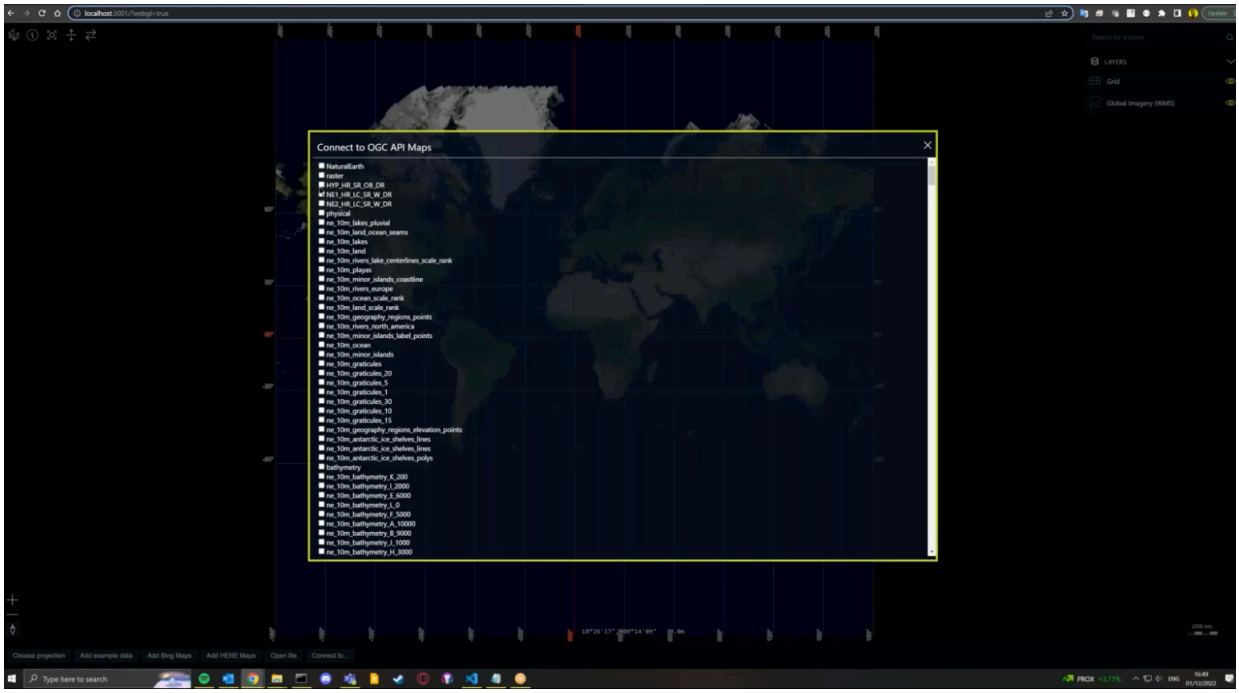


Figure 11 — List of collections retrieved by LuciadRIA from an OGC API - Maps instance

5.5.3.3. Retrieving tiles

Retrieving data can be done in 2 distinct ways: /map or /tiles. The first way, /map, is very similar to how WMS works, a user can request data for a certain bounding box. The newer way, available from /tiles, is similar to how many popular map servers, such as OpenStreetMap (OSM), work. The tiles are cut using a certain tile matrix set (https://www.ogc.org/standards/tms) and can then be requested using a format such as:

collections/{collectionID}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}.{format}

So for example:

collections/bluemarble/tiles/WebMercatorQuad/0/0/0.png

Multiple tile matrix sets can be offered and their specifications can be requested (in JSON or HTML). The specification offers the width and height of tiles, the number of columns and rows, the scale and a way to calculate the bounding box, so everything a client needs to calculate each tile. In the LuciadRIA example implementation, the team focused on the classic quad tree structure such as WebMercatorQuad.

The team extended UrlTileSetModel to implement such a connection. It would map easily on the /tiles offering, since it works exactly like that. To connect to the /maps offering, the team used the function `getTileBounds` to calculate the bounds of a tile and modify the URL so that it matches the template specified in the *OGC API – Maps* candidate Standard, after which a request is sent to the server with a bounding box referenced to a CRS.

5.5.3.4. Vector tiling

OGC API — Tiles supports vector tiles, in addition to map tiles. The tiling structure is similar for imagery data and is equally available at /tiles, but instead of PNG images, you can retrieve GeoJSON. Since LuciadRIA has an out-of-the-box connector for GeoJSON, it would be fairly straight forward to visualize those tiles. Since there is a UrlTileSetModel that can follow the needed tiling pattern, the plan was to use that to request tiles at the correct moment and then use the GeoJSON decoder to load and visualize them. However, after some further research it turned out that while it is easy to get the correct tiles the first time, the functionality to switch back to lower detailed tiles while zooming out was unattainable. After writing some initial code to create a custom tiling mechanism based on the Tile Matrix Set, the team decided that it could not be completed within the sprint's timeframe and consequently changed focus. The team noted, however, that the approach would be implementable if there was more time available.

5.5.3.5. Reprojecting

One of the core strengths of the Luciad portfolio is that it can be used to connect to and visualize data in any format or projection. LuciadRIA can even utilize the power of the GPU through WebGL to project data on-the-fly, such as imagery and vector data, to any known projection and visualize it as such. To demonstrate this, the team added another menu option in the sample to switch to different projections. Figure 12 shows a representation of a map retrieved in the EPSG:4978 coordinate reference system. Figure 13 shows a menu for selecting different coordinate reference systems. Figure 14 shows a representation of a map retrieved in the EPSG:27700 coordinate reference system. In sequence, these three screenshots demonstrate the ease of switching between different coordinate reference systems in LuciadRIA.

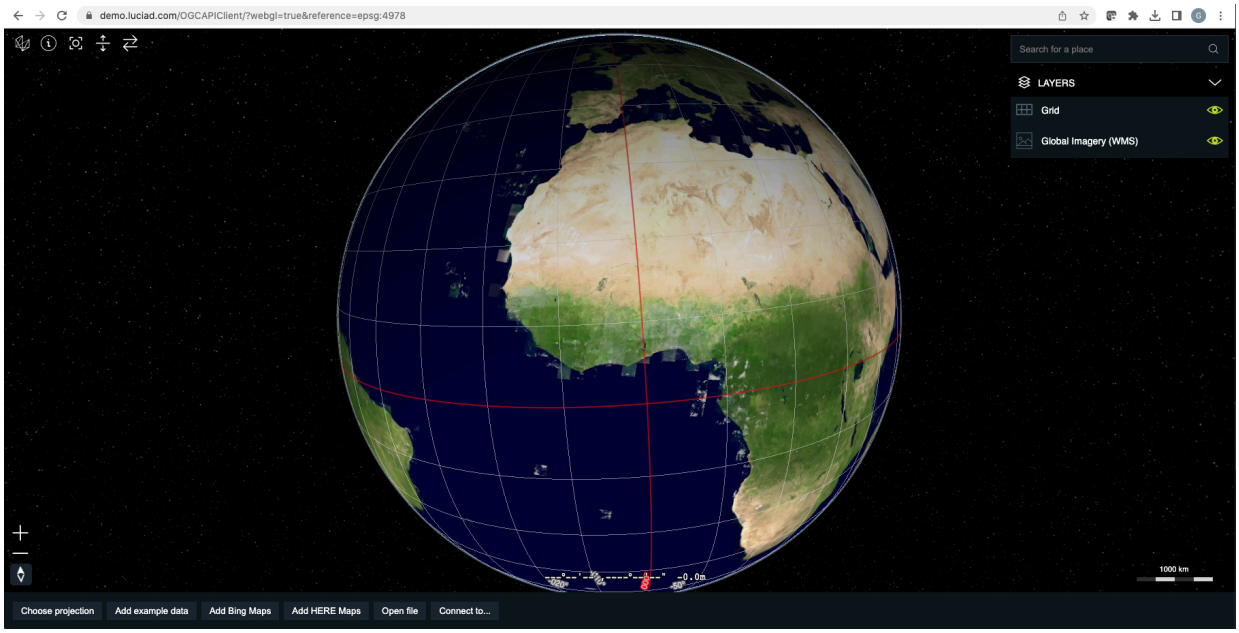


Figure 12 — Display of a map retrieved in EPSG:4978

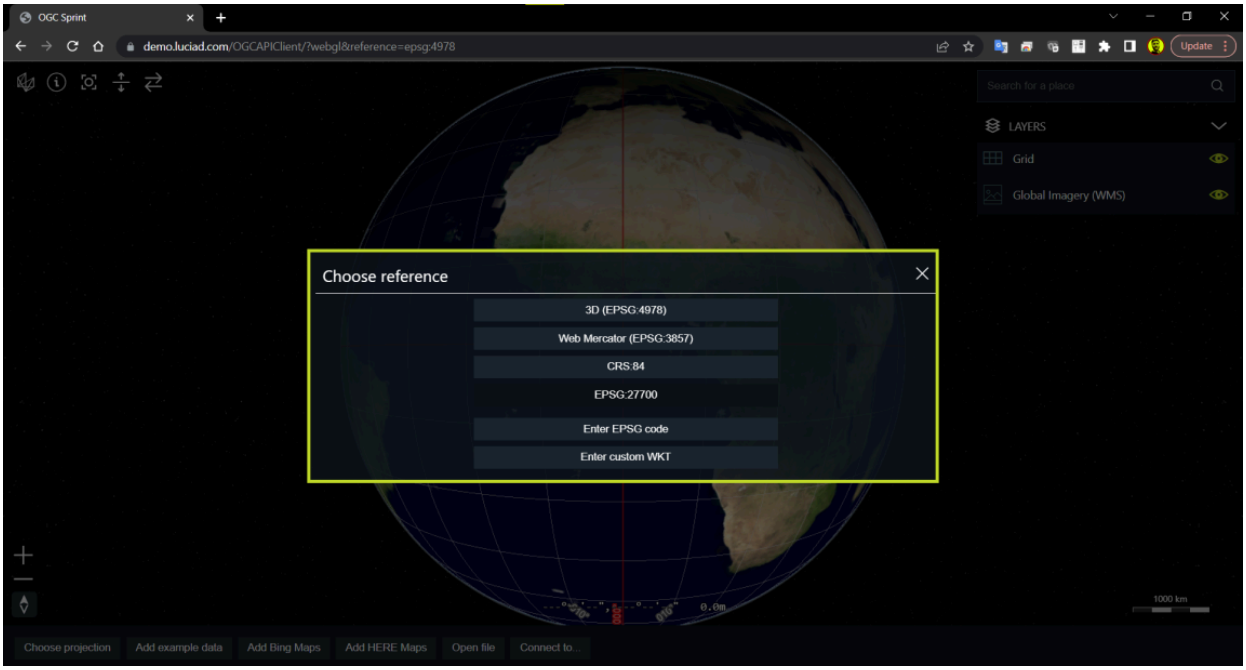


Figure 13 — Dialog to select a coordinate reference system

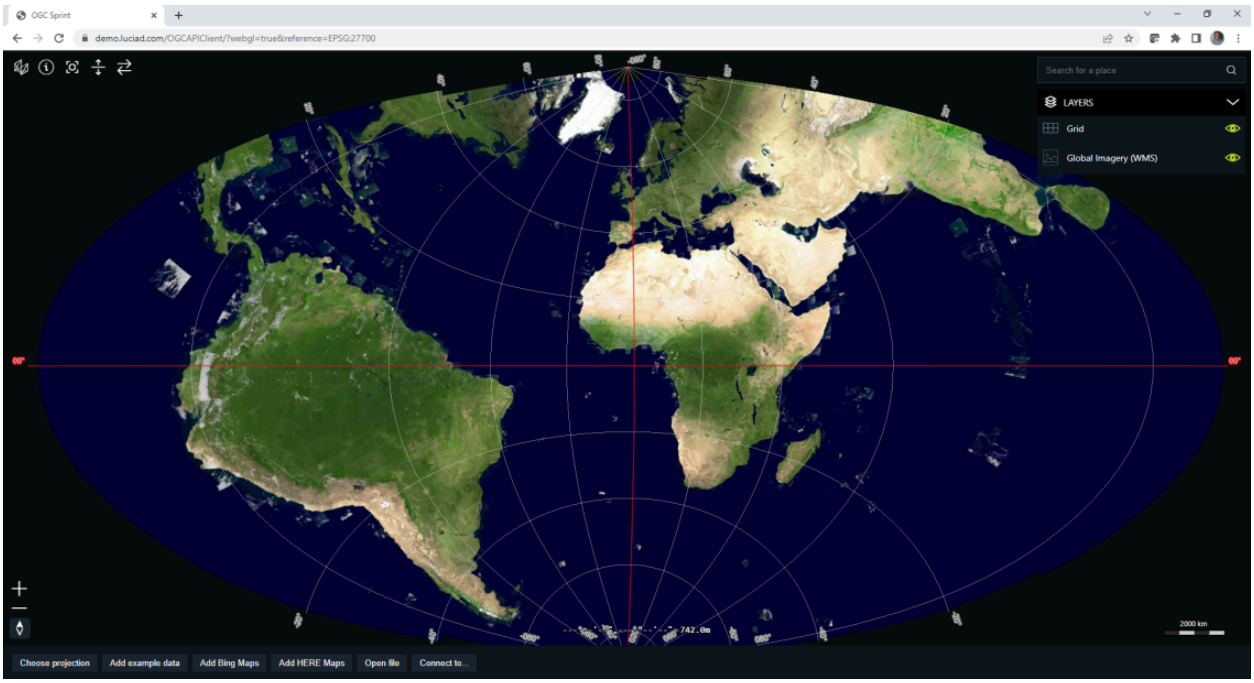


Figure 14 — Display of a map retrieved in EPSG:27700

Next to some predefined options such as 3D, CRS84, WebMercator and a projection requested by another participant (British National Grid projection (EPSG:27700)), the team also added the option to load any projection known to LuciadRIA. To minimize the size of a LuciadRIA application, not all projections are available in it out of the box. However, other projections can be added as WKT (Well Known Text)) by its identifier, which in most cases will be an EPSG code. If LuciadRIA does not know this code, it will automatically try to request it from an external service that provides definitions of projections. After getting the WKT, the application can be

restarted in the new projection and any subsequently loaded data will be reprojected into that projection.

5.5.3.6. Conclusions

There are some obvious quality-of-life benefits that come with this new API:

- The switch from XML to HTML/JSON is a very welcome one as parsing of JSON is much easier in web environments.
- Next to that, the building of tiles could definitely make it easier to load and visualize data.
- The addition of vector tiles also offers some obvious benefits over WFS.

All of the aforementioned benefits also come with some downsides, such as:

- The API offers so many options in terms of tiling, formats, structures, etc with quite some freedom for whoever implements it, that it could cause confusion or incompatibilities later on. For example, a client that chooses to implement only a certain format and tiling structure may not be able to connect to a server that also only supports another format/tiling structure.
- This freedom was also notable when parsing the metadata, as sometimes links were absolute, and other times the links were relative, and at times links were missing altogether.

5.5.4. Resources

The LuciadRIA demonstration sample (= extended Data Formats sample) is available online:

`https://demo.luciad.com/OGCAPIClient/?webgl&reference=epsg:4978`

Using the 'Connect To' button at the bottom, an end-user can add connections to servers that implement OGC API — Maps.

The video created, showcasing the LuciadFusion integration, can be found here.

Servers, from other participants, used by LuciadRIA during the sprint are listed below:

```
GNOSIS Map Server: https://maps.gnosis.earth/ogcapi/
CubeServ: https://test.cubewerx.com/cubewerx/cubeserv/demo/ogcapi/EuroRegionalMap
```

5.6. MiraMon Map Browser

Sprint participants from CREAF worked mainly on improving the implementation of a web map client, called the MiraMon Map Browser, to improve support for OGC API — Tiles, OGC

API -Maps (e.g. extending the MiraMon Map Browser to support the addition of layers from implementations of OGC API — Maps) and also to experiment with how to apply tessellation in an implementation of the OGC SensorThings API (STA). The analysis covered tiles generated on the server-side, as well as client-side tiled requests to implementations of the STA.

5.6.1. OGC API – Maps and support for Map Tiles

To begin with, the source code was overhauled to update the implementation to the approved version of OGC API — Tiles and the current draft version OGC API — Maps. A working instance of the MiraMon Map Browser was prepared showing collections from multiple providers, as demonstrated in Figure 15.

