DRAFT

OGC Vector Tiles Pilot 2

Vector Tiles Filtering Language Engineering Report

Table of Contents

1. Subject . . . 4
2. Executive Summary . . . 5
   2.1. Document contributor contact points . . . 6
   2.2. Foreword . . . 6
3. References . . . 7
4. Terms and definitions . . . 8
   4.1. Abbreviated terms . . . 8
5. Overview . . . 9
6. Introduction . . . 10
7. Previous work . . . 11
   7.1. Filter Encoding . . . 11
   7.2. Catalog Services . . . 11
   7.3. CQL adoption and extension in open source software . . . 13
   7.4. OGC Testbed 14 . . . 13
   7.5. OGC Testbed 15 . . . 14
   7.6. STAC and OGC API Sprint, OGC API Filtering extension . . . 15
8. Filter Conceptual Model and VTP2 testing considerations . . . 20
   8.1. Conceptual model . . . 20
   8.2. The CQL language . . . 21
9. Filtering Tiles . . . 23
   9.1. Vector Tiles Pilot 2 testing operator subset . . . 23
   9.2. Queryables . . . 23
   9.3. Single and multi-layer tile filtering . . . 23
10. Filtering GeoPackages with Vector Tiles . . . 25
   10.1. Attributes in Attributes Tables . . . 25
   10.2. Spatial Filtering . . . 25
11. Implementations . . . 30
   11.1. GeoSolutions D100 OGC API - Features . . . 30
   11.2. Terranodo D100 OGC API - Tiles . . . 33
      11.2.1. CQL . . . 33
      11.2.2. Sample Requests . . . 33
      11.2.3. Queryables . . . 35
   11.3. interactive instruments D101 Features, Tiles and Styles API . . . 37
      11.3.1. The starting point . . . 37
      11.3.2. Support for filters . . . 41
   11.4. GeoSolutions D102 OGC API - Tiles . . . 52
   11.5. Ecere D103 Features & OGC API - Tiles . . . 55
      11.5.1. Examples of filtering expressions used with the OGC API - Features . . . 55

11.5.2. Use of filtering for vector tiles . . . 60
11.5.3. Multi-layer and scale-based filtering (OGC API - Tiles) . . . 62
11.5.4. POST requests to the OGC API - Tiles . . . 67
11.5.5. Filtering driven by styles . . . 68
11.5.6. Retrieving attributes separately from geometry . . . 68
11.5.7. CMSS Expressions Syntax . . . 68
11.6. Ecere D105 Client . . . 70
11.7. Ecere D107 GeoPackage producer & client . . . 71
11.8. Skymantics D104 Client . . . 72
11.8.1. Use of CQL Unsuccessful . . . 72
11.8.2. Filtering Without CQL . . . 73
11.8.3. Future Work . . . 75
11.9. GeoSolutions D104 Client . . . 76
11.9.1. Attribute filter . . . 77
11.9.2. Spatial filter . . . 78
11.9.3. Temporal filter . . . 79
11.9.4. Mixed filters . . . 80
12. Results and findings . . . 82
12.1. Issues Encountered . . . 82
12.1.1. CQL language issues . . . 82
12.1.2. Quoted identifiers . . . 82
12.1.3. ENVELOPE constructor . . . 82
12.1.4. EXISTS operator . . . 82
12.2. Findings . . . 82
12.2.1. Client versus Server-side filtering . . . 82
12.2.2. Client-side control of contents and default filtering . . . 83
12.2.3. Filter capabilities and desired minimum filtering operator set . . . 83
12.3. Recommendations . . . 84
12.3.1. CQL CRS geometry support . . . 84
12.3.2. Filter capabilities support . . . 84
12.3.3. Support for complex filtering . . . 84
12.4. Future work . . . 85
12.4.1. Selection of returned attributes . . . 85
12.4.2. JSON based filtering languages . . . 85
12.4.3. Explore multi-layer tile filtering and querying support . . . 85
Appendix A: CQL BNF . . . 86
Appendix B: Queryables . . . 94
B.1. Requirement Class "Queryables" . . . 94
B.1.1. Fetch the queryable properties of the features in a collection . . . 94
Appendix C: Filter capabilities . . . 99
Appendix D: Revision History . . . 102

Appendix E: Bibliography 104

Publication Date: YYYY-MM-DD

Approval Date: YYYY-MM-DD

Submission Date: YYYY-MM-DD

Reference number of this document: OGC 19-084

Reference URL for this document: http://www.opengis.net/doc/PER/vtp2-D002

Category: OGC Public Engineering Report

Editor: Andrea Aime

Title: OGC Vector Tiles Pilot 2: Vector Tiles Filtering Language Engineering Report

---

OGC Public Engineering Report

COPYRIGHT

WARNING

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 Public 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

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to

indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Chapter 1. Subject

The OGC Vector Tiles Pilot 2: Vector Tiles Filtering Language Engineering Report (ER) defines a filter language for vector data delivered as tiles (also known as vector tiles). The language applies to vector tiles served through implementations of the OGC API – Features standard and the draft OGC API - Tiles specification, but also describes filtering support for GeoPackages that provide contents structured as vector tiles.

The ER further includes an assessment of filter languages, styles and online/offline symbol sharing for GeoPackages, OGC API - Features and OGC API - Tiles implementations for accuracy and completeness in applications that render vector tiles at local to regional scales.

Chapter 2. Executive Summary

Consistent application of filters on data is important, whether an application is connected to a network or not. This is more so in environments with Denied, Degraded, Intermittent or Limited (DDIL) connectivity. This engineering report presents work conducted by the OGC Vector Tiles Pilot Phase 2 (VTP2) project with respect to filter languages.

VTP2 sought to deliver a consistent, interoperable online/offline architecture consisting of feature and tile servers, and GeoPackage-producing components that could publish, display and query vector tiles. One of the objectives of the VTP2 pilot was to develop a filtering language for vector tiles, and implement and exercise the filtering language on clients and servers.

The purpose of this Engineering Report and associated component implementations is to study the filtering and production of tiled feature data (also known as vector tiles) and GeoPackages, with particular attention to the following filtering features based on spatial and temporal aspects, as well as alphanumeric

The APIs applied in the pilot are from the emerging suite of OGC API standards. The first of the OGC API standards to be approved by the Technical Committee is the OGC API - Features standard. The pilot also explored the draft OGC API - Tiles and OGC API - Styles specifications.

The report also focuses on the opportunity of client-side versus server-side filtering, coupled with the possibility of caching, overall performance and user experience.

Feature and tile servers deployed in VTP2 implemented the OGC API - Features standard and the draft OGC API - Tiles and OGC API - Styles specifications. Whereas the feature and tile servers allowed the pilot to explore online support for vector tiles, GeoPackage allowed the pilot to explore offline support.

The key findings of the pilot, with regard to a vector tiles filter language, are:

- Both server-side and client-side filtering are important. Server-side filtering reduces payload size and can simplify the client side development. The client-side helps to keep tiles static, and thus cachable, while adding interactivity.
- In the context of vector tiles some pre-defined server side filters are important to include pertinent information depending on the zoom level, and to keep the vector tile size in control.
- The filtering language can contain operators that are not needed for specific applications, and difficulties of implementing a text based parsers may limit its implementations.

The pilot made the following recommendations, with regard to a vector tiles filter language:

- A way to advertise support for a subset of filtering operators and expressions should be developed and included in filtering extensions.
- JSON based filtering languages should be explored to lower the implementation cost and provide a suitable option for clients offering Graphical User Interfaces (GUIs) to build filters.
- Complex filtering and multi-layer filtering should be investigated, and methods to support them implemented.

- While filtering can reduce the number of returned items, it's also necessary to develop support for limiting the returned attributes. A querying extension should both advertise which attributes can be removed, and provide ways to limit them in requests.

2.1. Document contributor contact points

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

Contacts

Name	Organization	Role
Andrea Aime	GeoSolutions	Editor
Stefano Bovio	GeoSolutions	Contributor
Clemens Portele	interactive instruments	Contributor
Jeff Yutzler	Image Matters	Contributor
Jerome Jacovella-St-Louis	Ecere	Contributor
Jeffrey Johnson	Terranodo	Contributor
Sergio Taleisnik	Skymantics	Contributor

2.2. Foreword

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.

Chapter 3. References

The following normative documents are referenced in this document.

- OGC: OGC 17-069r3, OGC® API - Features - Part 1: Core [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html]
- OGC: OGC 09-026r2, OGC® Filter Encoding 2.0 Encoding Standard – With Corrigendum [http://docs.opengeospatial.org/is/09-026r2/09-026r2.html]
- OGC: OGC 12-168r6, OGC® Catalogue Services 3.0 - General Model [http://docs.opengeospatial.org/is/12-168r6/12-168r6.html]
- OGC: OGC 12-176r7, OGC® Catalogue Services 3.0 Specification - HTTP Protocol Binding [https://docs.opengeospatial.org/is/12-176r7/12-176r7.html]
- OGC: OGC 05-078r4, OGC® Styled Layer Descriptor profile of the Web Map Service Implementation Specification [http://portal.opengeospatial.org/files/?artifact_id=22364]
- OGC: OGC 07-057r7, OpenGIS® Web Map Tile Service Implementation Standard [http://portal.opengeospatial.org/files/?artifact_id=35326]
- OGC: OGC 17-083r2, OGC Two® Dimensional Tile Matrix Set [http://docs.opengeospatial.org/is/17-083r2/17-083r2.html]
- OGC: OGC 19-014r1, Core Tiling Conceptual and Logical Models for 2D Euclidean Space, Draft [https://portal.opengeospatial.org/files/91763]

Chapter 4. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- **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). (from OGC 19-014r1).

  NOTE In the ER, the definition for “tile is further constrained as a tessellated representation of geographicdata, often part of a set of such elements, covering a spatially contiguous extent which can be uniquely defined by a pair of indices for the column and row along with an identifier for the tile matrix (adapted from OGC 07-057r7 [http://portal.opengeospatial.org/files/?artifact_id=35326])

- **GeoPackage**

  an open, standards-based, platform-independent, portable, self-describing, compact format for transferring geospatial information [geopackage [http://www.geopackage.org/]]

- **dataset**

  collection of data, published or curated by a single agent, and available for access or download in one or more formats [OGC 17-069r3 [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html]]

- **collection**

  a set of features from a dataset [OGC 17-069r3 [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html]]

4.1. Abbreviated terms

- CQL Common Query Language
- FE Filter Encoding
- GPKG GeoPackage
- MVT Mapbox Vector Tiles
- OGC Open Geospatial Consortium
- VT Vector Tiles
- WFS Web Feature Service
- WKT Well Known Text
- WMS Web Map Service

Chapter 5. Overview

The introduction provides a quick introduction to the VPT2 objectives.

The previous work section provides a summary of previous work related to tiling and filtering.

The Filter Conceptual Model and VTP2 testing considerations provides an overview of the filtering expressions and operators, a more in depth analysis of CQL.

The [Tiles Filtering] section provides a list of filters used in the VTP2 tests and implementations, and the protocol extensions required to advertise queryables and submit filters.

The GeoPackaage filtering section details application of filters against GeoPackages, and support structures that can be used to speed up tile contents filtering.

The implementations section covers each VPT2 server and client deliverable involved in filtering tiles. Each component is described in detail, along with implementation difficulties and unique features.

The results and findings section summarizes the issues found in the development of VPT2 components, findings, as well as future work items.

Annex A contains the full Baukus Naur Form (BNF) of the CQL text encoding.

Annex B contains a copy of they Queryables specification from the Testbed 15 Styles Engineering report [http://docs.opengeospatial.org/per/19-010r2.html#get_queryables]..

FilterCapabilities provides a sample filter capabilities document developed during the STAC and OGC API - Features and Catalogues Sprint [https://www.opengeospatial.org/projects/initiatives/ogcapi-featu].