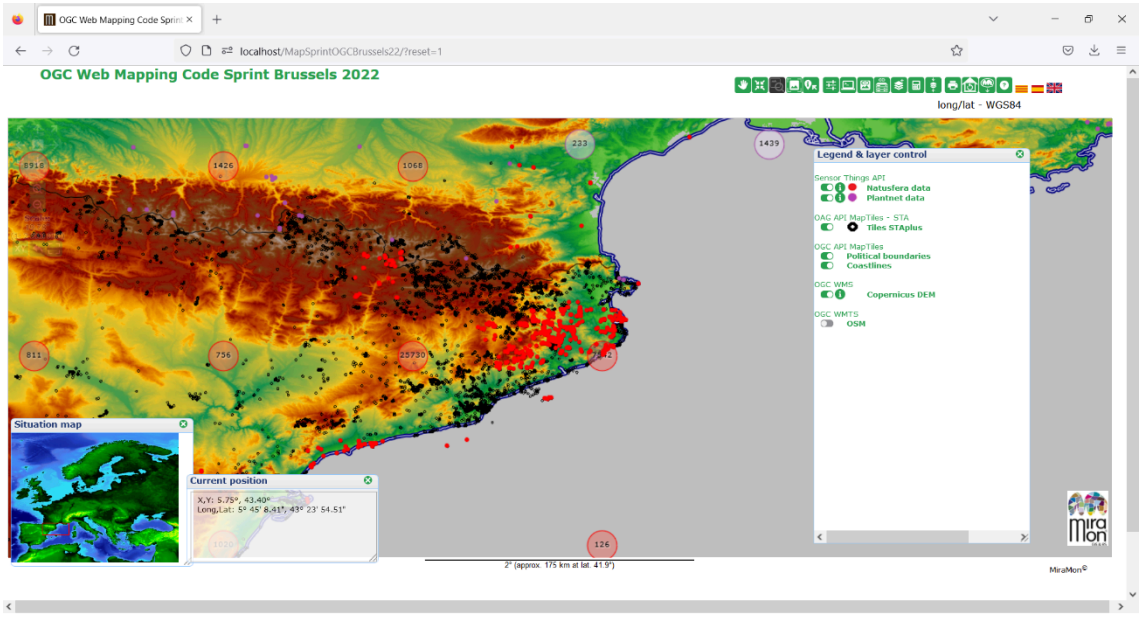


Figure 15 — Screenshot of the MiraMon Map Browser with a combination of several collections (or layers) in the same client using different OGC protocols and APIs, like WMS, WMTS, OGC API - Maps, OGC API - Tiles, and SensorThings API

After that, the work focused on having a full implementation of the OGC API - Maps candidate Standard in the MiraMon Map Browser to enable end-users to add collections from external servers. The map client is able to establish a connection with a server from the information provided by the landing page. The map client inspects the information and then follows the links, making the necessary requests to obtain a list of available map collections and their main characteristics.

As shown in Figure 16, the interface allows users to indicate the landing page of the service that should be explored. Once explored, it displays a list of available map collections and allows users to select the collection or collections to add as layers in the legend and the map display. The selected layers are added to the map display as demonstrated in Figure 17.

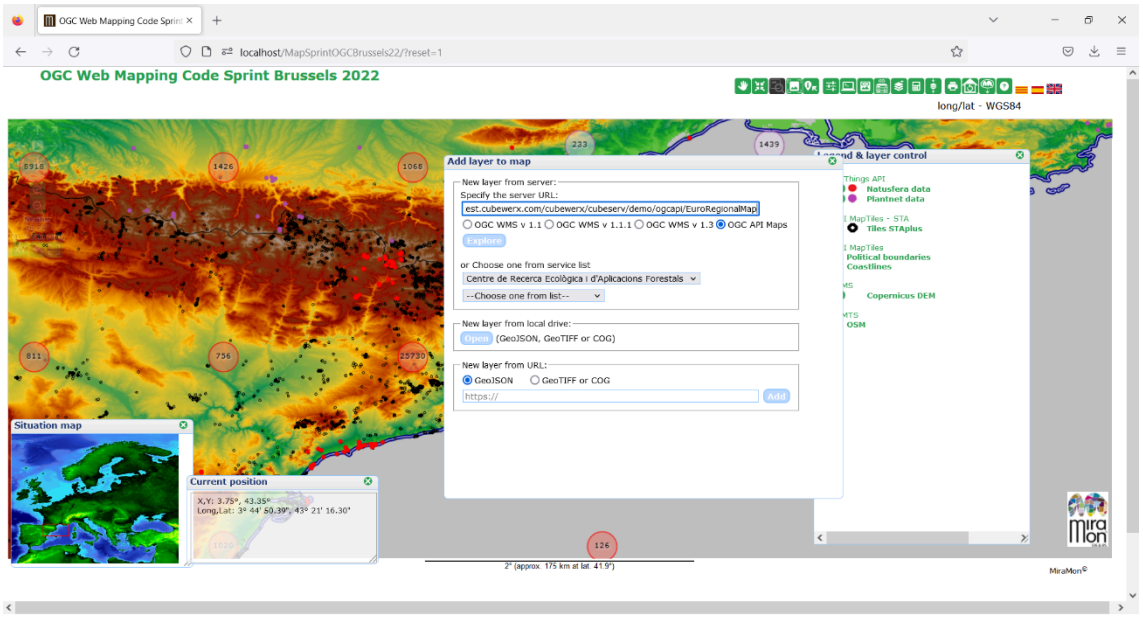


Figure 16 — Screenshot of the first step of the process for adding collections to the client from a OGC API - Maps compatible landing page prepared by Cubewerx

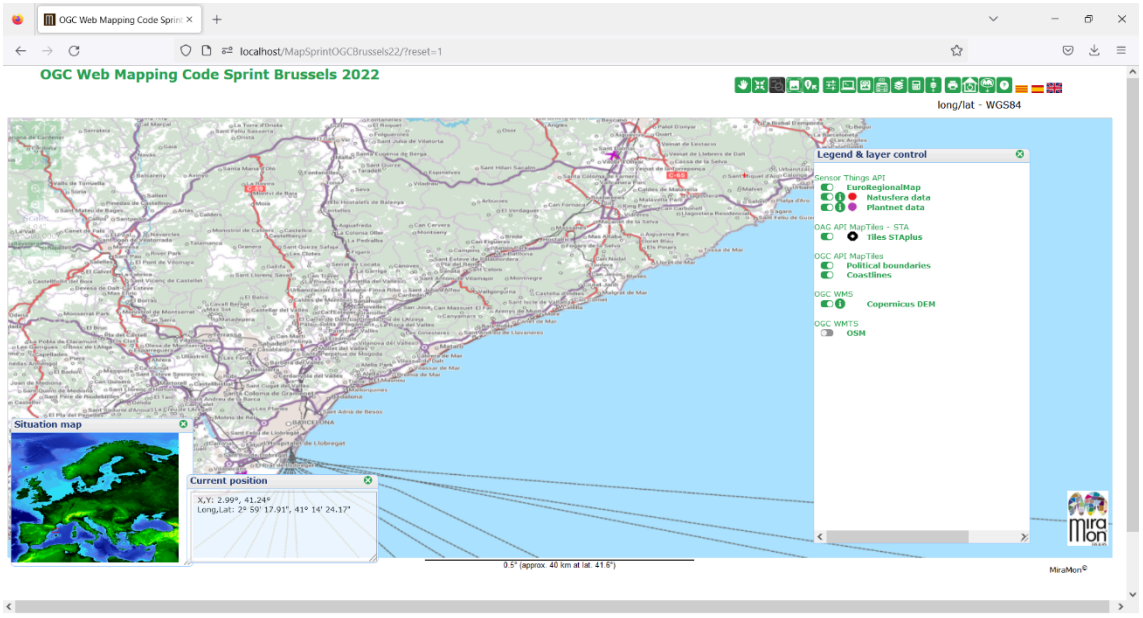


Figure 17 — Screenshot of the EuroGeographics map served by the Cubewerx API.

Finally, the focus of the work shifted to connecting to map collections retrieved from OGC API - Tiles implementations so that an end-user could add tiled maps to the client. The work consisted of obtaining the tiled description and the list of tiled maps available in a service from the landing page. The work was more extensive than the team had anticipated and in the process of an in-depth review of the Standard some difficulties were encountered to properly understand some aspects. A suggestion for incorporating more coding examples in some of the requirements classes was made.

5.6.2. OGC Sensor Things API implementation

When working with a large amount of citizen data accessible through an implementation of the SensorThings API Standard, the performance of a client application can be adversely impacted. In particular, trying to request the necessary data to present a map with a large number of features of interest can be a very slow and inefficient process. To address this issue, two approaches for tessellation of the data were tested during the code sprint and other approaches were proposed.

5.6.2.1. Some discussions on allowing tile requests in SensorThings API Standard

An approach based on the combination of OGC API — Tiles and STA would require modifying the current SensorThings API Standard to support a request such as:

https://cos4cloud.demo.secure-dimensions.de/staplus/v1.1/TileMatrixSetId("WorldMercatorWGS84Quad")/tileMatrix(45)/TileRow(3)/TileCol(2)?$select=feature

Other approaches were discussed based on the extension of the Well Known Text filtering syntax to allow the specification of new conditions, for example:

https://cos4cloud.demo.secure-dimensions.de/staplus/v1.1/FeaturesOfInterest/?$select=feature&$filter=properties/tilematrixset eq 'WorldMercatorWGS84Quad' and properties/tilematrix eq 4 and properties/tilerow eq 6 and properties/tilerow eq 10

https://cos4cloud.demo.secure-dimensions.de/staplus/v1.1/FeaturesOfInterest/?$select=feature&$filter=properties/tilematrixset eq 'WorldMercatorWGS84Quad'($filter=properties/tilematrix eq 4($filter=properties/tilerow eq 6 ($filter=properties/tilerow eq 10)))

https://cos4cloud.demo.secure-dimensions.de/staplus/v1.1/DataStream/?$select=feature&$expand=Locations&$filter=tilematrixset eq 'WorldMercatorWGS84Quad' and tilematrix eq 4 and tilerow eq 6 and tilerow eq 10

5.6.2.2. A client-side implementation

A client-based tessellation solution was implemented that addressed the situation of a server that offers a significant number of vector objects, whether those objects represent features, sensors, or their observations. This approach required implementation of two capabilities directly in the client side: a tile matrix set and a "pre-fetch" SensorThings API request for the number of features in a tile.

By defining a tile matrix set with several zoom levels on the client-side, parallel tiled requests based on this division can be sent to the server by specifying the bounding box of each tile without changing anything in the Standard or the server implementations. In addition, only the tiles with a number of objects below a threshold would be completely requested.

MiraMon was configured to issue a request for each STA tile twice, the first request is for getting the count (i.e. number of features of interest in a tile), and then the second request is for getting the actual features of interest. The number of features of interest is requested from the STA server in a prefetch request. If the count of the features in the "tile" requested is greater than the threshold, then the number of features of interest is represented in the center of the tile with the presentation of the number of objects in each tile. Otherwise, the full description of the features of interest in the tile is requested and each feature of interested is represented in the screen. This approach is proposed by the STAM library and it has been implemented from scratch in the MiraMon Map Browser using the same principles.

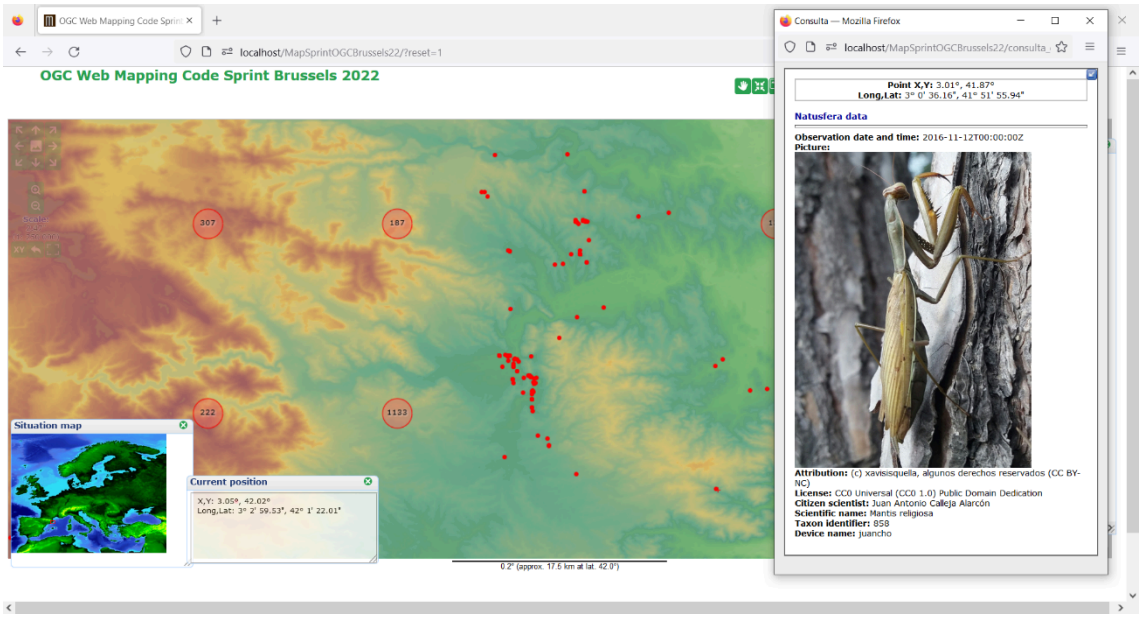


Figure 18 — Screenshot of the MiraMon Map Browser requesting some tiles with the count of features of interest represented as big circles and others with the features of interest represented as small red dots. In addition, the information related to a particular observation is shown, including a picture.

Figure 19 — Screenshot of the MiraMon Map Browser showing the URLs requested for each tile where the each bounding box can be seen expressed in WKT as a polygon.

5.6.2.3. A server-side implementation

Secure Dimensions contributed to the code sprint through a server-side implementation of a mechanism for creating tiles of observations. In this case, an OGC API - Tiles implementation is used as a facade for STA. The server implementing the OGC API - Tiles, renders the STA observation in PNG on-the-flight that is immediately delivered to the client. The MiraMon Map Browser is used as a client for this approach in order to visually verify that both server-side and client-side tiles generate the same view. Further work is needed to evaluate which of the two approaches could be more efficient in concrete use cases.

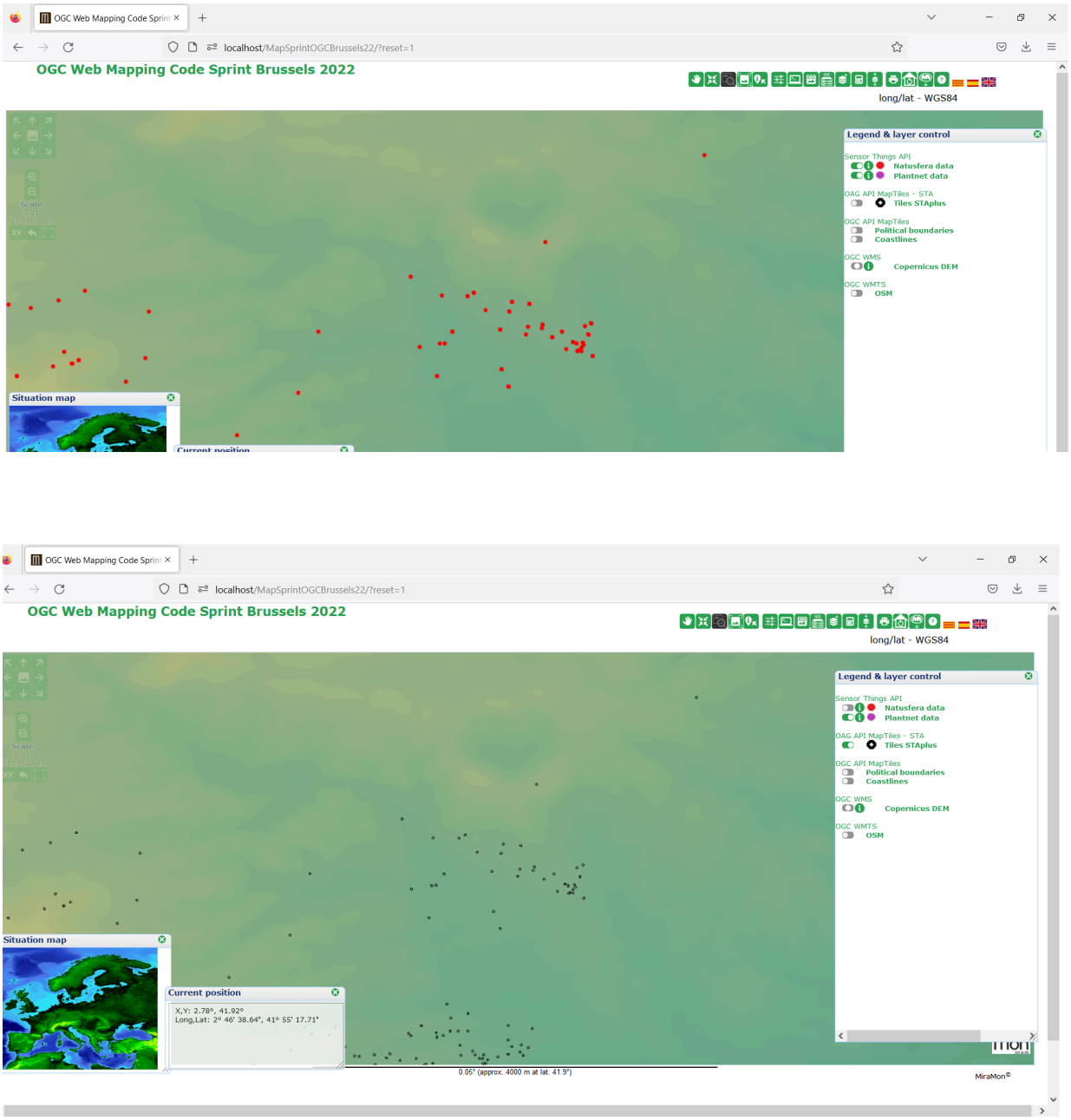


Figure 20 — Screenshot of the MiraMon Map Browser comparing the response of the the Secure Dimensions OGC API - tiles facade (bottom, represented as black dots) with a STA with the client side tile approach (top, represented as red dots) as described in the previous subsection. Spatial concordance between the position of the features of interest is checked.