Chapter 6. Introduction

The goal of the Vector Tiles Pilot Phase 2 filtering activity, as described in this Engineering Report (ER) and associated component implementations, was to study the filtering and production of tiled feature data (also known as vector tiles) and GeoPackages, with particular attention to the following:

- Filtering tiles based on the tile matrix set, range of zoom levels, bounding box and tile matrix set limits per level
- Removing layers and attributes that are not desired client side
- Filtering features based on spatial, temporal, as well as alphanumeric aspects

The ER also focuses on the opportunity for client-side versus server-side filtering, coupled with the possibility of caching, thereby enhancing overall performance and user experience.

Chapter 7. Previous work

This section discusses the various filtering languages and their usefulness in the context of filtering vector tiles.

7.1. Filter Encoding

The OpenGIS Filter Encoding 2.0 Encoding Standard [http://docs.opengeospatial.org/is/09-026r2/09-026r2.html] (FE) describes an XML and KVP encoding of a system neutral syntax for expressing projections, selection and sorting clauses collectively called a query expression.

The FE Standard includes both a conceptual filtering model as well as one encoding in Extensible Markup Language (XML). The FE language supports:

- Literals and value references (attributes)
- Expressions, as basic math, and functions
- Comparison operators for alphanumeric expressions
- Spatial operators for location-based expressions
- Temporal operators for time-based expressions
- Logical operators allowing the combination of all the above
- Value references can use a a subset of XPath [http://docs.opengeospatial.org/is/09-026r2/09-026r2.html#111], allowing sophisticated access to objects in a nested structure, making it a natural fit for application schemas based on complex features.

The FE conceptual model is distributed across the entire specification document. However, a simple overview diagram of it is included in the conceptual model chapter in this ER.

The XML encoding is best suited for usage in XML sent in HTTP POST requests. Usage in GET requests, as a query parameter, is also possible, but limited in applicability by the verboseness of the language, the URL encoding obfuscating most of its structure, and the practical length limits of a URL.

7.2. Catalog Services

OGC Catalog Services 3.0 [http://docs.opengeospatial.org/is/12-168r6/12-168r6.html] (CSW) supports multiple query languages by means of a `CONSTRAINTLANGUAGE` parameter in `GetRecords` requests.

The CSW Standard supports the usage of OGC Filter Encoding, but also defines the Common Query Language (CQL). CQL is a text based query language with "a syntax similar to the SQL Where Clause", extended to support both temporal and spatial filters.

The HTTP binding of the Catalog Services standard, also referred to as the Catalogue Services for the Web (CSW) standard (OGC 12-176r7), advertises CQL as the primary means to filter records. Below is an example used against the `AnyText` property, which in CSW allows for full text searches:

```
http://host/catalogServer/csw?
service=CSW&
version=2.0.2&
request=GetRecords&
resultType=results&
outputSchema=http%3A%2F%2Fwww.isotc211.org%2F2005%2Fgmd&
NAMESPACE=xmlns(gmd=http%3A%2F%2Fwww.isotc211.org%2F2005%2Fgmd)&
typeNames=gmd:MD_Metadata&
CONSTRAINTLANGUAGE=CQL_TEXT
CONSTRAINT=AnyText%20=%20'air%20temperature'&
CONSTRAINT_LANGUAGE_VERSION=1.1.0&
elementSetName=full
```

However, the CQL as specified in CSW has several significant limitations:

- The Baukus Naur Form (BNF) is not structurally valid
- The CQL imposes limitations on comparisons that are not present in OGC Filter, e.g., while `attribute = literal` is valid, `attribute1 > attribute2` is not
- The standard lacks CQL examples. The same is true of most CSW implementations. On the other hand, OGC Filter examples can be found in abundance, suggesting a less than widespread implementation of CQL.
- While geometries can be expressed in Well-Known Text (WKT), there is no support for defining their Coordinate Reference System CRS.

The implementation of OGC Catalog Services also have pioneered two concepts that are now relevant with OGC API - Features services, queryables and returnables.

Queryable properties [http://docs.opengeospatial.org/is/12-168r6/12-168r6.html#19] are "field-like objects" that can be used to build filters, and "may differ from the response data elements". In other words, queryables are property names, they do not necessarily match the names of the returned properties, and may not appear in the data schema at all (if any schema is present). One example of a queryable without a direct match with returned data is `AnyText`, which is defined as a "target for full-text search of character data types in a catalogue". Another interesting example is "Abstract". This a property that can be found among the results, but whose name may vary depending on the record format. For example, in Dublin Core records "Abstract" is named "description", while in ISO records "Abstract" matches the nested property `/gmd:MD_Metadata/gmd:identificationInfo/gmd:MD_DataI dentification/gmd:abstract`.

On the other hand Returnable properties [http://docs.opengeospatial.org/is/12-168r6/12-168r6.html#20] are property names found in the response. Some of them have a match with a corresponding queryable (e.g., `description`), while others do not. If this is the case, those properties cannot be used in filters (e.g, `publisher`, `contributor`).

7.3. CQL adoption and extension in open source software

The GeoServer [https://www.geoserver.org] open source project was the first reference implementation of the OGC Web Feature Service [https://www.opengeospatial.org/standards/wfs] (WFS) standard.

Users of the WFS standard reported difficulties in using the OGC Filter Encoding standard for filtering. This was partly due to the complexity of the language, and partly due to limits in its usage within HTTP GET requests. This limited the ability to share data references as simple links.

The GeoServer project adopted, in 2007, a `CQL_FILTER` query parameter for both WFS GetFeature and Web Map Service (WMS) GetMap:

- In WFS GetFeature, the parameter serves as an alternative to `FILTER`, which uses a different OGC fe XML encoding according to the WFS version in use
- In WMS GetMap, the parameter filters the map contents, allowing dynamic selection and filtering, from the client, without requiring to setup again the whole map style (see also the `SLD` and `SLD_BODY` parameters in OGC 05-078r4 [http://portal.opengeospatial.org/files/?artifact_id=22364]).

Eventually the limitations of first CQL version became apparent, so in 2010 Extended CQL [http://old.geotools.org/ECQL-Parser-Design_110493908.html] (ECQL) was created by the GeoTools and GeoServer communities, to bring the language closer to the OGC Filter capabilities. In particular:

- Support free form comparison expressions in terms of `expression OP expression`
- Support for FeatureId filters

The `CQL_FILTER` parameter has supported both CQL and ECQL since then.

To date one significant limitation of (E)CQL usage, is that it can only be used in a GET request, and has no support in XML POST requests.

7.4. OGC Testbed 14

Participants in the Testbed 14 [https://www.opengeospatial.org/projects/initiatives/testbed14] Complex Features Handling thread performed an extensive analysis of filtering languages [1]. Of particular interest is the ER chapter on API building blocks for queries [https://docs.opengeospatial.org/per/18-021.html#_api_building_blocks_for_queries]

That chapter contains an analysis of filtering languages following different approaches:

- Languages already available OGC standards, namely, OGC Filter Encoding and CQL.
- Emerging languages used by the Web Community, in particular, GraphQL [https://graphql.org/] and Falcor [https://netflix.github.io/falcor/].
- Creating a new language that is simple to implement (more about this in the STAC and OGC API sprint report below).

Notable observations from that work include:

- Due to being encoded in XML, OGC Filter Encoding is not considered a natural fit for JSON-first services (though it remains a good option for implementations returning GML based application schemas)
- Falcor is best suited for JSON only services, and needs to be extended to support spatial support capabilities.
- GraphQL needs extensions to its spatial capabilities. GraphQL also depends on the presence of a schema, which is not mandatory in OGC API - Features. Like CQL, GraphQL does not support multi-valued properties.

CQL has been considered suitable for usage in future extensions of the OGC API - Features but, due to its numerous limitations, but not without further work.

For CQL, the notion of a **queryable** was first used in combination with a data retrieving service. As with CSW, a queryable is a property that can be queried, but it is not necessarily tied to the schema of the dataset. A queryable could indeed be a visible property of the data. A queryable could also be referring to a nested element via a simple name, or referring to a summary property, such as `AnyText`, without the property actually being present in the result. The concept was further explored in OGC Testbed 15, where a dedicated OGC API - Features extension was proposed.

7.5. OGC Testbed 15

Participants in the OGC Testbed 15 [https://www.opengeospatial.org/projects/initiatives/testbed15] Open Portrayal thread investigated and prototyped a new Application Programming Interface API for styles publishing, discovery and editing. Results of this activity are documented in the OGC Testbed-15: Styles API Engineering Report [https://docs.opengeospatial.org/DRAFTS/19-010.html#rc_queryables].

When a client works on styles, especially editing, knowing which attributes can be used for collection filtering is important. This was handled by the queryables [https://docs.opengeospatial.org/DRAFTS/19-010.html#rc_queryables] OGC API - Features extension draft. Queryables are normally a subset of all possible attributes, and in general, may not even show up among the results of a "items" query. The returned attributes are referred to as "presentables" instead.

Quoting the Testbed 15 ER, each collection exposes a list of Queryable objects, each characterized by:

- `id` (required) - the property name for use in expressions.
- `type` (required) - the data type of the property, one of `string`, `uri`, `enum`, `number`, `integer`, `date`, `dateTime`, `boolean`
- `description` (optional) - a description of the property.
- `required` (optional) - indicator whether the property is always present in features.
- `mediaTypes` (optional) - in general, the representation of the queryables is meant to be independent of the feature encoding. However, this is not always the case. For example, length restrictions or namespace prefixes may result in different property identifiers for the same property. To support this, the definition of a queryable may be restricted to one or more feature encodings (media types).
- `pattern` (optional, only for "string" and "uri") - a regular expression to validate the values of the

property.

- `values` (required, only for "enum") - an array of valid values of the property.
- `range` (optional, only for "number", "integer", "date" and "dateTime") - the range of valid values expressed as an array with two items. Open ranges can be expressed using null for the minimum or maximum value.

As they provide a list of property names the filter can use, queryables are a natural fit for a filtering extension.

7.6. STAC and OGC API Sprint, OGC API Filtering extension

In November 2019 OGC organized the STAC and OGC API - Features and Catalogues Sprint [https://www.opengeospatial.org/projects/initiatives/ogcapi-featu]. The sprint hosted numerous groups, one of them focused on experimenting a possible "OGC API - Features - Part 3: Common Query Language" extension.

The CQL language was considered first in conceptual terms, defining the capabilities of the language and removing its limitations. Then, three possible encodings were prototyped:

- A text-based language, derived from the original CQL and similar to the GeoServer Extended CQL.
- A JSON-based hierarchical objects language, meant to be a port to JSON of the OGC Filter Encoding XML encoding.
- A JSON-based nested array language, that would read like a prefix notation, reminiscent of Mapbox Styles Expressions [https://docs.mapbox.com/mapbox-gl-js/style-spec/expressions/].