5.7. Ordnance Survey Implementation of OGC API — Styles

Sprint participants from Ordnance Survey implemented a prototype that supports *OGC API — Styles*. The prototype was implemented to offer the landing page, API definition, conformance declaration, list of styles, and styles metadata. To demonstrate the prototype, Esri stylesheets were served through the API. The prototype was implemented to support both the British National Grid projection (which has the identifier EPSG:27700) and the WGS 84 / Pseudo-Mercator (which has the identifier EPSG:3857). A sample from the landing page is shown below.


```
{
  "title" : "OS NGD Styles API",
  "links" : [
    {
      "href" : "https://localhost/api/v1",
      "rel" : "self",
      "type" : "application/json",
      "title" : "this document"
    },
    {
      "href" : "https://localhost/api/v1/conformance",
      "rel" : "http://www.opengis.net/def/rel/ogc/1.0/conformance",
      "type" : "application/json",
      "title" : "list of conformance classes implemented by this API"
    },
    {
      "href" : "https://localhost/api/v1/styles",
      "rel" : "http://www.opengis.net/def/rel/ogc/1.0/styles",
      "type" : "application/json",
      "title" : "the styles shared via this API"
    }
  ]

}
```


In addition to supporting the publication of styles, the prototype was implemented to also support the management of styles. The functionality was demonstrated during the code sprint, using Postman to send the request for creating a style. Future work will involve implementation of input validation, handling of error responses, support for PATCH functionality, and support for the management of resources such as sprites and fonts. A sample of the styles list of the OS NGD Styles API prototype is shown below.


```
{
  "styles" : [
    {
      "id" : "ngd27700",
      "title" : "Style ngd27700 for OS NGD API",
      "links" : [
        {
          "href" : "https://localhost/api/v1/styles/ngd27700?f=esri",
          "rel" : "stylesheet",
          "type" : "application/json",
          "title" : "ESRI stylesheet for style ngd27700"
        },
        {
```

```
          "href" : "https://localhost/api/v1/styles/ngd27700/metadata?f=json",
          "rel" : "describedby",
          "type" : "application/json",
          "title" : "Metadata for style ngd27700"
        }
      ]
    },
    {
      "id" : "ngd3857",
      "title" : "Style ngd3857 for OS NGD API",
      "links" : [
        {
          "href" : "https://localhost/api/v1/styles/ngd3857?f=esri",
          "rel" : "stylesheet",
          "type" : "application/json",
          "title" : "ESRI stylesheet for style ngd3857"
        },
        {
          "href" : "https://localhost/api/v1/styles/ngd3857/metadata?f=json",
          "rel" : "describedby",
          "type" : "application/json",
          "title" : "Metadata for style ngd3857"
        }
      ]
    }
  ]
}
```


5.8. OGC Styles and Symbology

Sprint participants in a breakout session on OGC Styles and Symbology discussed potential Conformance Classes for a new OGC Styles and Symbology Standard. The sprint participants went through the conformance classes to check that none were missing, and found that some were indeed missing. An example of a conformance class that was missing is the Geometry Selection conformance class in relation to the possibility of a symbolizer accessing the underlying geometry to manipulate it. Such a capability could be applied to visualization of proportional symbols, for example, transforming a polygon into a point with the ST_Centroid function (which is defined in the OGC Simple Features 1.2.1 Standard) to draw a proportional symbol.

As several OGC Styles and Symbology SWG members participated in the breakout session, the discussions will help to move towards a version 2.0 “Styles & Symbology Model & Encodings” Standard that includes SymCore and the conceptual/logical definitions of the concrete extensions. The sprint participants started the documentation of the envisaged version 2.0 on GitHub. In terms of naming, the SWG members plan to keep the SymCore name for version 2.0.

In parallel, the sprint participants worked on relevant use cases to test the functional coverage of the proposed model and to relate them to the conformance classes. The use cases are presented in Annex A. In doing so, the sprint participants tested existing encodings (e.g. GeoCSS) to start imagining the “natural” encoding that could also be documented with Part 1 of the “Styles & Symbology Model & Encodings” candidate standard. The use cases will continue to

inspire and inform the work of the SWG for the next few months as the next major milestone will be the 125th OGC Member Meeting, to be held February 2023.

5.9. OGC API — Styles

During the code sprint, participants of interactive instruments addressed two open issues in the specification. A pull request was created to update the candidate standard.

The pull request

- aligned the names of metadata properties with the OGC Two Dimensional Tile Matrix Set and Tile Set Metadata Standard 2.0;
- clarified how to apply changes to a style with multiple occurrences in an API.

5.10. OWSLib

The OWSLib project implemented client capability for the OGC API — Records — Part 1: Core candidate Standard. Support for the Requirements Class "Map Core" was implemented, reviewed and approved by the OWSLib development team. As a result, Python clients can now use OWSLib for simple Pythonic workflows to request and visualize data from an OGC API — Maps service.

A screenshot of a sample Python/OWSLib workflow of the new functionality is shown in the figure below.

```
from owslib.ogcapi.maps import Maps

m = Maps('http://localhost:5000')

euroglobalmap = m.collection('euroglobalmap')

data = m.map('euroglobalmap',
             width=1200,
             height=800,
             bbox=[4.022369384765626, 50.690447870569436, 4.681549072265626, 51.00260125274477],
             bbox_crs='http://www.opengis.net/def/crs/EPSG/0/3857',
             transparent=True)

with open("output.png", "wb") as f:
    f.write(data.getbuffer())
```

Figure 21 — Screenshot of a sample Python/OWSLib workflow

A screenshot of the OWSLib OGC API — Maps demo is shown below.

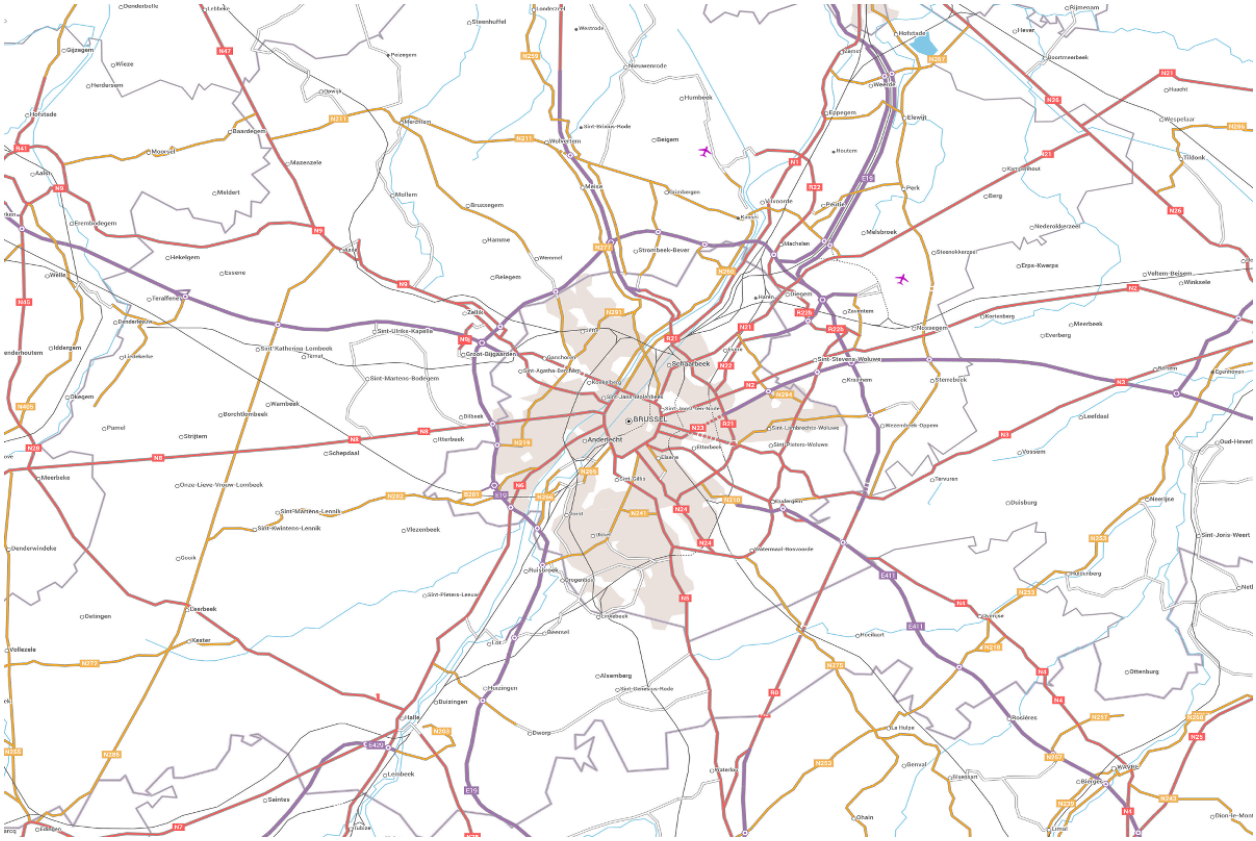


Figure 22 — Screenshot of the OWSLib OGC API - Maps demo

5.11. pygeoapi

5.11.1. OGC API — Maps implementation

The pygeoapi project implemented support for the OGC API — Maps — Part 1: Core candidate Standard. Support for the Requirements Class “Map Core” was implemented, reviewed and approved by the pygeoapi development team. As a result, pygeoapi implementations are now able to easily publish geospatial data as dynamic maps thanks to the building block approach of OGC API Standards.

pygeoapi’s plugin architecture facilitated the implementation of two plugin backends for maps:

- MapScript: using the MapServer web mapping engine as the map renderer, using data and styling definitions (Styled Layer Descriptor [SLD], MapServer CLASS configuration files).
- WMSFacade: a bridge/mediator to expose existing OGC WMS 1.3 services via OGC API — Maps

The official pygeoapi documentation provides more information on how to enable the new functionality. A screenshot of the associated OpenAPI/Swagger interface from the implementation is shown in the figure below.

MapScript

MapScript is MapServer's scripting interface to map rendering.

To publish a map via MapScript, the path to data is required, as well as the layer type (*options.type*). To style the data, set *options.style*. If no style is specified, the layer will be rendered with defaults.

MapServer layer types (*options.type*):

MS_LAYER_POINT

MS_LAYER_LINE

MS_LAYER_POLYGON

MS_LAYER_RASTER

Currently supported style files (*options.style*):

OGC Styled Layer Descriptor (SLD)

MapServer CLASS includes (i.e. file snippets with CLASS definitions)

```
providers:
    - type: map
      name: MapScript
      data: /path/to/data.shp
      options:
          type: MS_LAYER_POINT
          layer: foo_name
          style: ./foo.sld
      format:
         name: png
         mimetype: image/png
```

Figure 23 — Screenshot of the pygeoapi OGC API - Maps MapScript plugin

WMSFacade

To publish a WMS via pygeoapi, the WMS base URL (*data*) and layer name (*options.layer*) is required. An optional style name can be defined via *options.style*.

```
providers:
    - type: map
      name: WMSFacade
      data: https://demo.mapserver.org/cgi-bin/msautotest
      options:
          layer: world_latlong
          style: default
      format:
            name: png
            mimetype: image/png
```

Figure 24 — Screenshot of the pygeoapi OGC API - Maps WMSFacade plugin

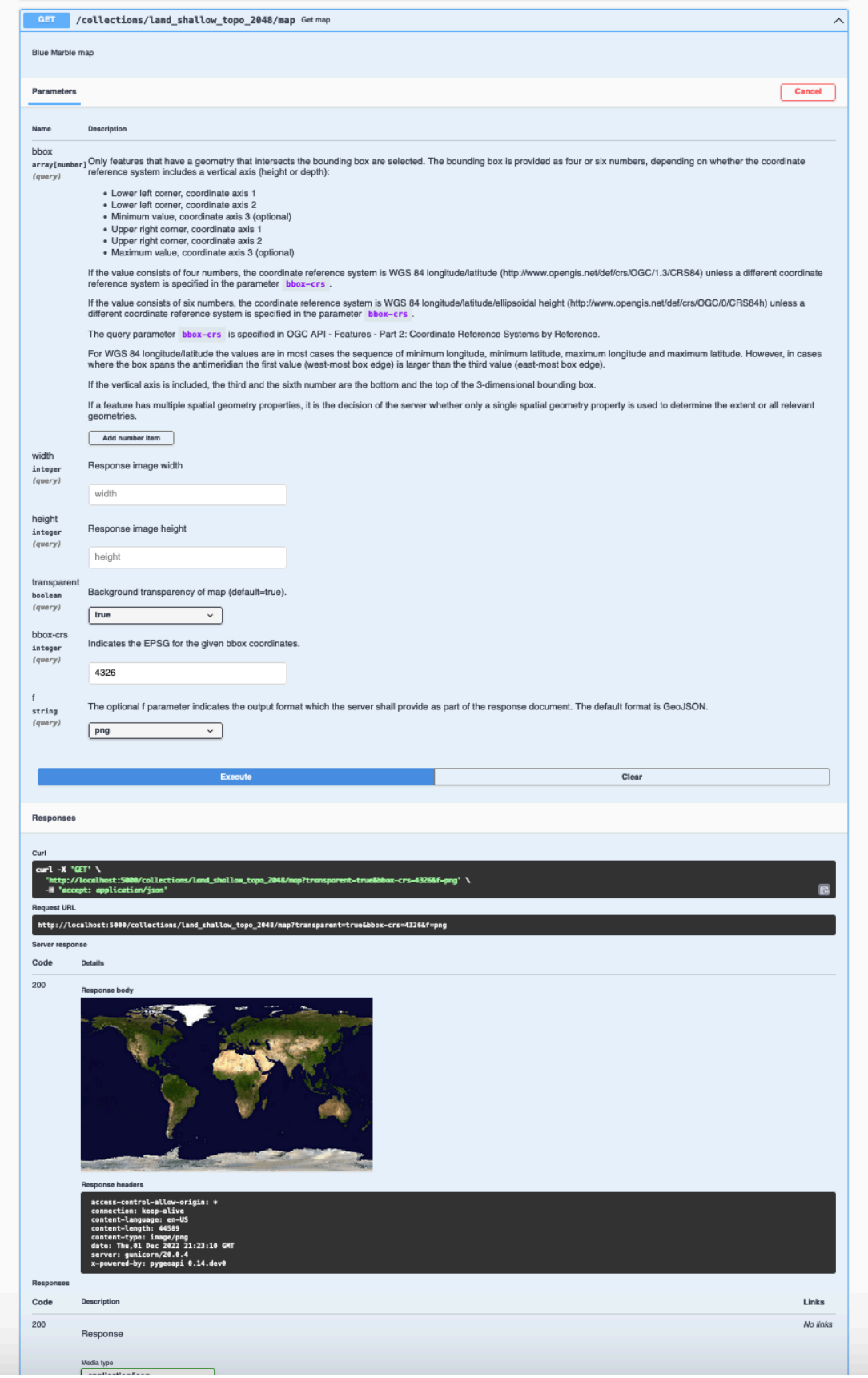


Figure 25 — Screenshot of the pygeoapi OGC API - Maps interface via OpenAPI/Swagger UI

Figure 26 — Screenshot of the pygeoapi OGC API - Maps WMSFacade using the Open Maps for Europe WMS from Eurogeographics

5.11.2. OGC API — Tiles updates

5.11.2.1. Tile Metadata

Support for the Tile Set Metadata schema has been added to TileJSON MapBox. It is also possible to support any format by adding a JSON metadata file directly into the filesystem. As well, the absence of metadata does not represent an issue, facilitating support for different tiles providers.

```
{
  "accessConstraints": "unclassified",
  "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
  "dataType": "vector",
  "description": "lakes of the world, public domain",
  "layers": [
    {
      "dataType": "vector",
      "id": "ne_110m_lakes"
    }
  ],
  "links": [
    {
      "href": "http://localhost:5000/collections/lakes/tiles/WorldCRS84Quad/{tileMatrix}/{tileRow}/{tileCol}?f=mvt",
      "rel": "item",
      "title": "WorldCRS84Quad vector tiles for ne_110m_lakes"
    }
  ],
  "tileMatrixSetURI": "http://schemas.opengis.net/tms/1.0/json/examples/WorldCRS84Quad.json",
  "title": "Large Lakes"
}
```


Figure 27

5.11.2.2. Vector tiles URL templates

The pygeoapi team added support for rendering vector tiles retrieved from an OGC API — Tiles implementation from a third-party service, using a generic URL template. One use case for this functionality would be to stream elasticsearch vector tiles, which are provided in elasticsearch >=8, as OGC API — Tiles.