The following table compares the various approaches:

Table 1. Comparison of CQL encodings friendliness

	Text	**Hierarchical JSON**	**Array based JSON**
User	Trivial to write by hand and understand	Needs support, easier for those familiar with OGC filter	Reads "backwards", easier for those familiar with Mapbox Styles Expressions
Developer	Requires writing a text based parser based on the BNF	JSON reads directly into a parse-tree like structure (based on nested objects)	JSON reads directly into a parse-tree like structure (based on nested arrays)

The following shows an example of filtering based on properties of a satellite image, and spatial location, in the three encodings:

Table 2. Encoding examples

Language	Example
Text	`beamMode='ScanSAR Narrow' AND swathDirection='ascending' AND polarization='HH+VV+HV+VH' AND intersects(geometry,POLYGON((-77.117938 38.936860,-77.040604 39.995648,-76.910536 38.892912,-77.039359 38.791753,-77.047906 38.841462,-77.034183 38.840655,-77.033142 38.857490)))`

Language	Example
Hierarchical json	

```
{
  "and": [
    {
      "eq": {
        "property": "beamMode",
        "value": "ScanSAR Narrow"
      }
    },
    {
      "eq": {
        "property": "swathDirection",
        "value": "ascending"
      }
    },
    {
      "eq": {
        "property": "polarization",
        "value": "HH+VV+HV+VH"
      }
    },
    {
      "intersects": {
        "property": "footprint",
        "value": {
          "type": "Polygon",
          "coordinates": [
            [
              [ -77.117938, 38.93686 ],
              [ -77.040604, 39.995648 ],
              [ -76.910536, 38.892912 ],
              [ -77.039359, 38.791753 ],
              [ -77.047906, 38.841462 ],
              [ -77.034183, 38.840655 ],
              [ -77.033142, 38.85749 ],
              [ -77.117938, 38.93686 ]
            ]
          ]
        }
      }
    }
  ]
}
```

Language	Example
Array based JSON
```
[
  "all",
  [
    "==",
    [ "get", "beamMode" ],
    "ScanSAR Narrow"
  ],
  [
    "==",
    [ "get", "swathDirection" ],
    "ascending"
  ],
  [
    "==",
    [ "get", "polarization" ],
    "HH+VV+HV+VH"
  ],
  [
    "intersects",
    [ "geometry" ],
    {
      "type": "Polygon",
      "coordinates": [
        [
          [ -77.117938, 38.93686 ],
          [ -77.040604, 39.995648 ],
          [ -76.910536, 38.892912 ],
          [ -77.039359, 38.791753 ],
          [ -77.047906, 38.841462 ],
          [ -77.034183, 38.840655 ],
          [ -77.033142, 38.85749 ],
          [ -77.117938, 38.93686 ]
        ]
      ]
    }
  ]
]
```
 |

In addition to work the filtering language, the sprint considered the cost of filter language implementation, from the filtering operators point of view. The CQL model contains a rich set of comparison, spatial, temporal and logic operators. Implementing the full set can be both expensive, and depending on the application, perhaps un-necessary.

Based on the OGC Filter Encoding 2.0 specification, and the OGC APIs, two options have were considered:

- Conformance classes for filters, identifying them in groups.

- Fine grained filter capabilities documents, listing the specific subset chosen by a server

The **conformance classes** approach allows a compact representation of the supported filters. This is at the expense of the ability to fine grain tune the supported operator set. A natural set of classes would be the existing operator subdivision, logical, comparison, spatial, and temporal. However, in practice subsets of operators tend to go across classes. For example, consider about a minimal subset covering the basic filtering needs, one might consider including all comparisons and logical operators, along with `bbox` and `during` basic spatial and temporal filtering. The uncommon and advanced operators are normally found within the spatial and temporal categories, but the common one are found in all categories.

On the other hand the **capabilities document** allows for a tailored operator selection but at the expense of listing all supported operators. Annex C contains an example of a full queryables document, as prototyped in GeoServer during the code sprint.

Combining the best aspects of the two approaches above is also possible: Declaring a conformance class when all operators in it are supported, and explicitly enumerating the ones for incomplete classes.

Chapter 8. Filter Conceptual Model and VTP2 testing considerations

8.1. Conceptual model

The following diagram provides a draft of the filter conceptual model, the elements, their relationship to the other significant elements available when using a filter in a service or GeoPackage implementation.

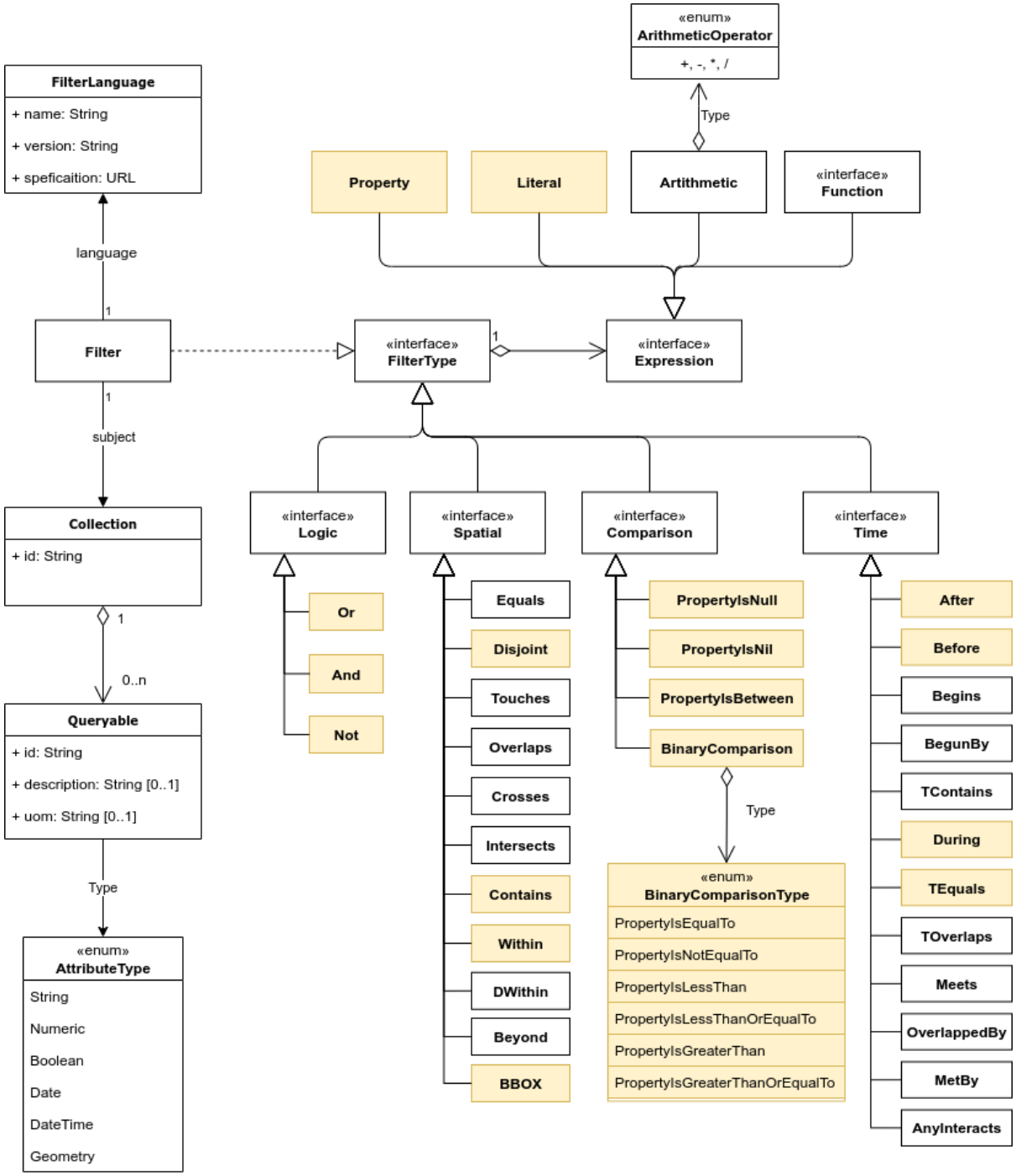


Figure 1. Filter conceptual models

In particular, a implementation of the model refers to:

- A filter language, an encoding, typically textual, that expresses the filter
- A collection, which is the target of the filtering
- A hierarchy of filter types and expression concepts, that can be used to compose a filter

The collection in turn exposes a set of queryables, property names and types that can be used to build a filter, and to verify its formal correctness.

The filter refers to the common building blocks. In summary:

- Expressions, be they property references, literals, functions and arithmetic combinations thereof.
- Filters, in particular, spatial ones, temporal, simple comparisons and logic combinations thereof.

Please refer to 09-026r2, OGC® Filter Encoding 2.0 Encoding Standard – With Corrigendum [http://docs.opengeospatial.org/is/09-026r2/09-026r2.html] for a detailed definition of each expression and filter.

8.2. The CQL language

The OGC API - Features Standards Working Group (SWV) is currently exploring a filtering extension and are evaluating a number of possible encodings. In particular, two encodings are currently being considered for use in the filtering extension:

- An extended version of the OGC Catalogue Common Query Language [http://docs.opengeospatial.org/is/12-168r6/12-168r6.html#14] (CQL)
- A JSON encoding of the filter conceptual model, based either on a hierarchical structure, similar to the OGC Filter XML encoding, or an array based structure, similar to the Mapbox GL filter expressions [https://docs.mapbox.com/help/glossary/filter/].

The JSON encoding is recognized to be most useful to express filters in a larger JSON document, while the CQL based syntax is considered more useful for direct human entry and usage in URLs.

Within the limits of Vector Tiles Pilot 2, the CQL language was chosen for prototyping purposes, as CQL is currently better defined, and better supports the target usage within tile resource URLS.

The CQL language is a text-based language that reads like a SQL 'Where' clause expresion, as it is regarded as easy to read and to enter directly, without support by a user interface. On the other hand, due to the text nature, CQL is more complex to handle software-wise, as a full parser needs to be written to handle it. By comparison, the two JSON structures mentioned above, can be parsed using a JSON parser, and would construct an object structure that is very similar to a parse tree, ready for use and evaluation.

Usage of CQL outside of the confines of Catalog Services has been pioneered [https://osgeo-org.atlassian.net/browse/GEOS-727] by GeoServer [https://docs.geoserver.org/latest/en/user/filter/syntax.html#cql-ecql] community. GeoServer is an open source project which began using an extended version of the CQL language in both the WFS and WMS protocols:

- In WFS, CQL can be used as an alternative to the XML filter encoding in GetFeature requests
- In WMS, CQL can be used as a way to further filter map contents, in addition to the filtering already performed by the Styled Layer Descriptor (SLD) styles.

In both cases the user can add a `&cql_filter=<filter expression>` query parameter in the request, and get back filtered results.

Compared to the Catalog CQL, the proposed filtering extension of OGC API - Features provided some improvements:

- Some of the original restrictions have been lifted, making this approach as expressive as OGC FE, and similar to the version adopted by GeoServer. For example, the original CQL did not allow comparing two properties, but only a property with a literal.
- The BNF was updated to correctly validate, and can now be parsed by a machine.

A copy of the current BNF version can be found in Annex A [#CQL_BNF].

The following table shows a few examples of text CQL filters:

Filter meaning	**CQL encoding**
Floors greater than 5	`floors>5`
Owner name contains 'Jones'	`` `owner LIKE '% Jones %'` ``
More than 5 floors and a swimming pool	`` `floors>5 AND swimming_pool=true` ``
A swimming pool and (more than five floors or material is brick)	`` `swimming_pool=true AND (floors > 5 OR material LIKE 'brick%' OR material LIKE '%brick')` ``
Updated between 7:30am June 10, 2017 and 10:30am June 11, 2017	`` `updated DURING 2017-06-10T07:30:00 2017-06-11T10:30:00` ``
Geometry that intersects with geometry `POLYGON((-10.0 -10.0,10.0 -10.0,10.0 10.0,-10.0 -10.0))` (as expressed in WKT)	`INTERSECTS(geometry,POLYGON\((-10.0 -10.0,10.0 -10.0,10.0 10.0,-10.0 -10.0)))`

Some considerations:

- The filter is normally human readable and quite similar to SQL
- Geometries are expressed in WKT syntax when possible
- Time literals are expressed in ISO format

Chapter 9. Filtering Tiles

This ER Section focuses on filtering tile contents. Final decisions and recommended API changes needed to generate outputs that match the VTP2 objectives are explained.

9.1. Vector Tiles Pilot 2 testing operator subset

The VTP2 implementations exercised the following subset of operators:

Table 3. VTP2 filter operator subset

Class	Operators
Spatial	BBOX, Contains, Within, Intersects, Disjoint
Temporal	During, Before, After, TEquals
Comparison	All
Logical	All
Expression	Property, Literal

The above set was selected because it is simple enough to be implemented within the time frame of VTP2, while being representative of the overall model filtering abilities.

9.2. Queryables

The list of valid properties, and their type, must be available for clients to build filters.

As indicated in Chapter 6, Testbed 15 Styles's Engineering Report extended the OGC API - Features with a concept of Queryables, a resource enumerating all attributes available for filtering.

The VTP2 participants used the same extension to advertise collection queryables, both in the OGC API - Features, and in the OGC API - Tiles (for those implementing a stand-alone Tile API, as opposed to a tiles building block in the Features API).

9.3. Single and multi-layer tile filtering

The VPT2 participants considered two filtering cases:

- Filtering a tile from a single collection
- Filtering a multi-layer collection

The first use case, filtering a single layer tile, is well supported by the existing work, in particular, the OGC API - Features CQL extension, as well as the Queryables extension, with the following workflow:

- The client inspects the queryables
- The client helps the user to build a filter reducing the features returned in tiles
- A request for tiles with the `filter` and `filter-lang` parameters is sent to the Features API tiles

extension, or to the stand-alone Tiles API, retrieving tiles with reduced content.

The second use case presents some extra complexities, in particular:

- Discovery of queryables for all the layers included in the tile must be possible.
- Providing multiple filters, one per collection, to the server must be possible.

Regarding **queryables discovery**, the issue is easily solved in an OGC API - Features providing multi-layer tiles at the root level `/tiles` resource. In this case, the client has to list the collections to be retrieved, and thus, can lookup the queryables on a layer by layer basis. A stand-alone Tiles API poses a challenge, as collections are opaque and can contain anything, including having a single collection delivering directly multi-layer tiles (see the GeoSolutions implementation in this regard). The queryables document, as a flat list of attributes, is insufficient for this case, and should be turned into a list of named layers, each having their own queryables.

The second issue, **providing multiple filters**, does not have a solution in the existing CQL language. GeoServer Extended CQL [http://old.geotools.org/ECQL-Parser-Design_110493908.html] supports a notion of "filter list", used to provide multiple filters in WMS GetMap requests. The grammar uses the semicolon to separate filters, as the comma is already used in the `IN` operator.

```
SequenceOfSearchConditions> ::=
          <search condition>
        | <SequenceOfSearchConditions> ; <search condition>
```

Another possibility would be to repeat the "filter" and "filter-lang" parameters, one instance per filtered collection, in the URL.

Regardless of the solution adopted, most base maps use a large number of layers, for example, an openstreemap.org [https://openstreemap.org] like map typically uses over 20 layers. Trying to filter on this would require placing over 20 filters in the request, quickly exhausting the practical length of a URL.

For these reasons, with the exception of the Ecere Tiles API, the VTP2 participants limited filtering to single layer collections, while filtering multi-layer collections with CQL is left as future exercise.

Ecere implemented support for compact multi-layer filtering expressions based on the selectors and expressions syntax from the CMSS styling language (see Chapter 8 - implementations).

Chapter 10. Filtering GeoPackages with Vector Tiles

GeoPackage is built on the SQLite relational database. As a relational database, filtering of feature data is easily done through conventional SQL querying. However, when the feature data is stored in vector tiles, filtering is more complicated. Both formats used in VTP2 (MVT and GeoJSON) embed attribute information in the vector tiles. Keeping the attributes embedded in the vector tiles undermines the capabilities of a GeoPackage-based architecture for the following reasons:

1. Since features may span multiple tiles, having the attribute information duplicated across each tile containing a particular feature is redundant.
2. Queries against the embedded attributes are not possible without opening a number of candidate tiles individually, an inefficient process.
3. There is no obvious way to identify the candidate tiles to open, beyond knowledge of the area of interest of a particular query.

In response to these concerns, the participants proposed some alternatives for managing attributes. Due to resource constraints, these approaches were only tested in a limited fashion.

10.1. Attributes in Attributes Tables

Through the Vector Tiles Attributes Extension (see the Summary ER), a GeoPackage may contain an attributes table [https://www.geopackage.org/spec121/#attributes] for each vector tiles layer. When this approach is used, two following benefits are attained. First, features can be filtered via their attributes using conventional SQL queries. Second, the attributes for a feature can be removed from the vector tiles so that they are only stored in the GeoPackage once.

Depending on the underlying architecture of the vector tiles server, this approach may impose significant additional processing during GeoPackage creation. In some architectures, the vector tiles are created in advance. In naive GeoPackage creation (ignoring filtering considerations), those vector tiles can simply be copied directly into the GeoPackage. However, in this scenario the vector tiles would have to be modified to remove the attributes. This would significantly increase the processing time required to produce the GeoPackage. An architecture where vector tiles are produced on-the-fly would not be subject to this performance impact.

10.2. Spatial Filtering

The approach described in the previous section provides significant benefits for scenarios where features need to be filtered by their attributes and spatial extents. However, there is a limitation if it is not possible or practical to isolate which tiles to open to find the geometries for the features that satisfy a particular query.

An R-tree spatial index, storing the overall extent of whole (untiled) features, can be used in conjunction with the attributes table to handle arbitrarily large geospatial extents. Such an index can speed up queries by focusing on the features present in a specific area, only performing more costly attributes comparison for those features within the bounding box of a query. That same

stored extent can also be used to easily identify where specific features are located, e.g. to center the view on a particular feature, potentially after having identified that feature by an attributes-based query.

The GeoPackage Related Tables Extension [http://www.geopackage.org/guidance/extensions/related_tables.html] can be used to establish a many-to-many mapping between features and the tiles containing those features. Once this is done, the query can be performed to identify a result set (based on feature IDs) and the mapping table can be queried to identify the tile or tiles that contain the geometries for those features. In some scenarios, this will improve the performance of filtering operations.

The following examples are practical queries performed on a GeoPackage produced by Ecere of the Daraa / OpenStreetMap Topographic DataStore for the pilot, with vector data tiled according to the World Mercator WGS84 Tile Matrix Set, and using the Attributes Table extension, the Tiles/Features Mapping Table extension, as well as a 32-bit integer R-tree spatial index (storing the extent of features as decimal degrees multiplied by a factor of 10^7, maintaining the same precision as OpenStreetMap).

Querying features and attributes

In this first example query, the client cares about neither the detailed geometry nor the tiles of the features, but does care about features within a specific geographic region. The client wishes to list the GeoPackage feature IDs, as well as the values for four of the attribute fields (*UFI*, *F_CODE*, *FCSUBTYPE* and *ZI005_FNA*) of the Daraa transportation lines (roads) layer, for all those features whose name (*ZI005_FNA*) is not *No Information*, and which are at least partially located within a geographic bounding box whose lower-left and upper-right corners are respectively (32.6083233°N, 32.6083233°E) and (32.6097047°N, 36.0987994°E). An inner join is used between the attributes table and the R-tree spatial index, whose *id* fields correspond.

```
select attributes_TransportationGroundCrv.id, UFI, F_CODE, FCSUBTYPE, ZI005_FNA
   from attributes_TransportationGroundCrv
   inner join rtree_attributes_TransportationGroundCrv_vector_tiles
   on attributes_TransportationGroundCrv.id =
rtree_attributes_TransportationGroundCrv_vector_tiles.id
   where ZI005_FNA != 'No Information' and
   maxLat > 326083233 and maxLon > 360899582 and
   minLat < 326097047 and minLon < 360987994;
```

Output (list of features):

id	UFI	F_CODE	FCSUBTYPE	ZI005_FNA
1442	36d6beb3-0a35-4ec3-a2f6-6b526ea1f659	AP030	100152	
3178	204af963-3d18-402d-8cfe-ff4195e992c7	AP030	100152	

Identifying tiles containing certain features

In this second example query, the client wishes to preserve the above conditions (only roads with name information, within that same bounding box), but rather than retrieving the attributes values, all the tiles (specifically of zoom level 16, in this case) in which those features will be found should be listed. Because this GeoPackage includes a Tiles / Features Mapping table, that table can be joined with both the Attributes table (through its *related_id*) as well as with the Tiles table (through its *base_id*). Note that the tiles returned by this query cover the entirety of the unclipped features which happen to intersect the bounding box of interest — not only the tiles which themselves intersect the bounding box.

```
select distinct tiles_Daraa2.id, zoom_level, tile_row, tile_column
   from mapping_table_TransportationGroundCrv
   inner join tiles_Daraa2
   on mapping_table_TransportationGroundCrv.base_id = tiles_Daraa2.id
   inner join attributes_TransportationGroundCrv
   on mapping_table_TransportationGroundCrv.related_id =
attributes_TransportationGroundCrv.id
   inner join rtree_attributes_TransportationGroundCrv_vector_tiles
   on attributes_TransportationGroundCrv.id =
rtree_attributes_TransportationGroundCrv_vector_tiles.id
   where ZI005_FNA != 'No Information'
   and maxLat > 326083233 and maxLon > 360899582
   and minLat < 326097047 and minLon < 360987994
   and zoom_level = 16;
```

Output (list of tiles):

id	zoom_level	tile_row	tile_column
7962	16	26518	39339
8059	16	26519	39338
8060	16	26519	39339
8161	16	26520	39337
8162	16	26520	39338
8163	16	26520	39339

Identifying tiles without a mapping table

Even without a Tiles / Features Mapping table, a client could still easily determine a list of tiles from the extent of the feature(s) as stored in the R-tree spatial index. For a large feature, often only the tiles in view are of interest, in which case the client could also calculate the intersection of the feature's extent with the view's extent to identify those tiles. It could then proceed to enumerate the full list (complete box) of tiles within that extent, in a similar way as how the limits of a tile matrix are calculated for a dataset's extent. The advantage of the Mapping table is to altogether skip tiles in which the features of interest may not be present at all, therefore avoiding to decode tiles not containing these. This would most likely only benefit large features spread non-uniformly across a vast geospatial extent, and come at the cost of additional storage, which should be taken into

consideration. The benefit would also not apply to scenarios where all tiles within a geographic area of interest should be decoded anyways (e.g. the simple visualization use case).

In this third example query, it is assumed that a Tiles / Features Mapping table is not available, and therefore the output of the query is an extent which will be programmatically converted to tile coordinates.

```
select min(minLat), min(minLon), max(maxLat), max(maxLon)
   from attributes_TransportationGroundCrv
   inner join rtree_attributes_TransportationGroundCrv_vector_tiles
   on attributes_TransportationGroundCrv.id =
rtree_attributes_TransportationGroundCrv_vector_tiles.id
   where ZI005_FNA != 'No Information'
   and maxLat > 326083233 and maxLon > 360899582
   and minLat < 326097047 and minLon < 360987994;
```

Output (extent of the unclipped features):

min(minLat)	min(minLon)	max(maxLat)	max(maxLon)
326079262	360884617	326191309	361000583

The extent returned in this case is larger and fully contains the extent of interest in the request. Converting the geographic extent to WorldMercatorWGS84Quad level 16 tile coordinates gives (26518, 39337)-(26520, 39339).

Here is the full list of tiles using those bounding tile coordinates, assuming the whole unclipped features are desired:

```
select id, zoom_level, tile_row, tile_column
   from tiles_Daraa2
   where zoom_level = 16
   and tile_row >= 26518 and tile_row <= 26520
   and tile_column >= 39337 and tile_row <= 39339;
```

Output (list of tiles):

id	zoom_level	tile_row	tile_column
7960	16	26518	39337
7961	16	26518	39338
7962	16	26518	39339
8058	16	26519	39337
8059	16	26519	39338
8060	16	26519	39339
8161	16	26520	39337
8162	16	26520	39338

8163	16	26520	39339

It can be seen that the Mapping table request in this scenario would save the client from decoding 3 out of 9 tiles (33%), which might be significant.

If the client is instead interested in the clipped portion rather than the unclipped features (i.e. only the tiles which themselves interesect the bounding box of interest), then the intersection of the returned extent with the requested extent should be used. This might have justified avoiding the query altogether and directly converting the area of interest to a list of tiles, depending on the circumstances. Converting this intersected extent (which here is the same as the original requested bounding box) to WorldMercatorWGS84Quad level 16 tile coordinates gives (26520, 39337)-(26520, 39339).