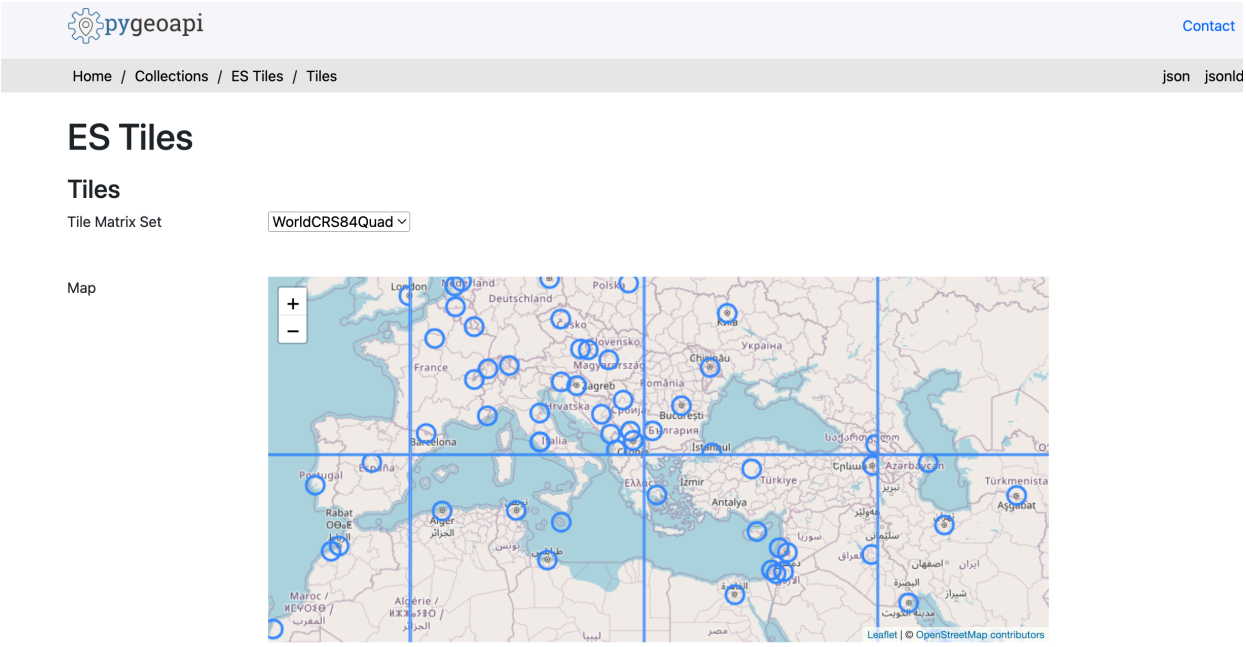


Figure 28 — Screenshot of pygeoapi OGC API - Tiles using the Elasticsearch vector tiles

Users could also use this functionality to render pg_tileserv vector tiles, or any other service which uses format `{z}/{y}/{x}` or `{z}/{x}/{y}`

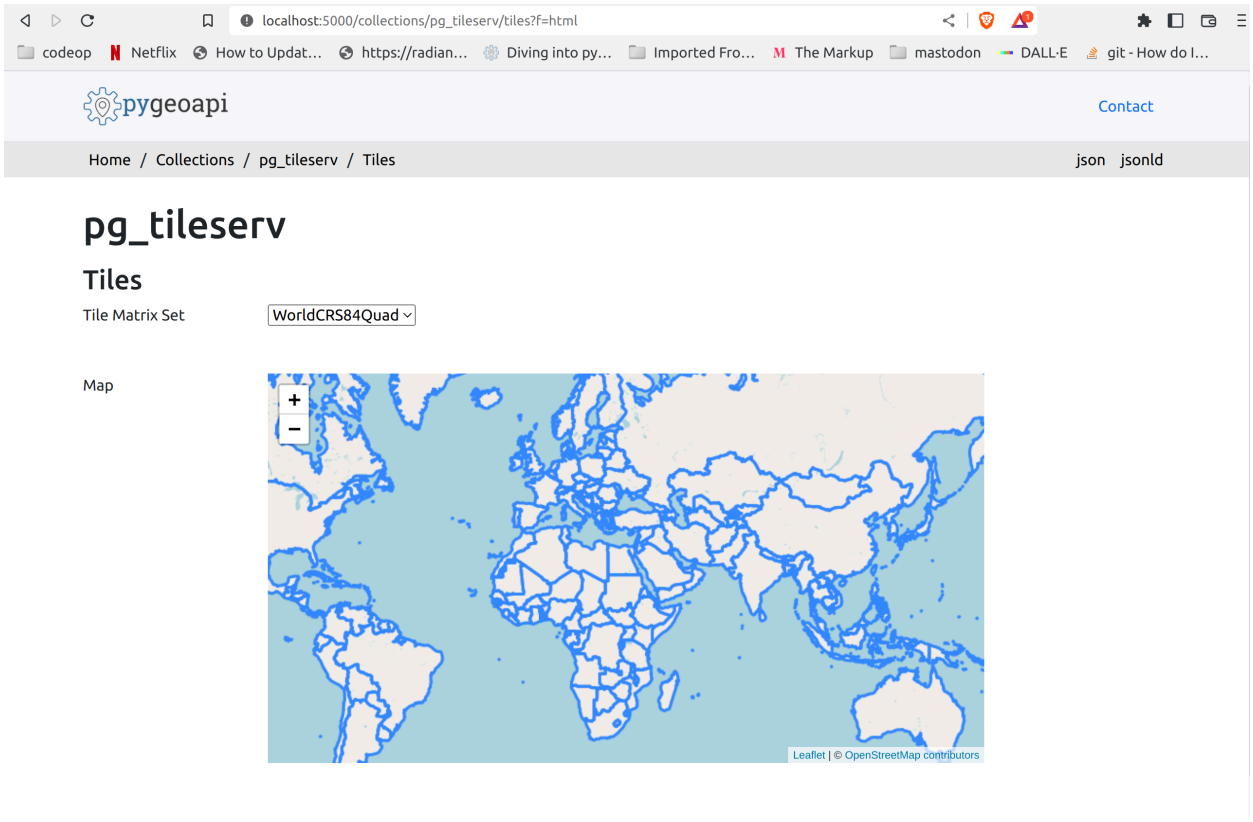


Figure 29 — Screenshot of pygeoapi OGC API - Tiles connecting to a pg_tileserv vector tiles service

5.11.3. OGC API — Coverages browser rendering

Through a lightweight HTML viewer, the team added HTML representation of coverages to pygeoapi. This viewer is a Leaflet implementation, which uses the Leaflet.ImageOverlay.Arrugator plugin, and the proj4js library to provide client-side reprojection. Whenever a web browser visits a `/coverage` path on pygeoapi, the web browser sends an `Accept: text/html` header which triggers pygeoapi to respond with the lightweight coverage viewer.

Figure 30 — Screenshot of pygeoapi lightweight coverage viewer

The emphasis is not in the viewer, but rather in the fact that the viewer is the HTML representation of the coverage resource. Whenever an API client requests the coverage resource, it will be given an HTML map viewer, or the JSON-LD representation of the coverage,

or the raw coverage data (as PNG, GeoTIFF, NetCDF, etc), in accordance with the established HTTP content negotiation rules (“Accept” HTTP headers, or “f” URL parameter).

This is aligned with pygeoapi’s implementation of OGC API — Features, where requesting features as HTML returns a map viewer loaded with those features.

5.11.4. Mentor session

A pygeoapi mentor session was provided in support of delivering vector tiles using pygeoapi. The session focused on:

- installation
- overview of various endpoints and creating a standard source
- creating vector tiles
- serving vector tiles
- reading vector tiles from different clients

5.11.5. Developer outreach

Members of the pygeoapi development team discussed project participation and contributions with various developers at the sprint.

5.12. TEAM Engine

The focus of work related to TEAM Engine was on the OGC CITE executable test suite (ETS) OGC API — Tiles.

The work was structured according to the GitHub issues related to the ETS. In total, five issues were worked on as part of the code sprint. The following screenshot summarizes those issues.

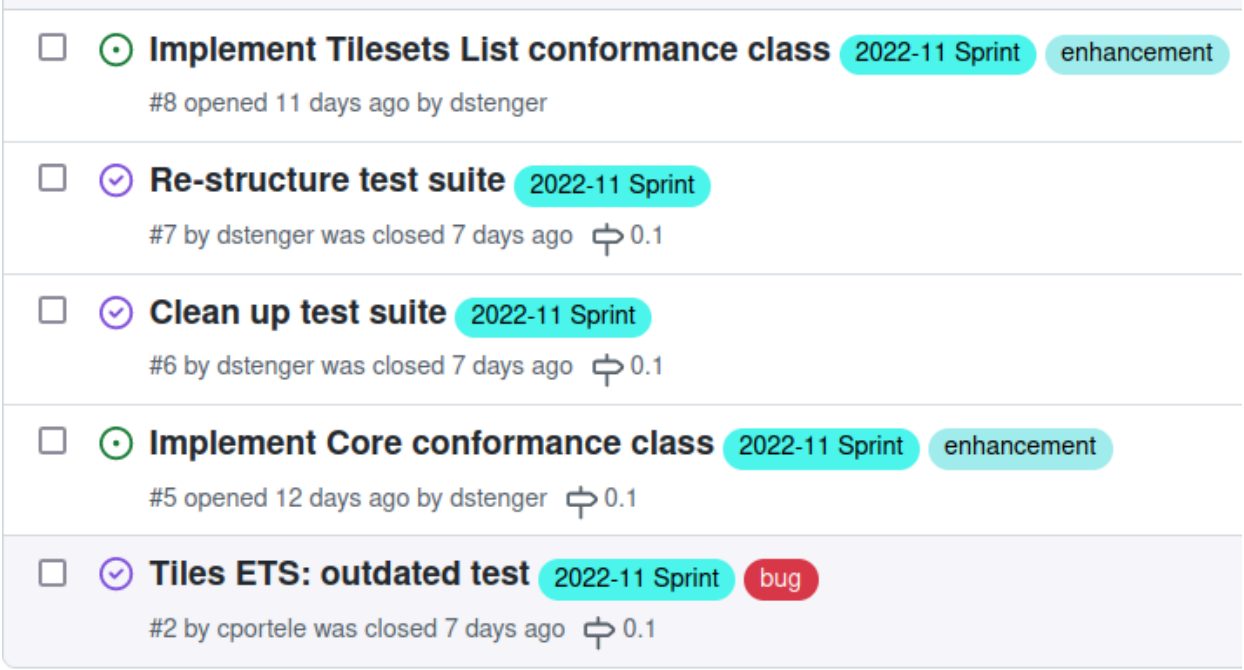


Figure 31 — Screenshot of GitHub issues created during code sprint

The team, which consisted of personnel from lat/lon, refactored the ETS so that the executable tests are closer to the structure of the abstract test suite of the standard. Afterwards, the test implementation of the Core conformance class was enhanced. This included the parsing of the OpenAPI document which is required to extract parameters needed to build requests. The parsing of the OpenAPI document is also part of the Tilesets List conformance class. In addition, an already reported bug was solved.

The result can be summarized by a screenshot of a sample HTML report created by the ETS, running from within TEAM Engine. The report structures the already existing tests into the correct conformance classes and gives an overview of already implemented tests.

Results for session s0001

Test Name: ogcapi-tiles-1.0
Test version: 0.1-SNAPSHOT
Time: 2022-12-12T16:51:53.616Z

Test INPUT:
Tested Instance : https://demo.ldproxy.net/daraa

Result:
Passed Core conformance classes (Implementations passing these classes can be certified): Yes
Number of conformance classes tested: 4
Number of conformance classes passed: 4
Number of conformance classes failed: 0

Core conformance classes (Pass = Green; Fail = Red; Skip = Grey):
Core

Color Legend	Pass	Fail	Skip

OpenAPI Specification 3.0

Pass: 1	Fail: 0	Skip: 0	Total tests: 1

Name	Reason
api Definition Validation	

Core

Pass: 1	Fail: 0	Skip: 0	Total tests: 1

Name	Reason
validate Conformance Operation And Response	

Tilesets List

Pass: 1	Fail: 0	Skip: 0	Total tests: 1

Name	Reason
validate Tilesets List Response	

Dataset Tilesets

Pass: 2	Fail: 0	Skip: 0	Total tests: 2

Name	Reason
tiles Landing Page Validation	
landing Page Retrieval	

See the detailed old test report.

Execute this session again | Delete this session | Download log Files

Sessions list

TEAM Engine 5.5.2

If you have any questions or suggestions, feel free to contact the site administrator.

Figure 32 — Screenshot of HTML report of OGC API - Tiles test suite created by TEAM Engine

5.13. OpenLayers

5.13.1. OGC API – Tiles updates

A previous OpenLayers release had ‘experimental’ support for rendering tiles from OGC API – Tiles implementations. This was added prior to the OGC API – Tiles Standard being finalized.

During the sprint, the implementation was tested against services conforming to the Tileset conformance class. Changes were proposed to graduate the source classes for rendering vector and map tiles from ‘experimental’ to ‘stable’. These changes were approved, and the classes will be part of the OpenLayers API in the next release.

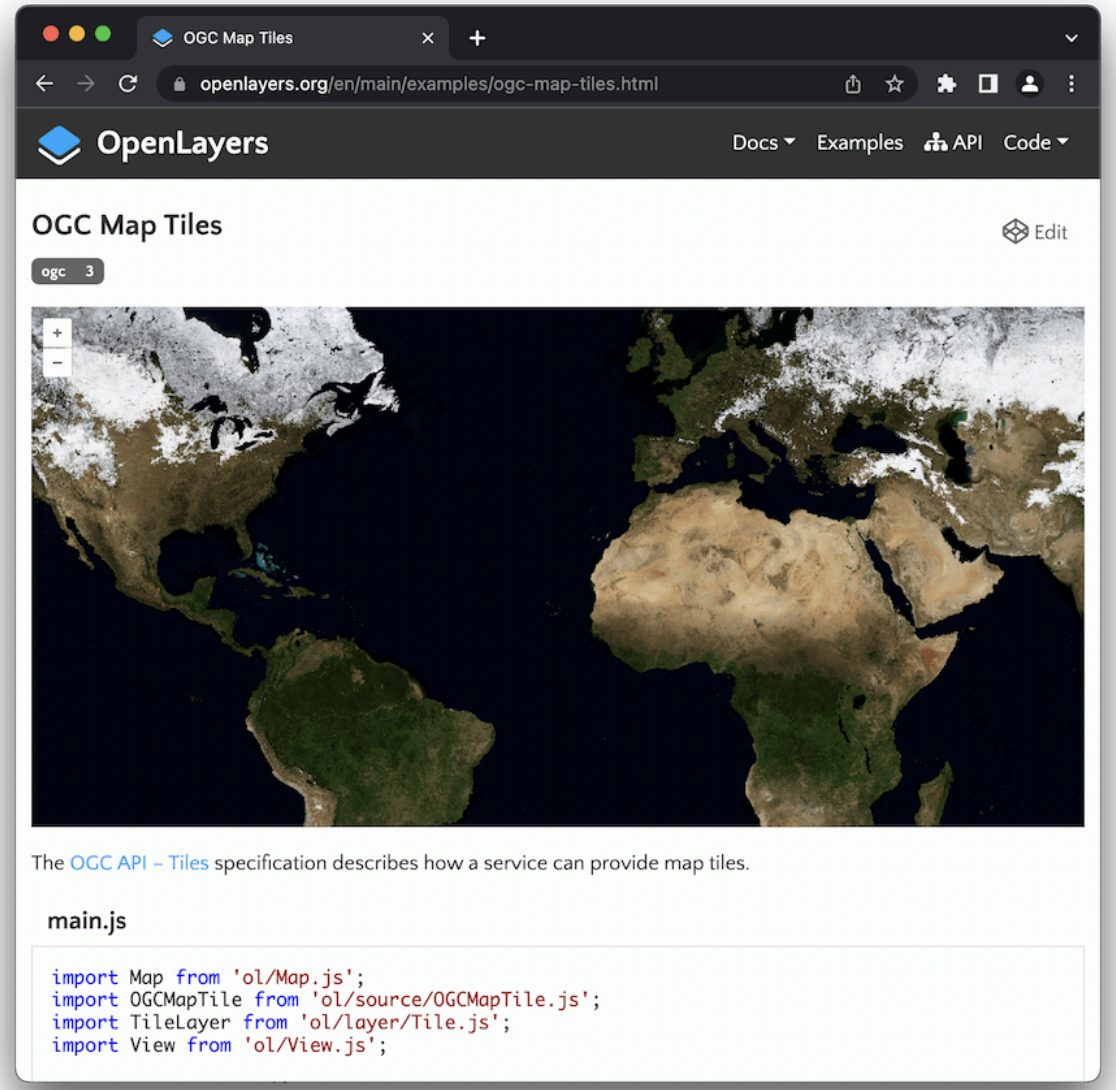


Figure 33 — Map tiles from an OGC API – Tiles service in OpenLayers

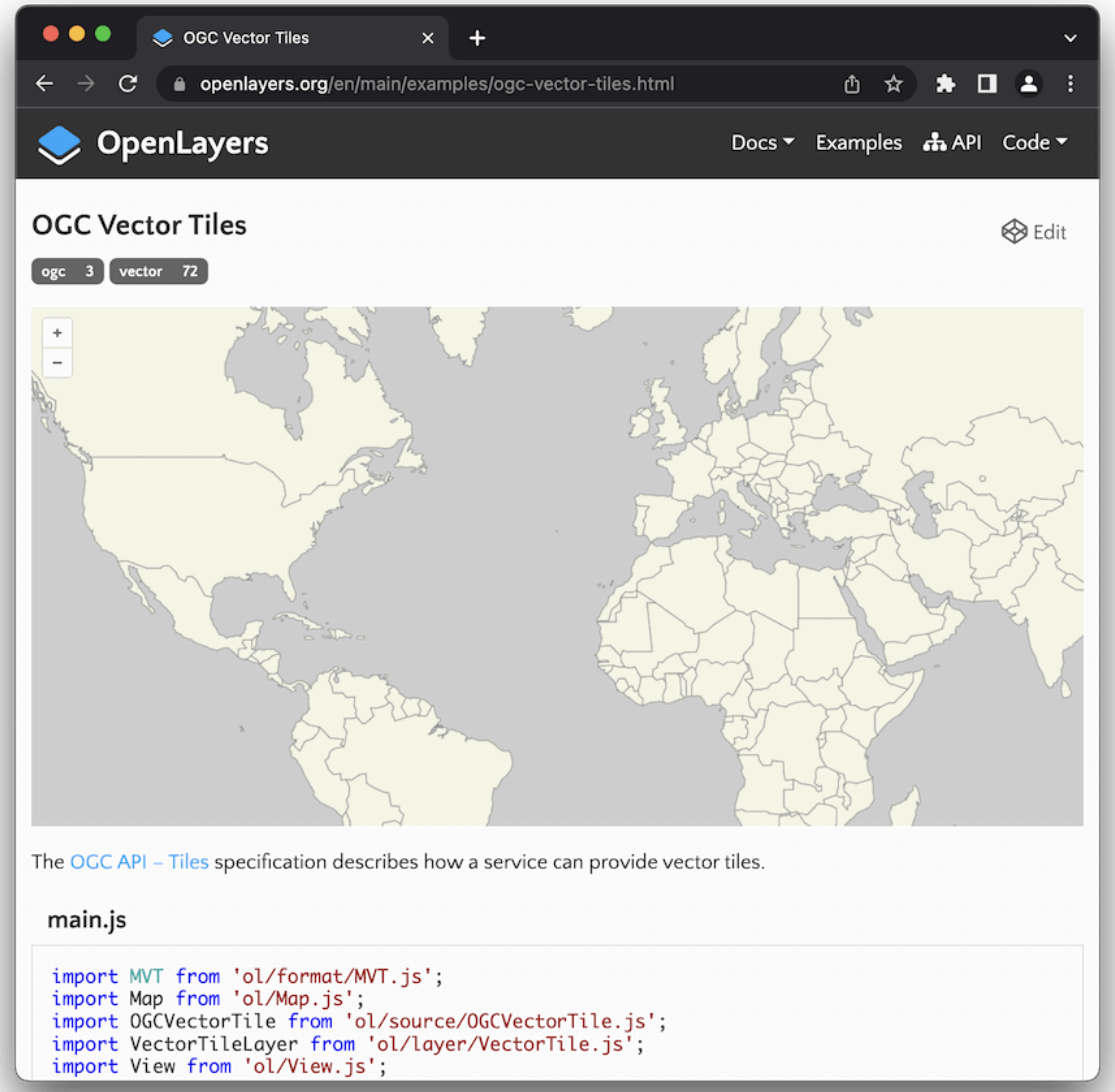


Figure 34 — Vector tiles from an OGC API – Tiles service in OpenLayers

5.14. xyz2ogc

The xyz2ogc program was developed to demonstrate how metadata can be added to XYZ tilesets to conform with additional OGC API – Tiles conformance classes. A goal of the project was to investigate how much of the OGC API – Tiles Standard can be implemented as a "static" service.

The xyz2ogc program has two primary commands: `serve` and `generate`. The program is configured with a single TOML file that lists URL templates and additional optional metadata

for existing XYZ tilesets. The `serve` command reads the configuration file and starts a local webserver that provides OGC API – Tiles compliant metadata. The `generate` command writes out a set of metadata files that can be hosted as a static website conforming to the specification.

The program generates metadata that conforms to the following OGC API – Tiles conformance classes:

- http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/core
- http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/json
- http://www.opengis.net/spec/ogcapi-common-1/1.0/req/oas30
- http://www.opengis.net/spec/ogcapi-tiles-1/1.0/conf/core
- http://www.opengis.net/spec/ogcapi-tiles-1/1.0/conf/tileset
- http://www.opengis.net/spec/ogcapi-tiles-1/1.0/conf/tilesets-list
- http://www.opengis.net/spec/tms/2.0/conf/tilematrixset
- http://www.opengis.net/spec/tms/2.0/conf/tilesetmetadata
- http://www.opengis.net/spec/tms/2.0/conf/json-tilematrixset
- http://www.opengis.net/spec/tms/2.0/conf/json-tilesetmetadata
- http://www.opengis.net/spec/tms/2.0/conf/tilematrixsetlimits
- http://www.opengis.net/spec/tms/2.0/conf/json-tilematrixsetlimits

5.14.1. Findings

The OGC API – Tiles Standard makes it easy for existing XYZ tilesets to conform with the Core requirements. This is possible as long as the service provides some way for the tile URL template and the meaning of the variables in that template to be obtained.

Since many existing XYZ tilesets do not have a formalized way for clients to obtain the tile URL template or to understand the meaning of the variables, it makes sense for tile providers to "upgrade" their services by conforming with the Tileset requirements.

In practice, conforming with the Tileset requirements means hosting two metadata documents for a single XYZ tileset: one providing the tileset metadata and one describing the tiling scheme. This can be reduced to a single document for the tileset metadata linking to an existing document hosted elsewhere that describes the tiling scheme.

It would be more convenient if OGC Standards were developed in a way that URIs for things like CRS or TileMatrixSet resolved to definitions of those resources. This could be implemented through Content Negotiation, as proposed by Issue#124 on the OGC Naming Authority GitHub repository. For example, it would be convenient if the WebMercatorQuad URI resolved to the definition for this tile matrix set. This would allow the tiling scheme for a tileset to be specified with the tile matrix set URI alone, and clients could reliably fetch the definition from this.

Unfortunately, it is not enough for the OGC URIs to redirect to a different resource providing this content. A browser-based client served via the HTTPS protocol will not fetch the OGC URI unless it is served via HTTPS (to avoid mixed active content violations) and it is served with Cross-Origin Resource Sharing (CORS) headers.

Suggestion: If in the future OGC URI 1) used the HTTPS scheme, 2) resolved to useful resources, and 3) were served with CORS headers, it would make browser-based client implementations easier.

For services providing more than a single tileset, it makes sense to conform to the Tileset List requirements. Having conformed with the Tileset requirements, it is mostly straightforward to conform to the Tileset List requirements by providing a document that lists the metadata (or a subset of the metadata) for each tileset. In the case of a static server, requirement 9 adds a small additional burden on implementers. The specification requires that the URL for the list of tilesets ends with `/tiles`. In many cases, static servers determine the content type for a response based on a document extension. For example, a common pattern is to have an `index.html` document that the server will provide in response to a request that matches a directory location (e.g. https://example.com/path/to/directory/). The same can be done for an `index.json` document with a list of tilesets as long as the service can be configured to treat `index.json` files as the "default" document. The additional potential complication is that the service may need to be configured to redirect a slash-less URL (e.g. https://example.com/path/to/tiles) to a URL ending in a slash (e.g. https://example.com/path/to/tiles/) so that the default document will be served.

Suggestion: If there is not a real benefit to requiring that the tileset list URL ends with `/tiles`, perhaps a future version of the Standard could relax or remove this requirement.

In addition to the Tileset, Tileset List, and associated TMS conformance classes, the `xyz2ogc` program implements a few of the conformance classes from OGC API – Common, namely Core, JSON, and OpenAPI 3.0. However, `xyz2ogc` chose not to implement the additional requirements from the OGC API – Tiles OpenAPI 3.0 conformance class. In particular, the requirements related to homogenous collections of tilesets (e.g. not including vector tiles in a list that includes map tiles) didn't match the design of the project (a single listing of tilesets that may include both vector and map types). A client capable of rendering both vector and map tiles can make this distinction invisible to users, and the idea with the `xyz2ogc` program is to provide users with a list of tilesets to choose from – dividing those into multiple lists based on tile content type seems arbitrary and a bit awkward.

Suggestion: In the future, it could be useful if the Standard allowed for linking to tilesets lists of mixed types.

6

DISCUSSION

6 DISCUSSION

6.1. JPEG or PNG responses

JPEG images are typically an order of magnitude smaller in byte size than PNG images of a similar width and height in pixels. JPEG images therefore result in a much more responsive experience, but they do not support transparency, so layer overlays are impossible. PNG images, on the other hand, do support transparency, but are typically larger in byte size for a similar width and height in pixels. Some client implementations deal with this by requesting the bottom layer as JPEG and all overlay layers as PNG, but this is far from ideal. One somewhat common solution is to support a "JPEG or PNG" format (represented by a requested content type of "image/x-jpegorpng" or "image/jpgpng" or a format string of "jop"). When this format is requested, the server can return the requested image in the optimal image format (typically PNG if there's transparency and JPEG otherwise). Several implementations, including CubeSERV, have taken this approach, and it has proven to be a highly effective optimization. However, the WMS and WMTS specifications failed to standardize such a request format, so this optimization is not interoperable.

This is not an issue for the "OGC API — Maps" candidate Standard and the "OGC API — Tiles" Standard, since they support format (i.e., content type) negotiation through use of the HTTP `Accept` request header. A client can simply indicate `Accept: image/png, image/jpeg`, which literally means "give me whichever of these two formats you consider the most appropriate". And as a bonus, a typical web browser will send an `Accept` string which gives image/jpeg and image/png equal weighting, so JPEG-or-PNG negotiation can automatically occur even through simple web-browser requests. It should be noted that some modern web browsers send an `Accept: image/avif, image/webp, /` header, which suggests that they prefer some of the newer formats which also happen to support an alpha channel. Therefore, use of the `Accept` header as already implemented by OGC API Standards also enables support for other encoding options.

However, there is a concern that many implementers of OGC API — Maps and OGC API — Tiles might not recognize the importance of this logic, and would likely write code which just chooses the first supported format in the list. There is therefore a need for awareness of this mechanism because such a mechanism is important enough to be explicitly provided as a recommendation in both OGC API — Maps — Part 1: Core and OGC API — Tiles — Part 1: Core.

A Recommendation could be added to an OGC API — Maps User Guide or some other developer resource. The sprint participants suggested that the SWG considers the following text for addition to a User Guide:

- If both "image/png" and "image/jpeg" are supported at an endpoint and are present in the HTTP "Accept" request header with the same q value, and no other supported output format is present with a higher q value, the server should return the requested

image in whichever of these two formats is considered optimal (typically PNG if there's transparency and JPEG otherwise).

Regarding the "f" parameter, if servers implement an "f" parameter that overrides the HTTP "Accept" request header, the servers would have the option of supporting custom values such as "jop" or "image/x-jpegorpng" which would trigger the optimization. However, since "jop" or "image/x-jpegorpng" are not standardized parameter values for indicating an encoding, the Standards need not provide any recommendations to this effect.

6.2. Data Statistics

Many clients have the need to retrieve a **feature count** element from the server, in order to construct their visualizations. OGC API — Features addresses this need through a property called "numberMatched" when a feature collection is retrieved. However, this leads clients sometimes to issue two separate get items requests: one to count the items, and another one to actually draw them.

There is a parallel with OGC API — Coverages as one of the use cases is fetching the coverage metadata (width, height, dimensions, bit depth, time dimension, number of channels/bands, native CRS) but not the data itself. That is how the pygeoapi viewer implementation works as of now.

OGC API — Tiles addresses this issue by offering tile set metadata and TileMatrixSet definitions. However, a similar ability to retrieve data statistics without retrieving the data appears to be missing from OGC API — Features and OGC API — Coverages.

The sprint participants suggested that a super class for presenting data statistics could be specified in OGC API — Common and then specialized in other OGC API Standards.

6.3. Cancel Requests

When requesting tiles, and panning around, clients often leave behind requests from tiles which are no longer in the viewport. This has a performance cost, as the server gets overloaded with requests which are no longer needed. One way of handling this situation, would be to fire a cancel request, for tiles which are no longer in the viewport. Some popular browsers, such as Mozilla Firefox, support this capability through the XMLHttpRequest.abort() method.

The sprint participants suggested the following as recommendations:

- Clients should be able to send a cancel or abort request so that a server aborts a long-running operation.

- Clients for OGC API — Maps should make a best effort to avoid making several requests in shorts periods of time (e.g. 1 second)
- Servers for OGC API — Maps can reply with HTTP code `429 Too Many Requests` if a client is straining the server resources

6.4. Security/access control

The EuroGeographics WMS access token as applied in OGC API — Maps via pygeoapi brought forth questions about how cascading / broker services deal with secure services. The pygeoapi documentation provides an overview on security and access control. The September 2020 OGC API Code Sprint included a security thread (OGC 20-091), however since then there has not been a similar thread in OGC Code Sprints. There was a suggestion to have a sprint (or thread in same) focused on general security/access control issues, across OGC API and broader OGC Standards. There are several implementations of OGC API Standards that implement access control mechanisms. Such a code sprint could pull in lessons learnt from many of the OGC API implementations that support access control, for example Ordnance Survey's NGD API — Features, NLS Finland's Topographic Database, and those from the OGC Testbeds (OGC 21-020r1).