The full list of tiles for the portion of the features of interest intersecting the area of interest is:

```
select id, zoom_level, tile_row, tile_column
   from tiles_Daraa2
   where zoom_level = 16
   and tile_row >= 26520 and tile_row <= 26520
   and tile_column >= 39337 and tile_row <= 39339;
```

Output (list of tiles):

id	zoom_level	tile_row	tile_column
8161	16	26520	39337
8162	16	26520	39338
8163	16	26520	39339

In this case, only 3 tiles are needed, and a query using the Mapping table does not eliminate any tile from the full box of tiles covered by the extent, since all tiles of the bounding box contain those features (no saving).

Chapter 11. Implementations

Each component provides filtering abilities, available both on the client and server side. This chapter includes salient implementation notes, descriptions and demonstrations, as well as relevant feedback from each deliverable.

11.1. GeoSolutions D100 OGC API - Features

GeoSolutions worked on extending the OGC API - Features to match the requirements of the VTP2 initiative. Specifically to filtering and tiles, the following changes were implemented:

- Exposed GeoServer (E)CQL filtering capabilities as an implementation of the current OGC API Features CQL extension [https://github.com/opengeospatial/ogcapi-features/tree/master/extensions/cql] draft, along with queryables support.
- Implemented the draft OGC API - Tiles building blocks as part of the OGC API - Features, and added filtering support in the same way as the OGC API - Features extensions.

When the Tiled Features plugin is added, the landing page of the service reports the tile matrix set endpoint, as well as advertising tiling resources in the API definition.

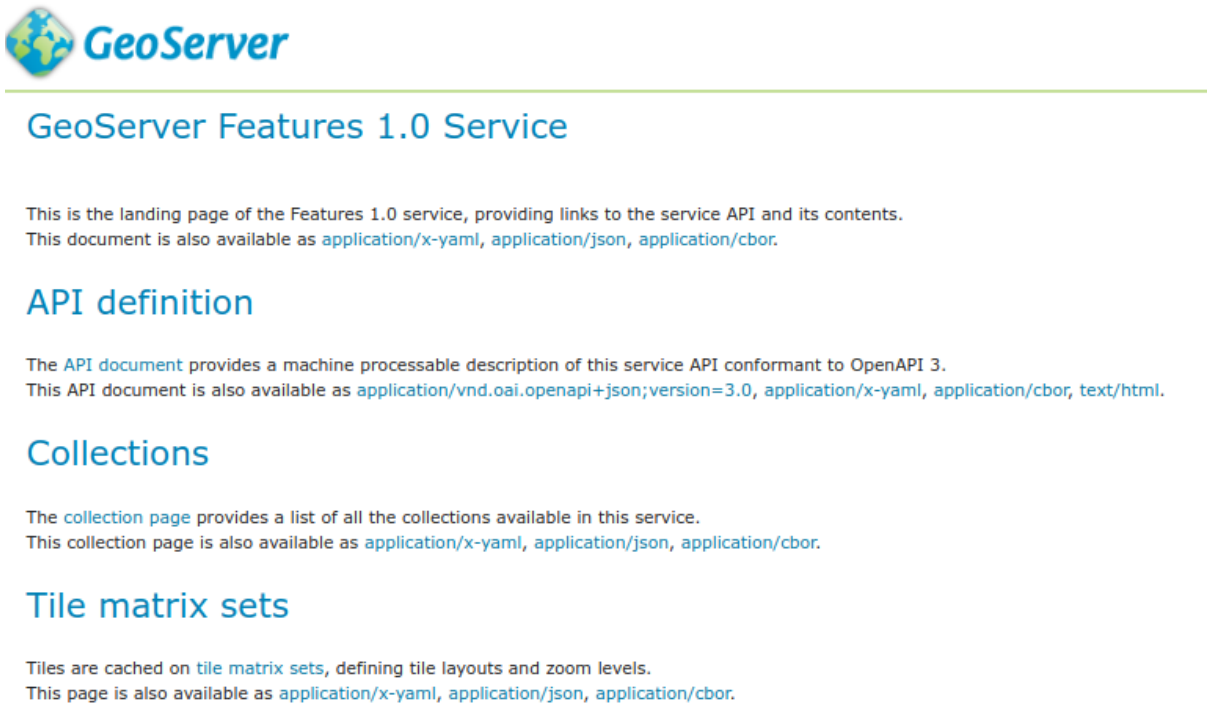


Figure 2. The landing page of the OGC API - Features allows access to the tile matrix sets.

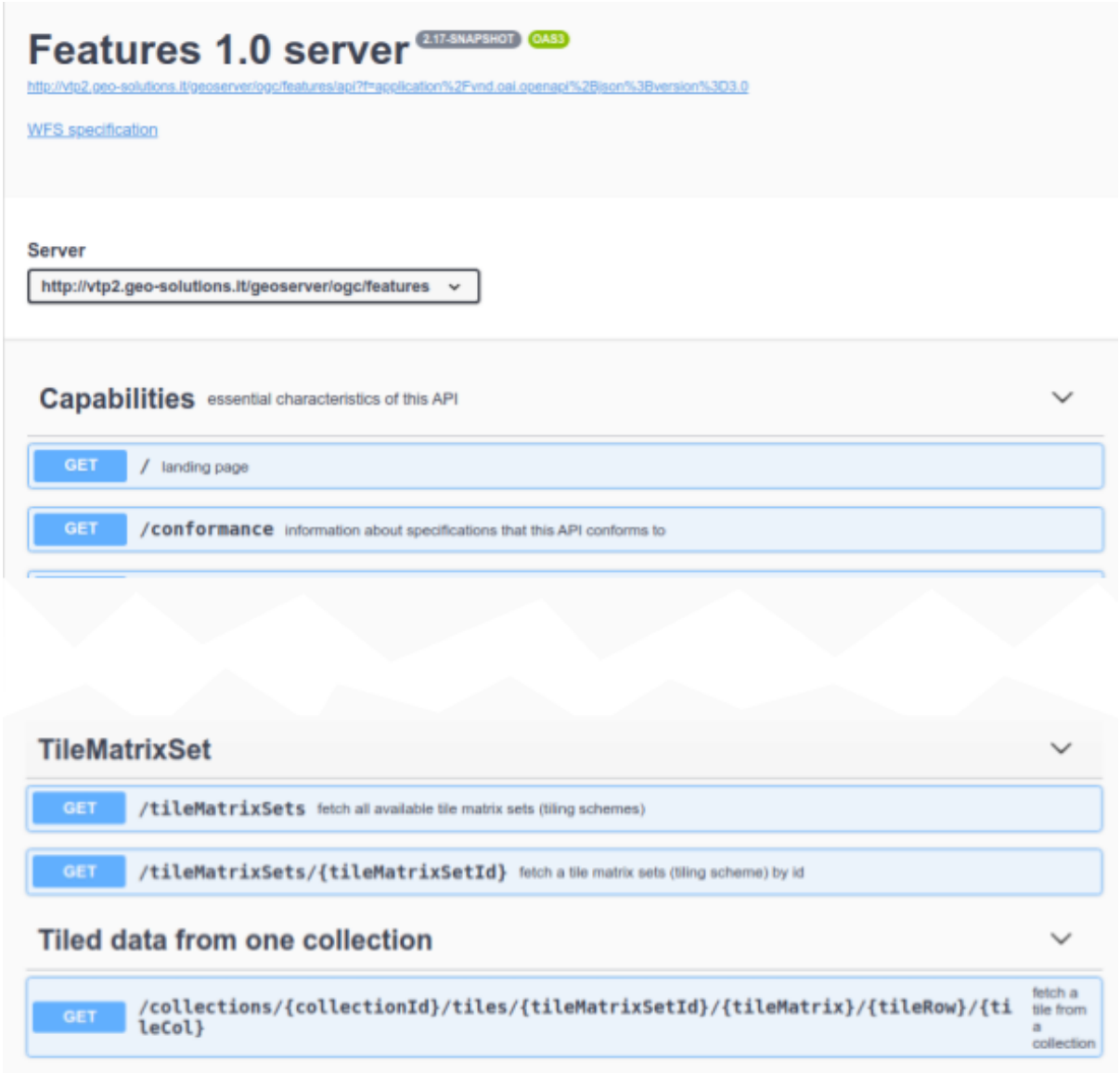


Figure 3. The OGC API - Features advertises tile matrix set and data tile access.

Queryables support has been added in all collections. The following is an excerpt from the queryables of the `syria_vtp:building_s` collection (only a few attributes are shown to keep the example short):

http://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp:TransportationGroundCrv/queryables?f=json


```
{
  "queryables": [
    {
      "id": "AOO",
      "type": "number"
    },
    {
      "id": "ARA",
      "type": "number"
    },
    {
      "id": "BEN",
```

```
      "type": "string"
    },
    {
      "id": "CAA",
      "type": "integer"
    },
    {
      "id": "CCN",
      "type": "string"
    },
    {
      "id": "geom",
      "type": "geometry"
    }
  ],
  "links": [
    {
      "href": "http://vtp2.geo-
solutions.it/geoserver/ogc/features/collections/vtp%3ATransportationGroundCrv?f=applic
ation%2Fx-yaml",
      "rel": "alternate",
      "type": "application/x-yaml",
      "title": "This document as application/x-yaml"
    },
    {
      "href": "http://vtp2.geo-
solutions.it/geoserver/ogc/features/collections/vtp%3ATransportationGroundCrv?f=applic
ation%2Fjson",
      "rel": "self",
      "type": "application/json",
      "title": "This document"
    },
    {
      "href": "http://vtp2.geo-
solutions.it/geoserver/ogc/features/collections/vtp%3ATransportationGroundCrv?f=text%2
Fhtml",
      "rel": "alternate",
      "type": "text/html",
      "title": "This document as text/html"
    }
  ]
}
```


Support for the `filter` and `filter-lang` query parameters has been added both to the `items` and the `tiles` resources.

An example of filtered `items` request follows, reformatted and URL decoded for readability:

https://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp:TransportationGroundCrv/items?f=json&
filter=(%22RIN_ROI%22%20%3D%20%273%27)%20AND%20(BBOX(geom,36.140,32.595,36.142,32.65

8))&filter-lang=cql-text

- `https://vtp2.geo-solutions.it/geoserver/ogc/features` (service base)
- `collections/vtp:TransportationGroundCrv/items?` (items access)
- `f=application/vnd.mapbox-vector-tile` (requests to return Mapbox Vector Tile format)
- `filter=("RIN_ROI" = '3') AND (BBOX(geom,36.140,32.595,36.142,32.658))&` (the filter)
- `filter-lang=cql-text` (filter language is CQL)

An example of single collection filtered `tiles` request follows, reformatted and URL decoded for readability:

https://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp:TransportationGroundCrv/tiles/WebMercatorQuad/14/6618/9833?f=application/vnd.mapbox-vector-tile&filter=(%22RIN_ROI%22%20=%20%273%27)&filter-lang=cql-text

- `https://vtp2.geo-solutions.it/geoserver/ogc/features` (service base)
- `collections/vtp:TransportationGroundCrv/tiles/WebMercatorQuad/14/6618/9833?` (tile access)
- `f=application/vnd.mapbox-vector-tile` (requests to return mapbox vector tiles)
- `filter=("RIN_ROI" = '3')&` (the filter)
- `filter-lang=cql-text` (filter language is CQL)

Regarding caching, it is worth noting that a layer needs to have some caching configuration in order to expose tiled access, as tiling is internally managed by GeoWebCache. It is however not required to make filtered tiles cacheable, if not enabled, then the filtered tiles are computed on-the-fly.

11.2. Terranodo D100 OGC API - Tiles

The open source tegola [https://tegola.io] software implementated the OGC API - Features CQL extension draft, along with collection level queryables support.

11.2.1. CQL

CQL was implemented in tegola with the following functionality:

- CQL filters are supported via the `filter=` parameter
- CQL text is supported. CQL JSON is planned for future implementation

11.2.2. Sample Requests

Spatial Predicates example requests, CQL filter first, followed by a full URL-encoded API request:

```
INTERSECTS(geometry,POLYGON((36.0710334777832 32.59845703812064, 36.13574981689453
32.59845703812064, 36.13574981689453 32.633592568907005, 36.0710334777832
32.633592568907005, 36.0710334777832 32.59845703812064)))`
```

https://ogc-vtp.gospatial.org/maps/WebMercatorQuad/transport_lines/13/4917/3309.pbf?filter=(INTERSECTS(geometry%2CPOLYGON36.0710334777832%2032.59845703812064%2C%2036.13574981689453%2032.59845703812064%2C%2036.13574981689453%2032.633592568907005%2C%2036.0710334777832%2032.633592568907005%2C%2036.0710334777832%2032.59845703812064))&filter-lang=cql-text

```
WITHIN(geometry,POLYGON((36.0710334777832 32.59845703812064, 36.13574981689453
32.59845703812064, 36.13574981689453 32.633592568907005, 36.0710334777832
32.633592568907005, 36.0710334777832 32.59845703812064)))
```

https://ogc-vtp.gospatial.org/maps/WebMercatorQuad/transport_lines/13/4917/3309.pbf?filter=(WITHIN(geometry%2CPOLYGON36.0710334777832%2032.59845703812064%2C%2036.13574981689453%2032.59845703812064%2C%2036.13574981689453%2032.633592568907005%2C%2036.0710334777832%2032.633592568907005%2C%2036.0710334777832%2032.59845703812064))&filter-lang=cql-text

```
DISJOINT(geometry,POLYGON((34.73876953125 34.53823752729575, 37.825927734375
31.835565983656224, 38.72680664062501 34.06631196472105, 34.73876953125
34.53823752729575)))
```

https://ogc-vtp.gospatial.org/maps/WebMercatorQuad/transport_lines/7/78/50.pbf?filter=(DISJOINT(geometry%2CPOLYGON34.73876953125%2034.53823752729575%2C%2037.825927734375%2031.835565983656224%2C%2038.72680664062501%2034.06631196472105%2C%2034.73876953125%2034.53823752729575))&filter-lang=cql-text

Non spatial predicates examples, CQL filter first, followed by a full URL-encoded API request:

```
class='landuse'
```

https://ogc-vtp.gospatial.org/maps/WebMercatorQuad/landuse_areas/13/4914/3309.pbf?filter=class%3D%27landuse%27&filter-lang=cql-text

```
class like '%land%T'
```

https://ogc-vtp.gospatial.org/maps/WebMercatorQuad/landuse_areas/13/4914/3310.pbf?filter=class%20LIKE%20%27%25land%25%27&filter-lang=cql-text

Temporal predicates examples:

```
BEFORE 2020-03-03`
```

https://ogc-vtp.gospatial.org/maps/WebMercatorQuad/syria_acled/12/2461/1655.pbf?filter=event_date%20BEFORE%202020-03-03

```
AFTER 2016-03-03
```

https://ogc-vtp.gospatial.org/maps/WebMercatorQuad/syria_acled/12/2461/1655.pbf?filter=event_date%20AFTER%202016-03-03

Logical Operators examples:

```
INTERSECTS(geometry,POLYGON((35.9 32.7,36 32.7,36 32.8,35.9 32.8,35.9 32.7))) AND
class='swamp'
```

https://ogc-vtp.gospatial.org/maps/WebMercatorQuad/landuse_areas/12/2457/1654.pbf?filter=INTERSECTS(geometry%2CPOLYGON35.9%2032.7%2C36%2032.7%2C36%2.8%2C35.9%2032.7)%20AND%20class%3D%27swamp%27

```
INTERSECTS(geometry,POLYGON((35.9 32.7,36 32.7,36 32.8,35.9 32.8,35.9 32.7))) OR
class='swamp'
```

https://ogc-vtp.gospatial.org/maps/WebMercatorQuad/landuse_areas/12/2457/1654.pbf?filter=INTERSECTS(geometry%2CPOLYGON35.9%2032.7%2C36%2032.7%2C36%2.8%2C35.9%2032.7)%20OR%20class%3D%27swamp%27

11.2.3. Queryables

The following resource encodes queryables as a JSON document:

https://ogc-vtp.gospatial.org/ogc-api-tiles/collections/syria_acled/queryables


```
{
  "queryables": [
    {
      "id": "wkb_geometry",
      "type": "geometry"
    },
    {
      "id": "event_id_no_cnty",
```

```
      "type": "string"
    },
    {
      "id": "event_date",
      "type": "date"
    },
    {
      "id": "sub_event_type",
      "type": "string"
    },
    {
      "id": "actor2",
      "type": "string"
    },
    {
      "id": "ogc_fid",
      "type": ""
    },
    {
      "id": "data_id",
      "type": "string"
    },
    {
      "id": "event_type",
      "type": "string"
    },
    {
      "id": "region",
      "type": "string"
    },
    {
      "id": "notes",
      "type": "string"
    },
    {
      "id": "interaction",
      "type": "string"
    },
    {
      "id": "source",
      "type": "string"
    },
    {
      "id": "admin3",
      "type": "string"
    },
    {
      "id": "location",
      "type": "string"
    },
    {
```

```
      "id": "event_id_cnty",
      "type": "string"
    },
    {
      "id": "actor1",
      "type": "string"
    },
    {
      "id": "country",
      "type": "string"
    },
    {
      "id": "admin1",
      "type": "string"
    },
    {
      "id": "admin2",
      "type": "string"
    }
  ]
}
```

11.3. interactive instruments D101 Features, Tiles and Styles API

11.3.1. The starting point

At the beginning of the OGC Vector Tiles Pilot 2, the open-source tool ldproxy [https://github.com/interactive-instruments/ldproxy] implemented the following capabilities that were the basis for the work in the pilot:

- *OGC API - Features - Part 1: Core*: The following conformance classes were used in the pilot: Core, HTML, GeoJSON, OpenAPI 3.0. ldproxy was the first OGC Reference Implementation for the standard.
- *OGC API - Features - Part 2: Coordinate Reference Systems by Reference*: The latest draft (the public review started during the pilot).
- *OGC API - Tiles*: The following conformance classes of the latest draft were used in the pilot: Core, Tile Matrix Set, Tiles from more than one collection. Mapbox Vector Tiles and GeoJSON tiles were supported.
- *OGC API - Styles*: The following conformance classes of the latest draft were used in the pilot: Core, Resources, HTML, Mapbox Styles, Style Info, Queryables.
- A query parameter "properties" on all tile and feature resources to return only the selected properties in the response.

For the demonstration server, a test dataset with data from OpenStreetMap from the region of Daraa, Syria, converted to the Topographic Data Store (TDS) schema of NGA was used.

The following screenshots of the HTML view of the API resources illustrate the starting point for the work in the pilot. The HTML view is a rendering of the JSON content that was also available for each resource.

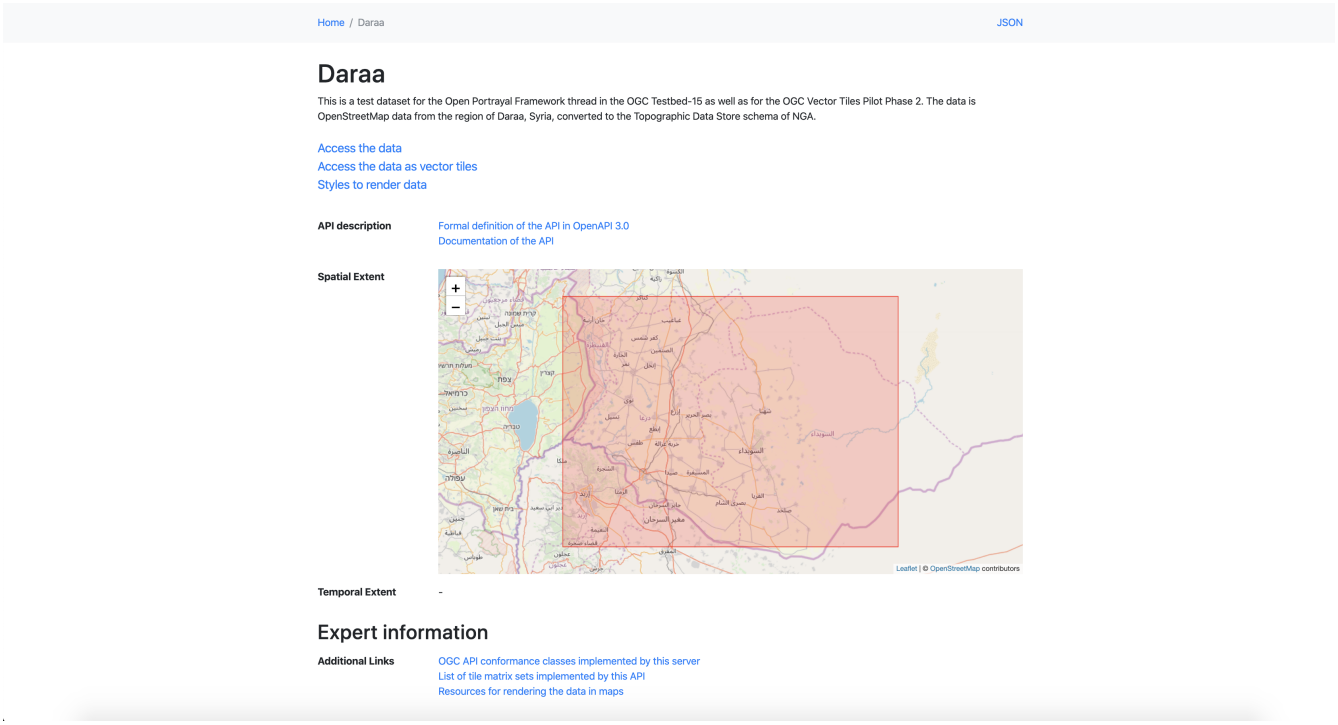


Figure 4. interactive instruments - the landing page of the API

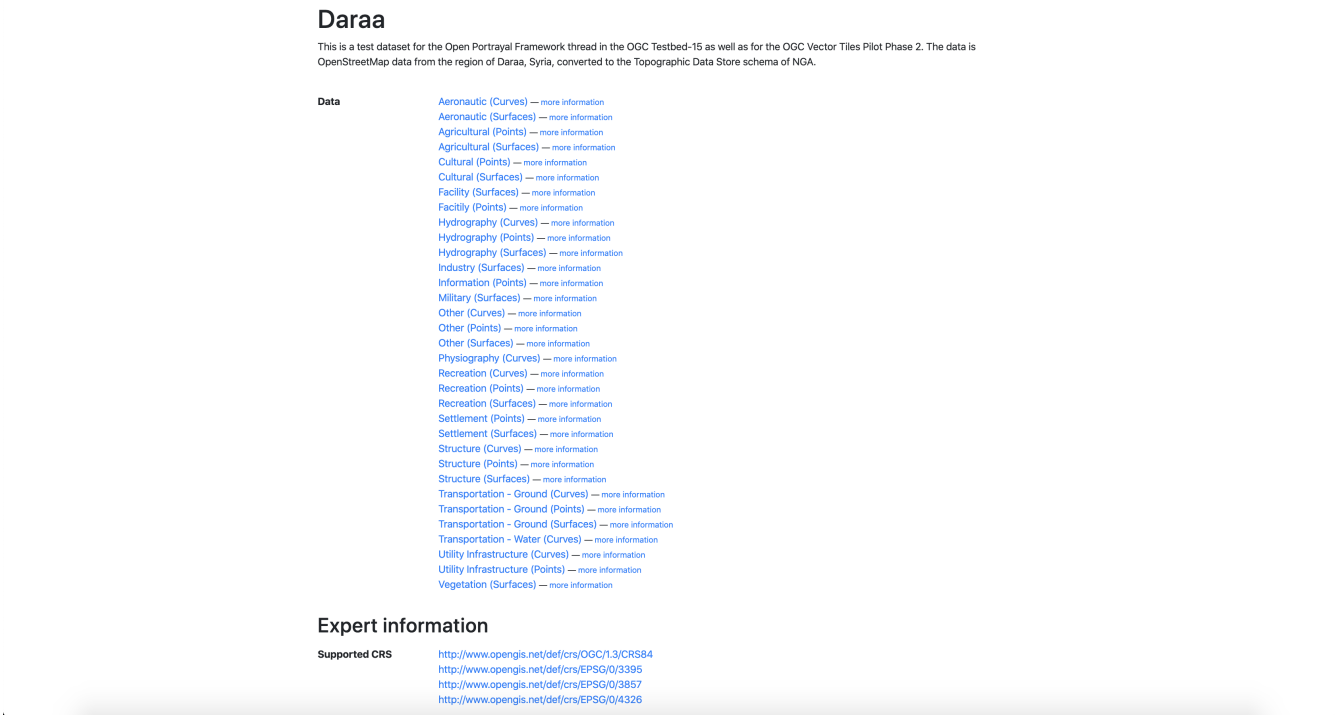


Figure 5. interactive instruments - the collections (feature types) in the dataset

For filtering, knowledge about the available properties for use in filter predicates is important. These were published as a separate resource for each collection, in the following screenshot for the Transportation (Ground) features with a line string geometry.