6.5. Filtering of styles

There was a discussion about whether it is possible to filter styles returned by an implementation of OGC API — Styles in a standardized way. Currently, just like with OGC API — Features, a client simply receives a list of collections without the ability to filter the collections according to specific criteria. There have been suggestions that all collections should have the ability to be filtered. This would support the ability to filter on a particular field, such as the description. A client would only get the list of styles that are relevant to that filter. Whereas it is possible to do, at present this capability is not standardized.

6.6. HTML Previews

There was a discussion about the retrieval of HTML previews of data from an OGC API implementation. This followed the work described in Clause 5.11.3 regarding a coverage viewer for pygeoapi. It was noted that the ability to request an HTML representation of a dataset means that an OGC API could potentially also offer support for querying a dataset through a web browser. This leaves the question of how a query language such as CQL2 could be applied to tile matrix sets. Further, the HTML Preview capability creates an opportunity to enable the actual data to be queried such that the value of a location on a coverage can be retrieved rather

than the RGB (Red-Green-Blue) values of the image that is presented. The sprint participants noted that this could be a topic for a testbed or another type of innovation initiative.

Another topic discussed in relation to HTML Previews was that the `Accept` header or "f" parameter indicated the format of the response payload, but not the format of any embedded elements. So there is an expectation that the web browser supports whichever image format is embedded in the HTML response of the API. One option is to introduce a recommendation that if one of the media types is `text/html`, and the other is an image, then the server should return an HTML Preview that contains an image in the format specified. The sprint participants also noted that HTML offers a `<picture>` element which contains an `<img>` element, as well as zero or more `<source>` elements that offer alternative versions of an image for different situations HTML. A web browser considers each `<source>` element and selects the most appropriate among them. If there are no appropriate options, the web browser uses the URL in the `src` attribute of the `<img>` element.

6.7. Creating and Updating Styles

There are two issues regarding the HTTP PUT method and how it might be applied to OGC API — Styles. The first issue is related to the fact that there can be different stylesheets for a single style. An API can offer a resource in multiple encodings, for example SLD, ArcGIS, MapInfo, Mapbox or others. This leads to the questions of: Should changes to a resource be propagated to each of the encodings? What happens in situations where the encodings do not support a concept that is supported by other encodings? A pull request was created that addresses aspects of this challenge. The pull request will be discussed by the SWG after the code sprint.

The second issue is regarding specification of the PUT mechanism in the OGC API — Styles candidate Standard itself. *OGC API — Features — Part 4* was the first of the OGC API candidate Standards to specify interfaces for create, replace, update, and delete operations. *OGC API — Features — Part 4* has two different requirements classes, one is general and is written for any resource. The other requirements class is for features. *OGC API — Styles* implements the general resource requirements class — which could be eventually moved to *OGC API — Common*. The idea is for the general resource requirements class to support the ability to create a style when it receives a PUT request with an identifier.

7

CONCLUSIONS

7 CONCLUSIONS

The code sprint successfully facilitated the development and testing of prototype implementations of OGC API Standards, including candidate Standards, that relate to web mapping. Further, the code sprint provided a foundation for the development of the next version of the Symbology Core Standard. Furthermore, sprint participants were able to provide feedback directly to the editors of the Standards, whereas the editors were able to clarify any issues encountered by the sprint participants. The code sprint also raised awareness about the Standards. The code sprint therefore met all of its objectives.

7.1. Future Work

The sprint participants made the following recommendations for future innovation work items:

- Prototyping and demonstration of support for the querying of coverages in HTML Previews of coverage tiles
- A testbed activity to implement the prototype Symbology Core v2.0 would help to evaluate support for the various use cases identified in this sprint
- A testbed activity to evaluate options for access control across OGC API Standards (covering at least the Features, Records, Tiles, and Styles APIs)
- A testbed activity analyzing the potential for consistent modification and harmonization of different stylesheets (e.g. SLD, Mapbox styles, ArcGIS, MapInfo)

The sprint participants also made the following recommendations for things that the SWGs should consider:

- A code sprint focused on general security/access control issues, across OGC API and broader OGC Standards
- Design an approach for supporting 'Cancel' requests, so that long-running jobs can be terminated as users zoom or pan across a map
- A user guide for OGC API — Tiles would help to accelerate uptake (e.g. describing how to build static tile servers in the Cloud)
- A future version of the OGC API — Tiles Standard could relax or remove the requirement that the tileset list URL ends with `/tiles`

It is envisaged that the SWGs will consider the above-listed recommendations for future work items.

A

ANNEX A (INFORMATIVE) STYLES AND SYMBOLOGY USE CASES

A ANNEX A (INFORMATIVE) STYLES AND SYMBOLOGY USE CASES

This annex presents the Styles and Symbology use cases that were discussed and documented during the code sprint.

A.1. Graduated and Categorized: conditional coloring of a map

This use case is discussed in styles-and-symbology#Issue 17.

Existing conformance classes required for this use case

- Symbology Core: Class Symbolizer
- Example stylesheet:

Describe any capability not already addressed by existing conformance classes

- Categorized: coloring map according to a unique value e.g. coloring map according to region name (unique value)
- Graduated: coloring map according to a range of values e.g. coloring map according to the population of a zone in a particular range (0-5000, 5000-10000)

Example image illustrating the portrayal use case:

- Categorized:

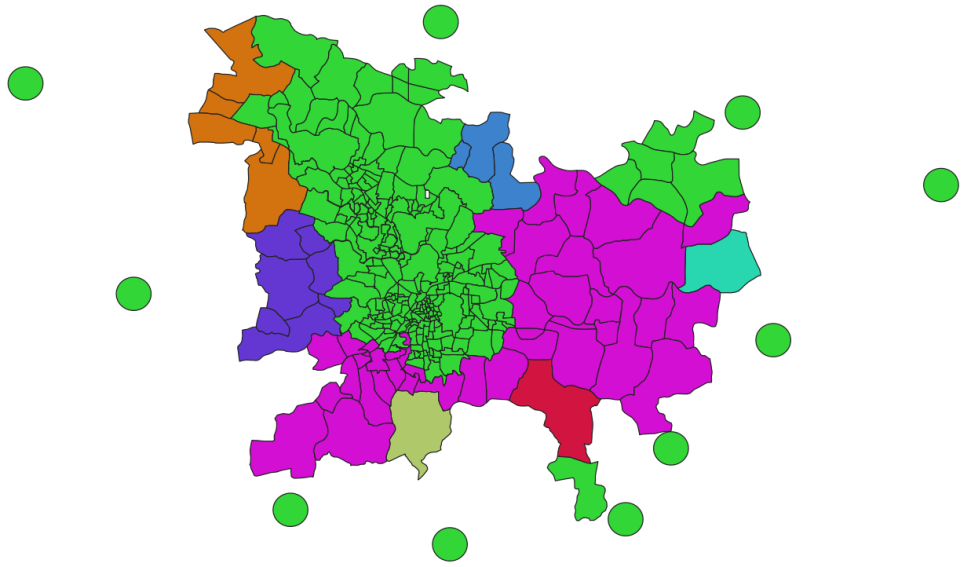


Figure A.1 — Categorized conditional coloring of a map

- Graduated

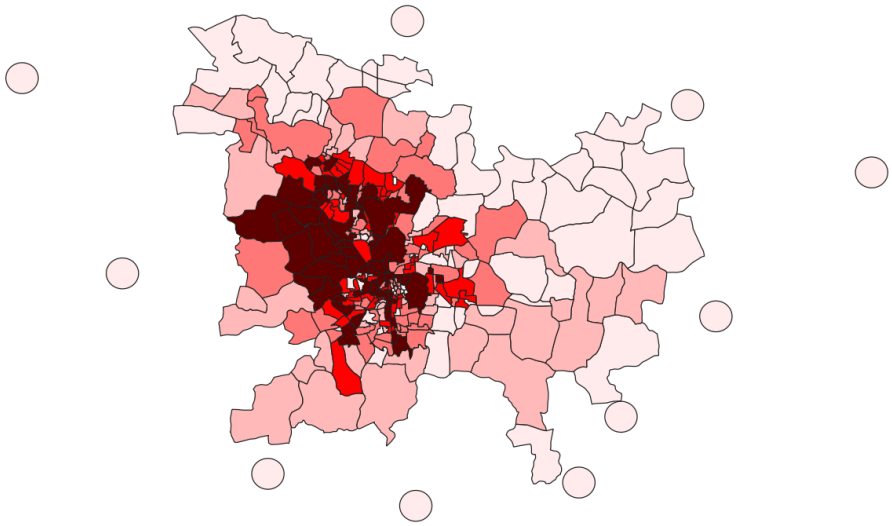


Figure A.2 — Graduated conditional coloring of a map

One possible solution is that when classes are unknown (in a raster map) then pixel gradient values can be used to group the pixels into classes and to color them according to that class, creating a choropleth or graduated map. In this use case, the vector data is presented as graduated (color maps) even where there is a large number of unknown classes.

A.2. Unique values map

This use case is discussed in styles-and-symbology#Issue 18.

Existing conformance classes required for this use case

- basic vector feature styling

Example stylesheet:

Below is a CSS encoding to describe a unique values map to display Local Climate Zones type (LCZ_PRIMARY property).

```
/* Metadata
* @title Local Climate Zones I
* @abstract Local Climate Zones I
*/

/* Main rule*/

* {
  stroke: rgb(255,255,255);
  stroke-width: 0.26;

/* Styling according to the Local Climate Zones */

  /* @title LCZ 1: Compact high-rise */
  [LCZ_PRIMARY = 1] {
    fill: rgb(139, 1, 1);
  };
  /* @title LCZ 2: Compact mid-rise */
  [LCZ_PRIMARY = 2] {
    fill: rgb(204, 2, 0);
  };
  /* @title LCZ 3: Compact low-rise */
  [LCZ_PRIMARY = 3] {
    fill: rgb(252, 0, 1);
  };
  /* @title LCZ 4: Open high-rise */
  [LCZ_PRIMARY = 4] {
    fill: rgb(190, 76, 3);
  };
  /* @title LCZ 5: Open mid-rise */
  [LCZ_PRIMARY = 5] {
    fill: rgb(255, 102, 2);
  };
  /* @title LCZ 6: Open low-rise */
  [LCZ_PRIMARY = 6] {
    fill: rgb(255, 152, 86);
  };
  /* @title LCZ 7: Lightweight low-rise */
  [LCZ_PRIMARY = 7] {
    fill: rgb(251, 237, 8);
  };
  /* @title LCZ 8: Large low-rise */
  [LCZ_PRIMARY = 8] {
    fill: #bcbcba;
```

```
  };
  /* @title LCZ 9: Sparsely built */
  [LCZ_PRIMARY = 9] {
    fill: rgb(188, 188, 186);
  };
  /* @title LCZ 10: Heavy industry */
  [LCZ_PRIMARY = 10] {
    fill: rgb(87, 85, 90);
  };
  /* @title LCZ A: Dense trees */
  [LCZ_PRIMARY = 101] {
    fill: rgb(0, 103, 0);
  };
  /* @title LCZ B: Scattered trees */
  [LCZ_PRIMARY = 102] {
    fill: rgb(5, 170, 5);
  };
  /* @title LCZ C: Bush,scrub */
  [LCZ_PRIMARY = 103] {
    fill: rgb(100, 132, 35) ;
  };
  /* @title LCZ D: Low plants */
  [LCZ_PRIMARY = 104] {
    fill: rgb(187, 219, 122);
  };
  /* @title LCZ E: Bare rock or paved */
  [LCZ_PRIMARY = 105] {
    fill: rgb(1, 1, 1);
  };
  /* @title LCZ F: Bare soil or sand */
  [LCZ_PRIMARY = 106] {
    fill: rgb(253, 246, 174);
  };
  /* @title LCZ G: Water */
  [LCZ_PRIMARY = 107] {
    fill: rgb(109, 103, 253);
  };
}
```

Describe any capability not already addressed by existing conformance classes

Example image illustrating the portrayal use case:

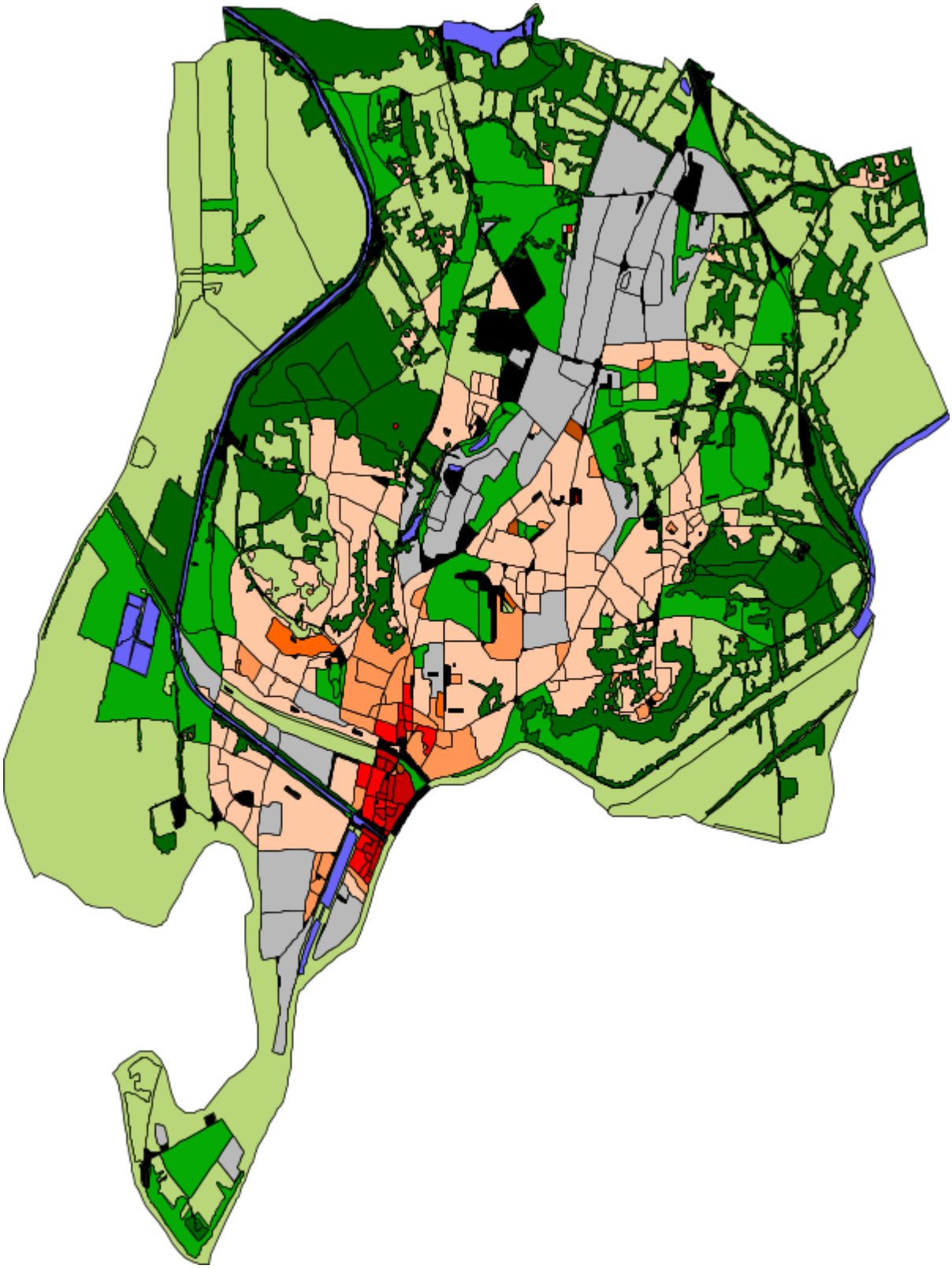


Figure A.3 — Unique values map

Material:

- Data : https://github.com/orbisgis/POC-Carto/blob/main/data/rsu_lcz.geojson
- Tool : The POC-Carto is based on the Geotools library.

A.3. Choropleth or graduated map

This use case is discussed in styles-and-symbology#Issue 19.