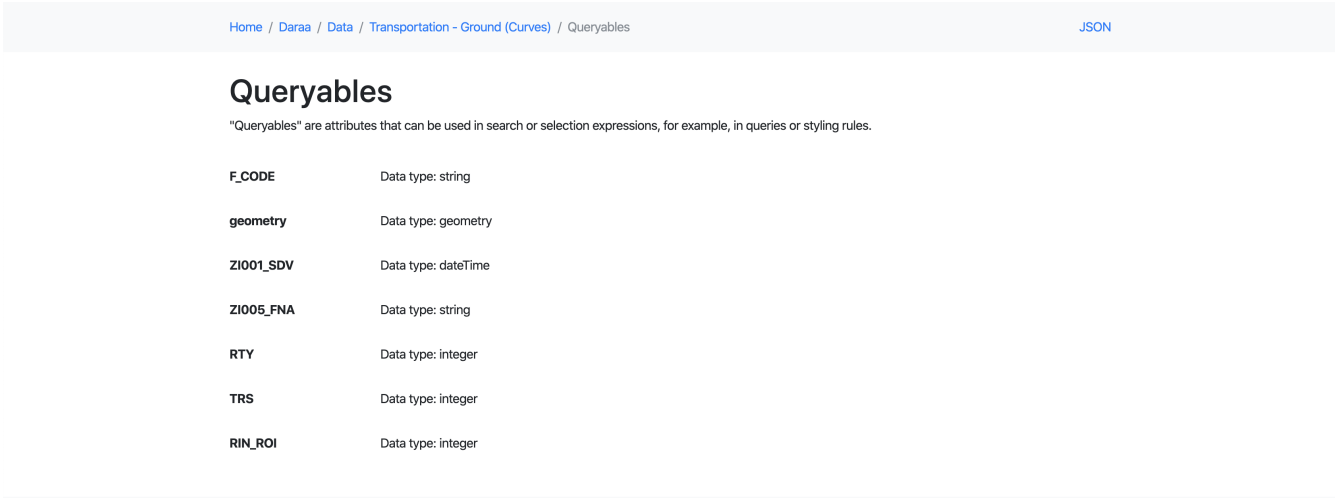


Figure 6. interactive instruments - queryable properties for the Transportation (Ground) features with line string geometry

To illustrate the data, here is a screenshot of a road feature:

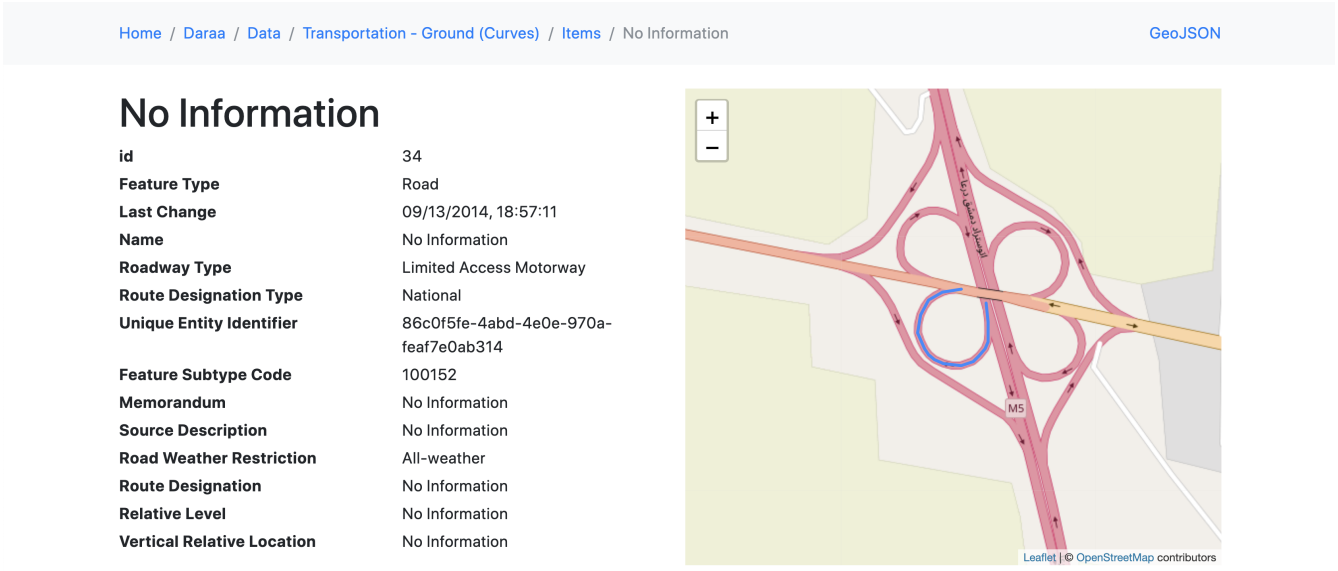


Figure 7. interactive instruments - a road feature

The following code snippet is the same feature in GeoJSON (coordinates have been truncated):

interactive instruments - a road feature in GeoJSON


```
{
   "type":"Feature",
   "links":[
      {
         "href":"https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv/items/34?f=json",
         "rel":"self",
         "type":"application/geo+json",
         "title":"This document"
      },
      {
         "href":"https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv/items/34?f=html",
         "rel":"alternate",
         "type":"text/html",
         "title":"This document as HTML"
      },
      {
         "href":"https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv?f=json",
         "rel":"collection",
         "type":"application/json",
         "title":"The collection the feature belongs to"
      }
   ],
   "id":"34",
   "geometry":{
      "type":"MultiLineString",
      "coordinates":[ ... ]
   },
   "properties":{
      "F_CODE":"AP030",
      "ZI001_SDV":"2014-09-13T18:57:11Z",
      "ZI005_FNA":"No Information",
      "RTY":2,
      "RIN_ROI":3,
      "UFI":"86c0f5fe-4abd-4e0e-970a-feaf7e0ab314",
      "FCSUBTYPE":100152,
      "ZI006_MEM":"No Information",
      "ZI001_SDP":"No Information",
      "ZI016_WTC":1,
      "RIN_RTN":"No Information",
      "RLE":-999999,
      "LOC":-999999
   }
}
```


All features were also available as vector tiles, both for each individual collection and a single

multi-layer tile set. The screenshot below shows the vector tiles of the Transportation (Ground) features in an OpenLayers map.

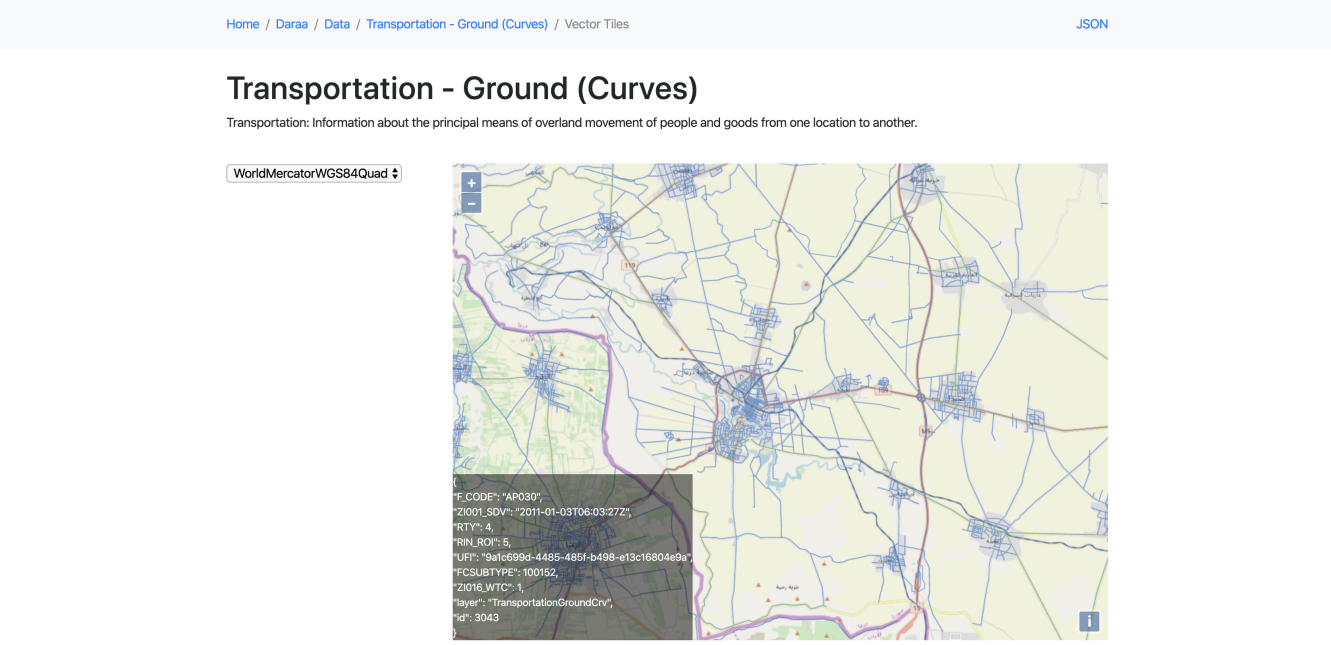


Figure 8. interactive instruments - Mapbox Vector Tiles for the Transportation (Ground) features in OpenLayers

11.3.2. Support for filters

Support for CQL filters was implemented in ldproxy and support for the query parameters `filter` and `filter-lang` was added to the following resources:

- `/collections/{collectionId}/items` - to filter the features resources
- `/collections/{collectionId}/tiles` - to filter the features in the vector tiles of a collection
- `/tiles` - to filter the features in the multi-collection vector tiles

Both representations of CQL expressions were supported (`cql-text` for text, `cql-json` for hierarchical).