Existing conformance classes required for this use case

- basic vector feature styling

Example stylesheet:

Below is a CSS encoding to describe a graduated map that represents a fraction of vegetation on a grid (HIGH_VEGETATION_FRACTION property).

```
*{
  stroke: #000000;
  stroke-width: 1px;
  [HIGH_VEGETATION_FRACTION>=0 OR HIGH_VEGETATION_FRACTION<=0.200]
  {
     fill: yellow;
  };
  [HIGH_VEGETATION_FRACTION>0.200 OR HIGH_VEGETATION_FRACTION<0.500]
  {
     fill: orange;
  };
  [HIGH_VEGETATION_FRACTION>0.500]
  {
     fill: red;
  };
}
```

Describe any capability not already addressed by existing conformance classes

Example image illustrating the portrayal use case:

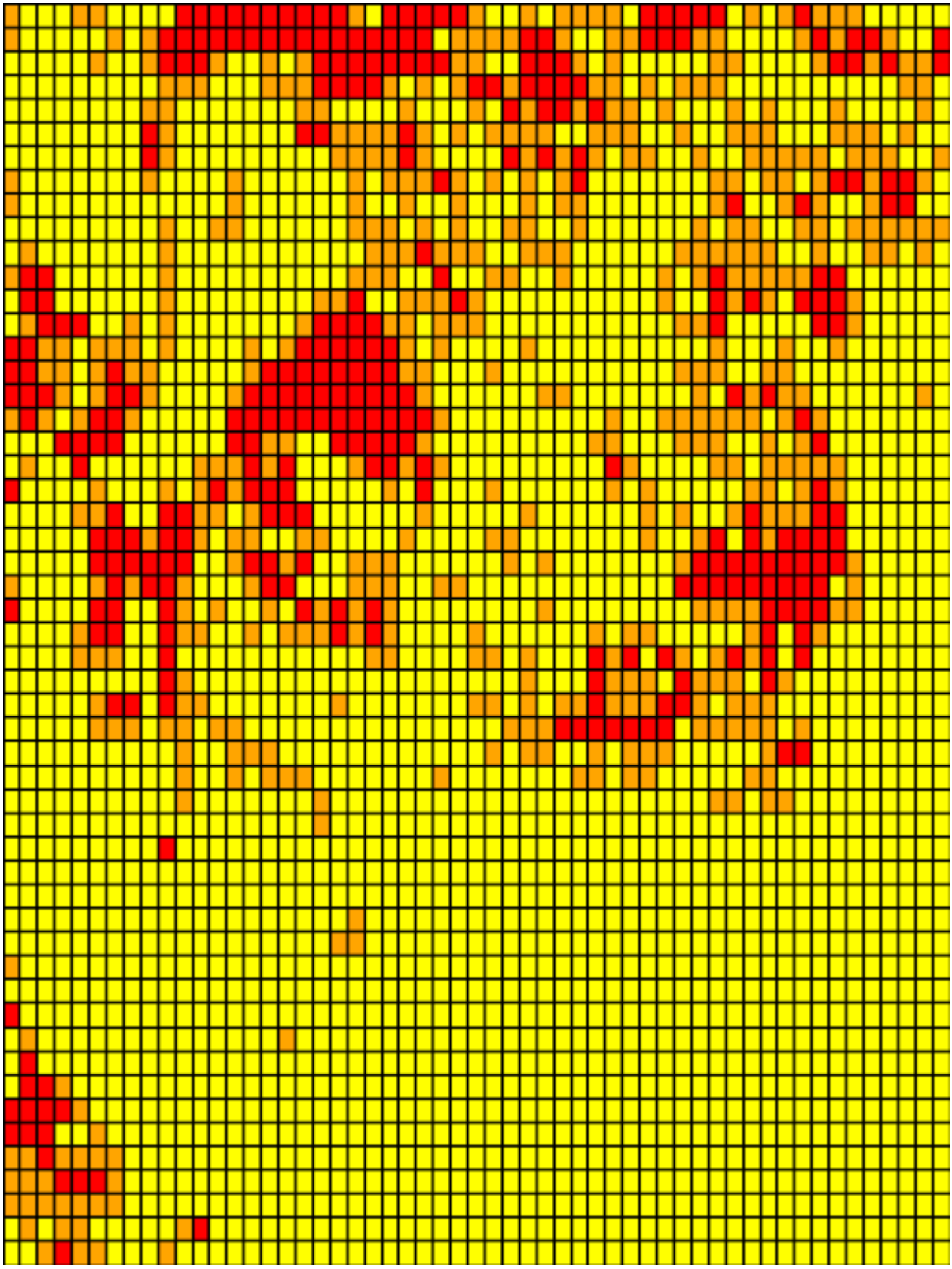


Figure A.4 — Choropleth or graduated map

A.4. Proportional symbol

This use case is discussed in styles-and-symbology#Issue 20.

Existing conformance classes required for this use case

- vector feature styling
- viz/feature.pass (define conformance classes) #
- geometry selection #
- Geometry Manipulation Functions (centroid #)
- Interpolate function #
- Shape Graphics

Example stylesheet:

Below is a CSS encoding to describe a proportional symbol map that represents the number of inhabitants, on a regular grid (SUM_POP property).

```
*{
  stroke: grey;
  stroke-width: 1px;
  [SUM_POP>0] {
    geometry: centroid(the_geom);
    mark:   symbol(circle);
    mark-size: [Interpolate(
         SUM_POP,
         0, 10,
         30, 20,
         84, 100,
         'numeric',
         'linear')];
   :mark {
       fill: orange;
       fill-opacity: 0.2;
       stroke: black;
       stroke-width : 1px;
     }
  }
}
```

Describe any capability not already addressed by existing conformance classes

Example image illustrating the portrayal use case:

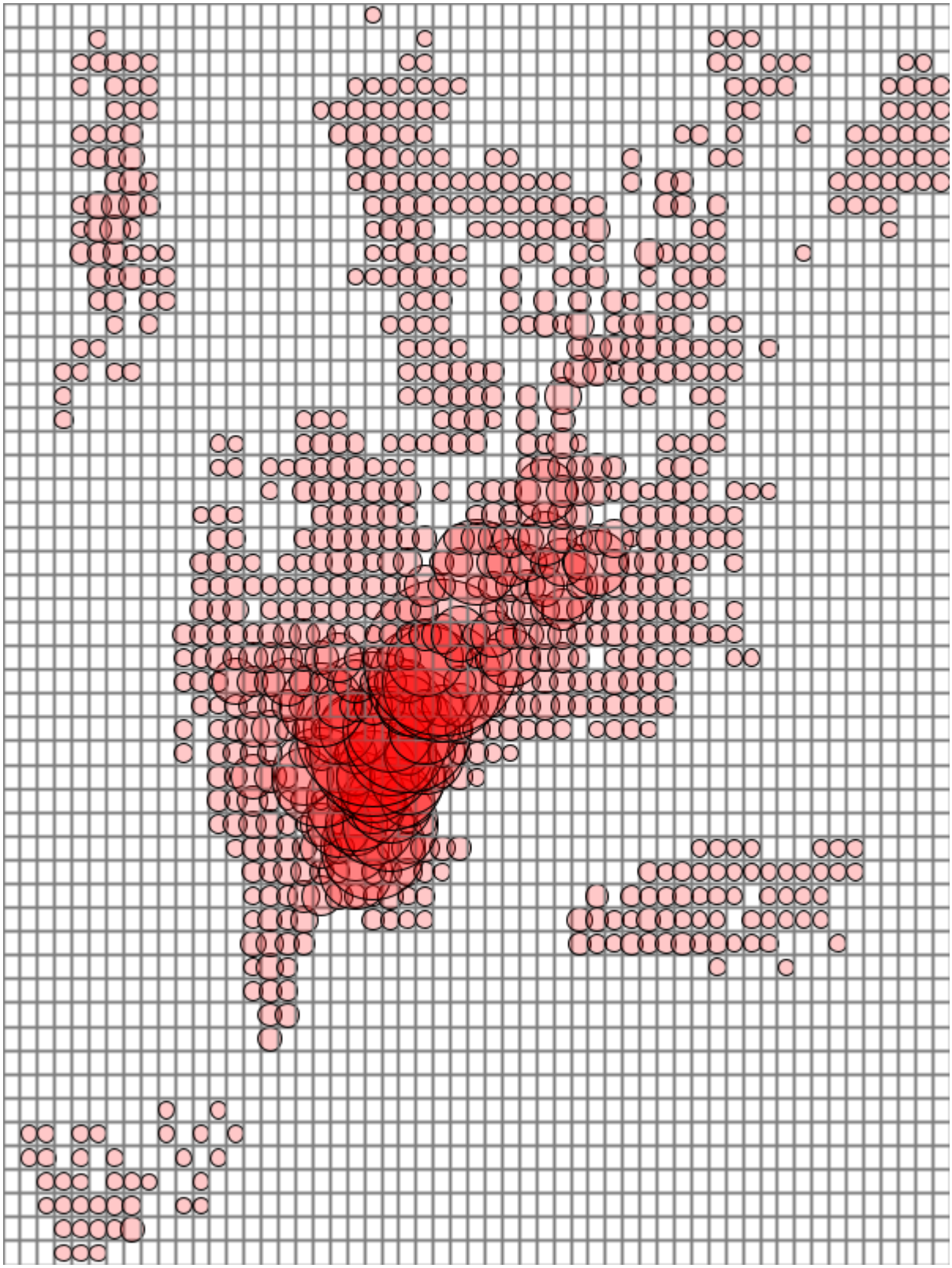


Figure A.5 — Proportional symbol

A.5. Proportional bivariate map

This use case is discussed in styles-and-symbology#Issue 21.

Existing conformance classes required for this use case

- Not specified

Example stylesheet:

Below is a CSS encoding to describe a proportional symbol map with a color for each Local Climate Zones type filtered to represent a climatic hazards.

```
*{
 [LCZ_PRIMARY != 104]{
   stroke: grey;
   stroke-width: 1px;
 }
 [SUM_POP>0] {
    geometry: centroid(the_geom);
    mark:   symbol(circle);
    mark-size: [Interpolate(
         SUM_POP,
         0, 10,
         30, 20,
         84, 100,
         'numeric',
         'linear')];
 };
 /* @title high climate risk */
 [LCZ_PRIMARY = 1],
 [LCZ_PRIMARY = 2],
 [LCZ_PRIMARY = 3],
 [LCZ_PRIMARY = 10],
 [LCZ_PRIMARY = 105]{
   :mark{
     fill: red;
   }
 };

 /* @title moderate climate risk */
 [LCZ_PRIMARY = 4],
 [LCZ_PRIMARY = 5],
 [LCZ_PRIMARY = 6],
 [LCZ_PRIMARY = 7],
 [LCZ_PRIMARY = 8],
 [LCZ_PRIMARY = 9]{
   :mark{
     fill: orange;
   }
 };

 /* @title low climate risk */
 [LCZ_PRIMARY = 101],
 [LCZ_PRIMARY = 102],
 [LCZ_PRIMARY = 103],
 [LCZ_PRIMARY = 106],
```

```
[LCZ_PRIMARY = 107]{
  :mark{
    fill: green;
  }
};
}
```

Describe any capability not already addressed by existing conformance classes

Example image illustrating the portrayal use case:

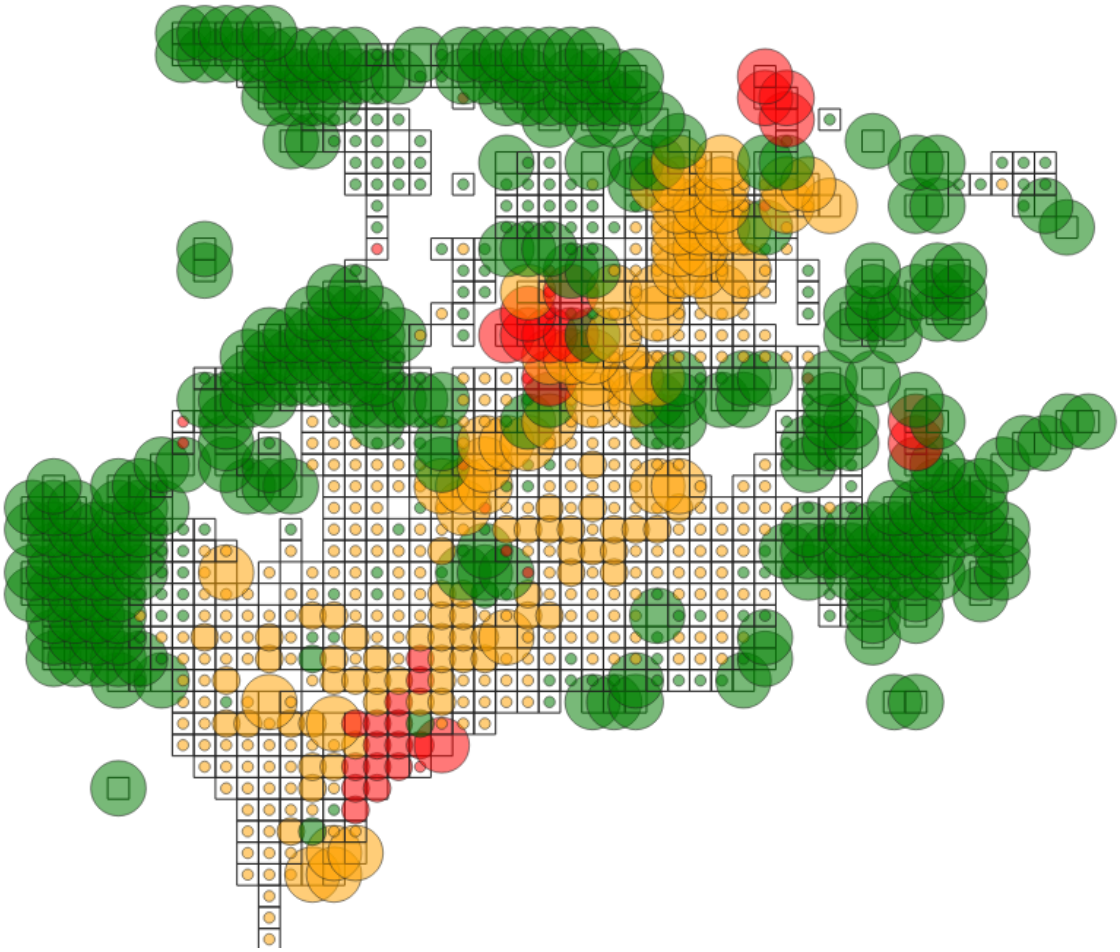


Figure A.6 — Proportional bivariate map

A.6. Dot map density

This use case is discussed in styles-and-symbology#Issue 22.

Existing conformance classes required for this use case

- Dashes, Stipples, Hatches and Gradients
- (conformance class for more specific stippling)

Example stylesheet:

Below is a CSS encoding to describe a dot map. A dot map is used to create a visual impression of density by placing a dot or some other symbol in the approximate location of one or more instances of the variable being mapped. The mapped variable is the area of the LCZ geometry. The dot map can be colored according the LCZ types. e.g. : LCZ equals to 1, 2 , 3 filled in red to represent a high climatic hazards.

Describe any capability not already addressed by existing conformance classes

- DotFill must be defined with the following properties
  - quantityPerMark : the quantity represented by a single dot.
  - totalQuantity : the total quantity to be represented.
  - mode : the algorithm to distribute the mark random, grid...
- a mark or a set of marks

Example image illustrating the portrayal use case:

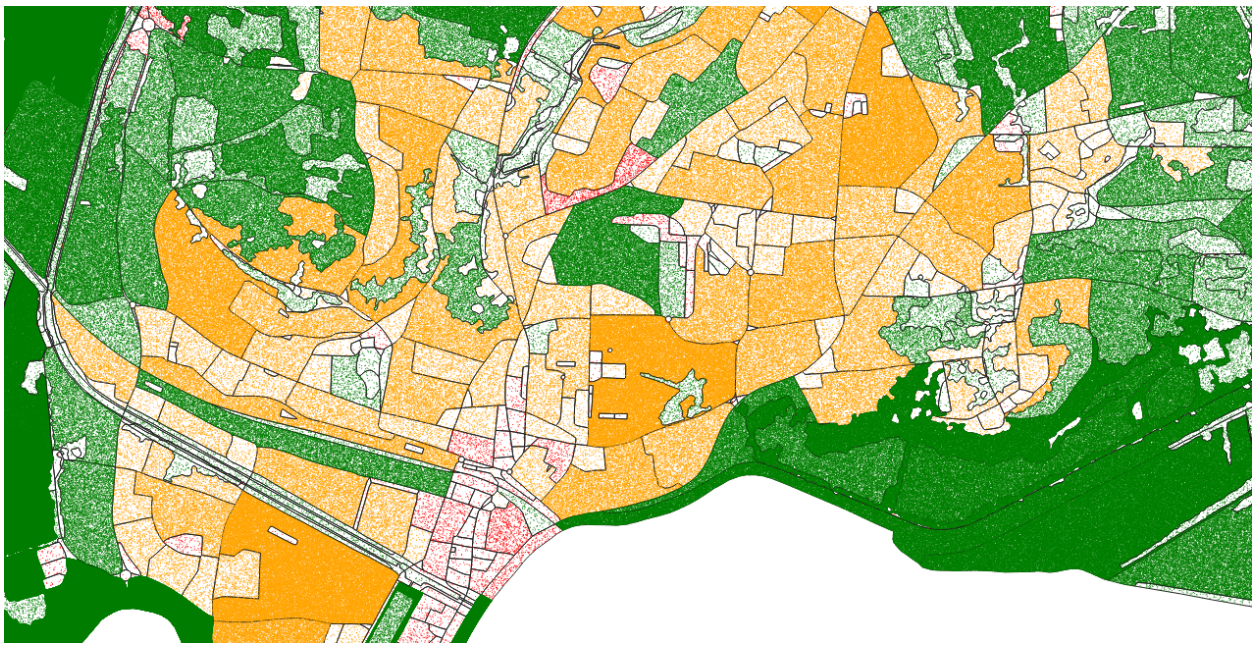


Figure A.7 — Dot map density

A.7. Bivariate proportional symbol

This use case is discussed in styles-and-symbology#Issue 23.

Existing conformance classes required for this use case

Example stylesheet:

Bivariate map is a technique in cartography to display two different thematic variables at the same time. One of the most common techniques is to create a bivariate map is to combine of visual variables. The approach can support different map reading tasks. The following map uses the same visual variable to represent two variables (Half Circle). It permits a cross-variable comparison between the number of permits in 2005 and 2014.

```
*{
  stroke: grey;
  stroke-width: 1px;
  [NB_PERMITS_2005>0] {
    geometry: centroid(the_geom);
    mark:   symbol(semicircle);
    mark-size: [Interpolate(
         NB_PERMITS_2005,
         0, 10,
         659, 100,
         'numeric',
         'linear')];
   :mark {
       fill: rgb(153, 153, 255);
       fill-opacity: 0.2;
       stroke: black;
       stroke-width : 1px;
     }
  };
    [NB_PERMITS_2014>0] {
      geometry: centroid(the_geom);
      mark:   symbol(semicircle);
      mark-size: [Interpolate(
           NB_PERMITS_2014,
           0, 10,
           659, 100,
           'numeric',
           'linear')];
     :mark {
         fill: rgb(102, 0, 204);
         fill-opacity: 0.2;
         stroke: black;
         stroke-width : 1px;
         rotation: 180deg;
       }
    }
}
```

Describe any capability not already addressed by existing conformance classes

Example image illustrating the portrayal use case:

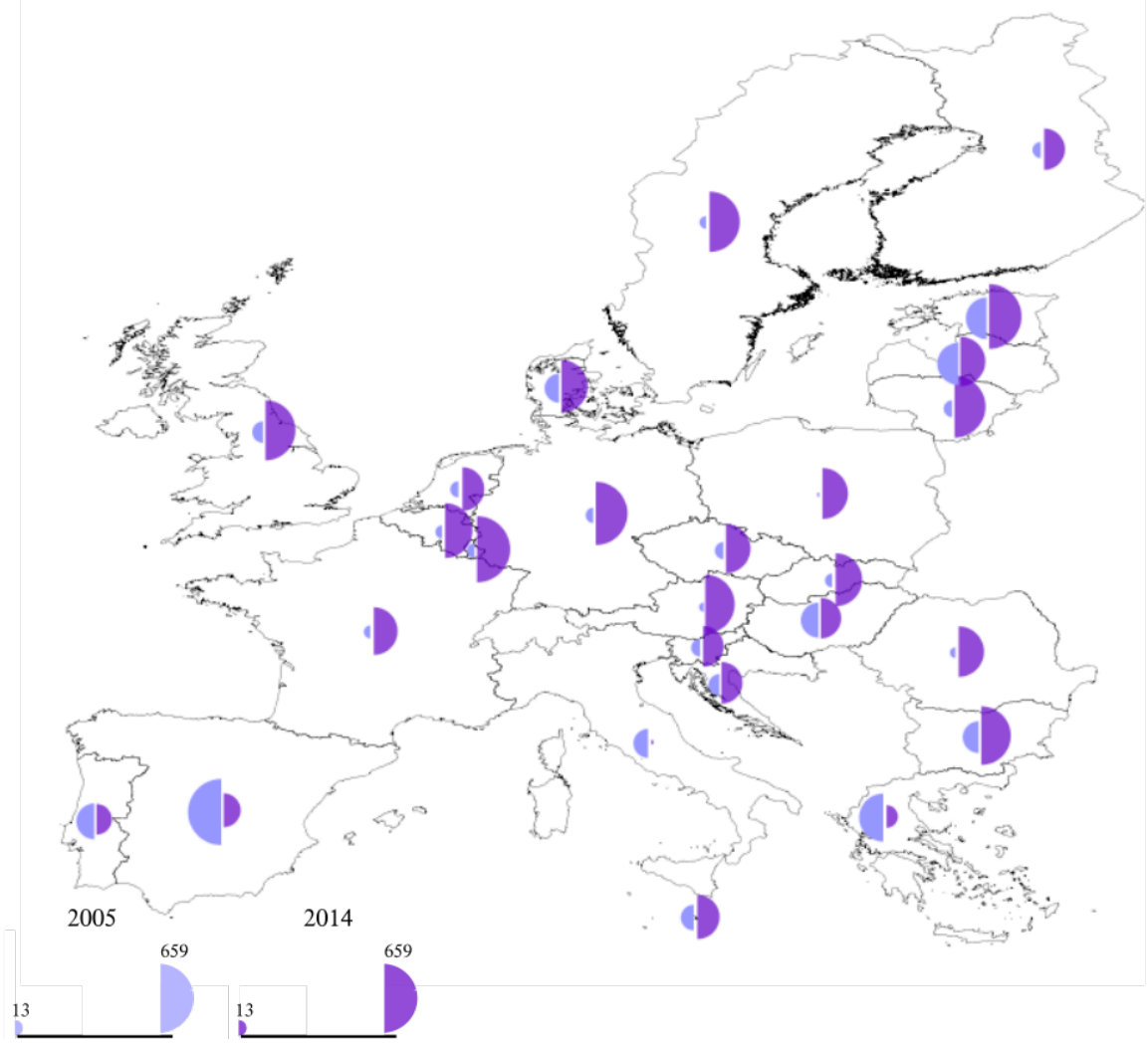


Figure A.8 — Bivariate proportional symbol

An additional example is presented below.

Figure A.9 — Additional example of Bivariate proportional symbol

A.8. Custom fill

This use case is discussed in styles-and-symbology#Issue 24.

Existing conformance classes required for this use case

Example stylesheet: The following map shows assembled visual variables expressed with custom fills : Graphic Fill and Hatched Fill.

Describe any capability not already addressed by existing conformance classes

HatchedFill must be defined with the following properties

- angle : the orientation of the hatches
- distance : the perpendicular distance between two hatches
- offset : the offset of the hatches.
- stroke

Example image illustrating the portrayal use case:

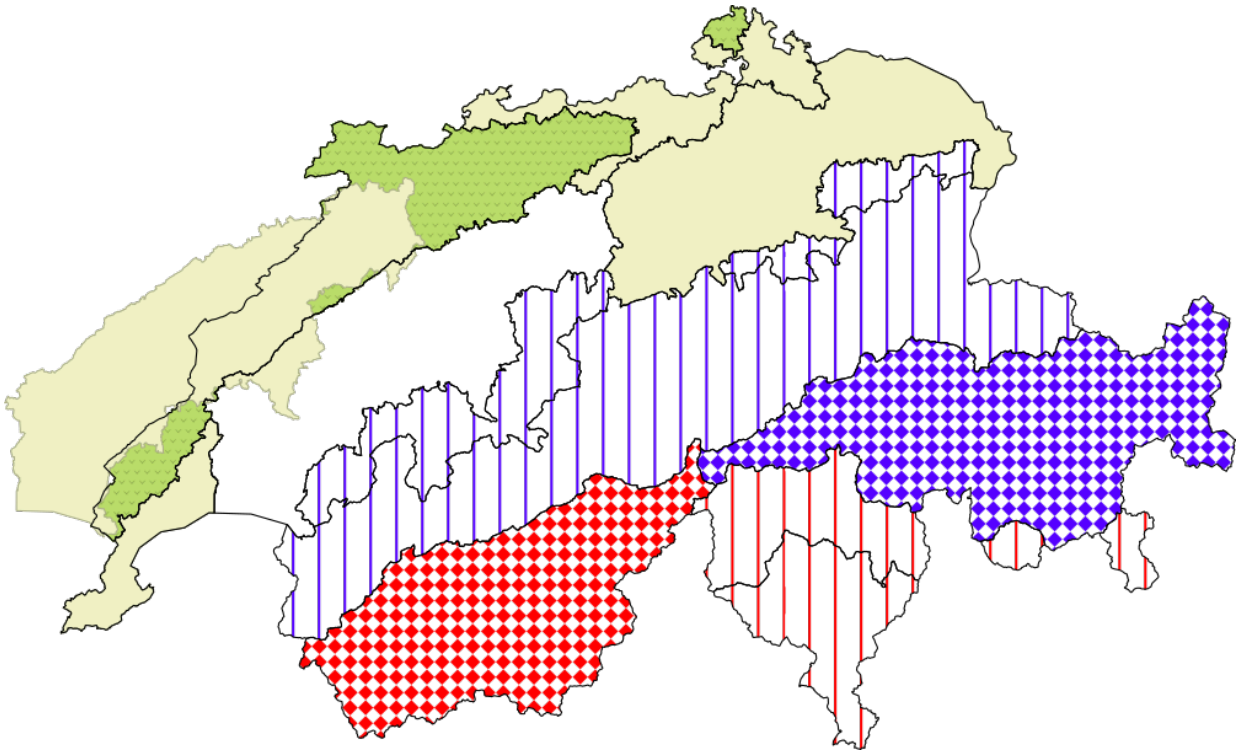


Figure A.10 — Custom fill

An additional example of Custom fill is shown below.

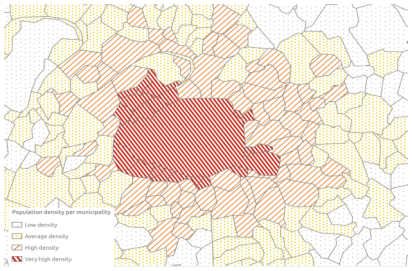


Figure A.11 — Another example of Custom fill

A.9. Proportional colored text

This use case is discussed in styles-and-symbology#Issue 25.

Existing conformance classes required for this use case

Example stylesheet:

- Proportional label map uses the same technique as proportional symbols, provided that the size of the font is scaled proportionately.
- Here the Annex describes a CSS file that represents the Local Climate Zone types with a text and a color according a climatic hazards.

```
*{
     stroke: grey;
     stroke-width: 0.26;
     label: [LCZ_PRIMARY];
     font-family: Arial;
     font-size: [Interpolate(
                          area(geometry)/10000,
                          0, 10,
                          5, 20,
                          10, 32,
                          'numeric',
                          'linear')];
      font-style: normal;
      font-weight: bold;
      font-fill: black;
      label-anchor: 0.5 0.5;
      label-auto-wrap: 60;
      label-max-displacement: 150;

       /* @title high  */
          [LCZ_PRIMARY <= 4],
          [LCZ_PRIMARY = 8],
          [LCZ_PRIMARY = 10],
          [LCZ_PRIMARY = 105] {
              font-fill: red;
          };
          /* @title mid */
          [LCZ_PRIMARY >= 5]
```

```
            [LCZ_PRIMARY <= 7],
            [LCZ_PRIMARY = 9],
            [LCZ_PRIMARY = 10] {
             font-fill :orange;
            };
            /* @title low */
            [LCZ_PRIMARY >= 101]
            [LCZ_PRIMARY <= 104],
            [LCZ_PRIMARY = 107]{
              font-fill:  green;
            }
}
```

Describe any capability not already addressed by existing conformance classes

Example image illustrating the portrayal use case:

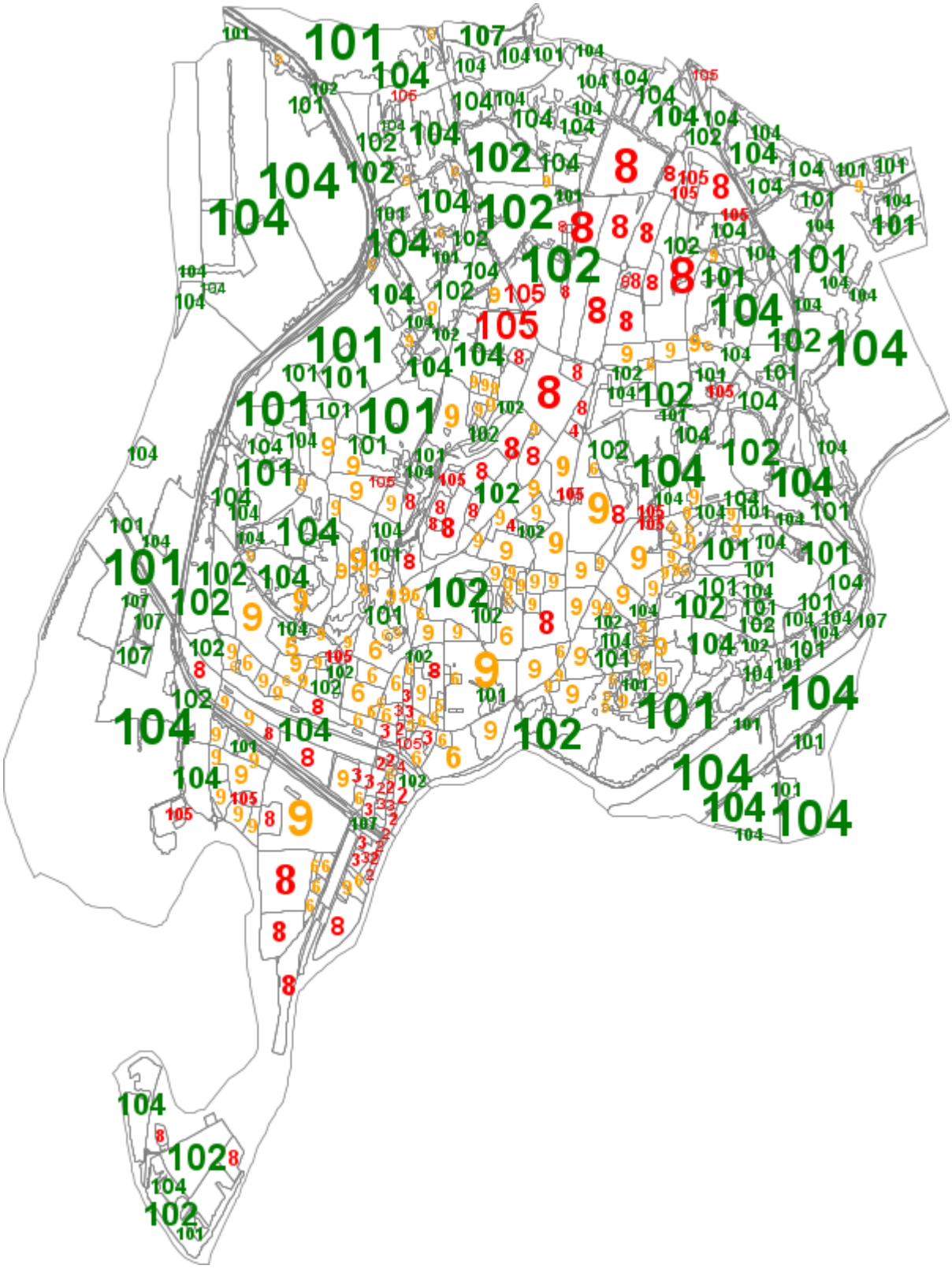


Figure A.12 — Proportional colored text

B

ANNEX B (INFORMATIVE) REVISION HISTORY

B ANNEX B (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2022-12-19	0.1	G. Hobona	all	initial version

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Gobe Hobona: OGC 20-091, *OGC API – Common and OGC API – Features Sprint 2020: Summary Engineering Report*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-091.html.

[2] Andreas Matheus: OGC 21-068, *OGC Best Practice for using SensorThings API with Citizen Science*. Open Geospatial Consortium (2022). https://docs.ogc.org/bp/21-068.pdf.

[3] Aleksandar Balaban, Andreas Matheus: OGC 21-020r1, *OGC Testbed-17: Data Centric Security ER*. Open Geospatial Consortium (2022). https://docs.ogc.org/per/21-020r1.html.