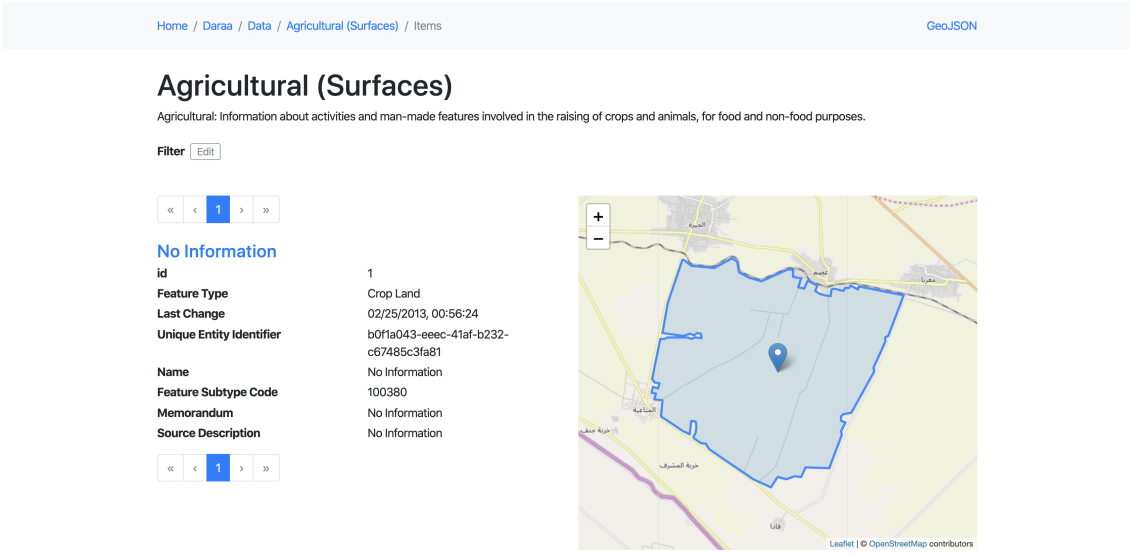
Most of the CQL grammar was implemented, with a few limitations:

- only the temporal operators specified in Filtering Tiles;
- no existence operators, since their semantics is not well-defined (see [Results]);
- no arithmetic expressions (add, subtract, multiply, divide).

Some examples for filters:

Fetch the agricultural area that contains a point	
CQL:	`CONTAINS(geometry,POINT(36.3544 32.4675))`

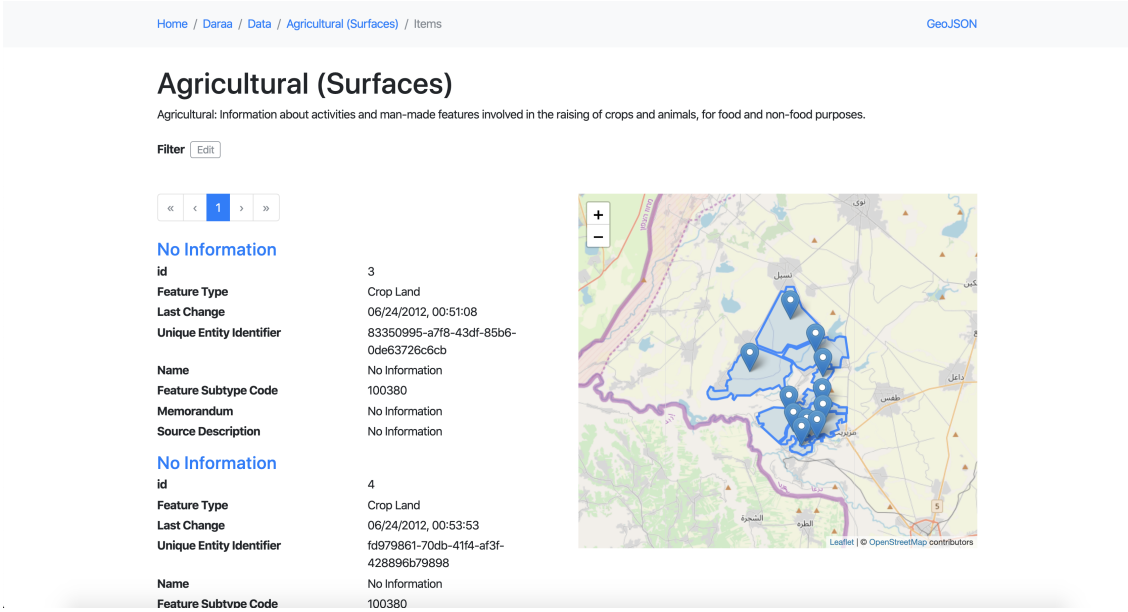
Features:	Request Link [https://services.interactive-instruments.de/t15/daraa/collections/AgricultureSrf/items?filter=CONTAINS(geometry%2CPOINT(36.3544%2032.4675))]



Fetch all other agricultural areas (not containing the point)	
CQL:	`DISJOINT(geometry,POINT(36.3544 32.4675))`
Features:	Request Link [https://services.interactive-instruments.de/t15/daraa/collections/AgricultureSrf/items?filter=DISJOINT(geometry%2CPOINT(36.3544%2032.4675))]

Fetch the agricultural areas in a polygon/bbox	
CQL:	`INTERSECTS(geometry,POLYGON( (35.9 32.7,36 32.7,36 32.8,35.9 32.8,35.9 32.7) ))`

Features:	link:https://services.interactive-instruments.de/t15/daraa/collections/AgricultureSrf/items?limit=100&filter=INTERSECTS(geometry%2CPOLYGON35.9%2032.7%2C36%2032.7%2C36%2032.8%2C35.9%2032.8%2C35.9%2032.7)[Request Link]

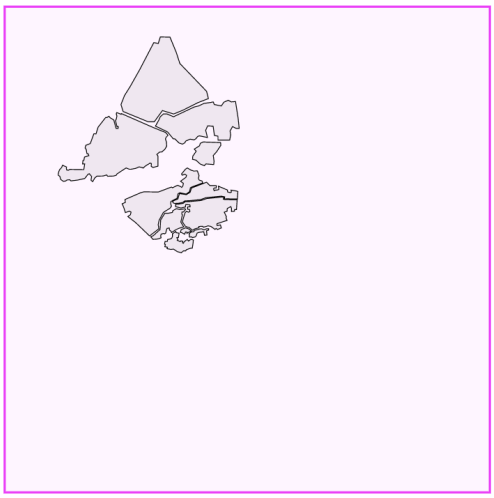


Tile:	link:https://services.interactive-instruments.de/t15/daraa/collections/AgricultureSrf/tiles/WebMercatorQuad/10/413/614?f=mvt&filter=INTERSECTS(geometry%2CPOLYGON35.9%2032.7%2C36%2032.7%2C36%2032.8%2C35.9%2032.8%2C35.9%2032.7)[Request Link]

resulting url or manual input

https://services.interactive-instruments.de/t15/daraa/collections/AgricultureSrf/tiles/WebMercatorQuad/10/413/614?f=mvt&filter=INTERSECTS(geometry%2CPOLYGON((35.9%2032.7%2C36%2032.7%2C36%2032.8%2C35.9%2032.8%2C35.9%2032.7)))

Use



File

Property	Value	Evaluation	Comment
Size (uncompressed)	2 KiB (1847 Bytes)	💚	
Content-Type	application/vnd.mapbox-vector-tile	💚	
Number of Layers	1	💚	

Layer Details

#	name	features	points	#Keys	#Values	keys	keyStringLength	valueTypes	valueStringLength
1	AgricultureSrf	11	441	4	20	F_CODE, ZI001_SDV, UFI, FCSUBTYPE	27	string 19 number 1	541

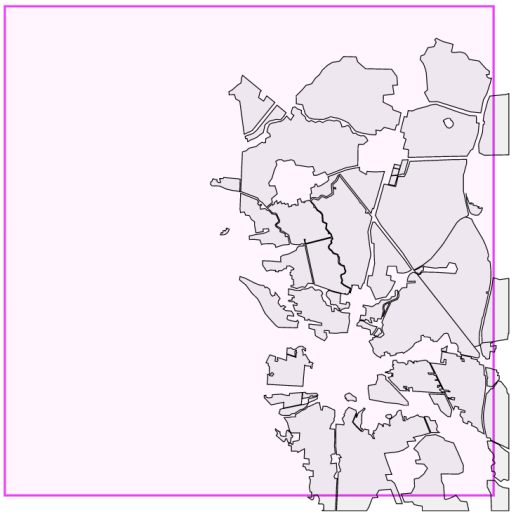
Fetch all the other agricultural areas	
CQL:	`DISJOINT(geometry,POLYGON( (35.9 32.7,36 32.7,36 32.8,35.9 32.8,35.9 32.7) ))`

Tile:	link:https://services.interactive-instruments.de/t15/daraa/collections/AgricultureSrf/tiles/WebMercatorQuad/10/413/614?f=mvt&filter=DISJOINT(geometry%2CPOLYGON35.9%2032.7%2C36%2032.7%2C36%2032.8%2C35.9%2032.8%2C35.9%2032.7)[Request Link]

resulting url or manual input

https://services.interactive-instruments.de/t15/daraa/collections/AgricultureSrf/tiles/WebMercatorQuad/10/413/614?f=mvt&filter=DISJOINT(geometry%2CPOLYGON((35.9%2032.7%2C36%2032.7%2C36%2032.8%2C35.9%2032.8%2C35.9%2032.7)))

Use



File

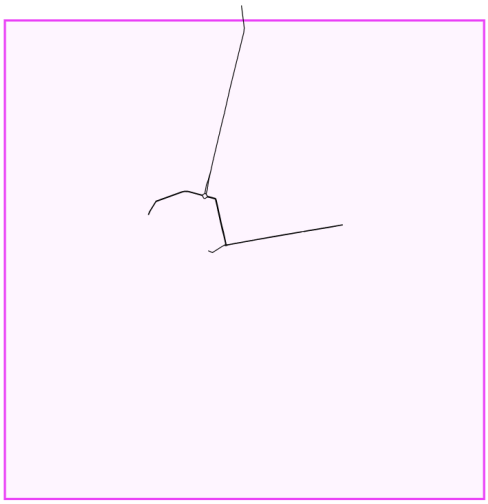
Property	Value	Evaluation	Comment
Size (uncompressed)	9 KiB (8698 Bytes)	💚	
Content-Type	application/vnd.mapbox-vector-tile	💚	
Number of Layers	1	💚	

Layer Details

#	name	features	points	#Keys	#Values	keys	keyStringLength	valueTypes	valueStringLength
1	AgricultureSrf	56	1987	4	103	F_CODE, ZI001_SDV, UFI, FCSUBTYPE	27	string 101 number 2	2886

Fetch the major roads in the eastern part of the city that were updated since 2013; only return selected attributes	
CQL:	`INTERSECTS(geometry,POLYGON( (36.1 32.6,36.15 32.6,36.15 32.65,36.1 32.65,36.1 32.6) )) AND F_CODE='AP030' AND RIN_ROI<4 AND ZI001_SDV AFTER 2013-01-01`

Features:	link:https://services.interactive-instruments.de/t15/daraa/collections/TransportationGroundCrv/items?limit=100&properties=ZI005_FNA,ZI001_SDV,RIN_ROI,RTY&filter=INTERSECTS(geometry%2CPOLYGON36.1%2032.6%2C36.15%2032.6%2C36.15%2032.65%2C36.1%2032.65%2C36.1%2032.6)%20AND%20F_CODE%3D%27AP030%27%20AND%20RIN_ROI%3C4%20AND%20ZI001_SDV%20AFTER%202013-01-01[Request Link]

Tile:	link:https://services.interactive-instruments.de/t15/daraa/collections/TransportationGroundCrv/tiles/WebMercatorQuad/11/827/1229?f=mvt&properties=ZI005_FNA,ZI001_SDV,RIN_ROI,RTY&filter=INTERSECTS(geometry%2CPOLYGON36.1%2032.6%2C36.15%2032.6%2C36.15%2032.65%2C36.1%2032.65%2C36.1%2032.6)%20AND%20F_CODE%3D%27AP030%27%20AND%20RIN_ROI%3C4%20AND%20ZI001_SDV%20AFTER%202013-01-01[Request Link] **resulting url or manual input** https://services.interactive-instruments.de/t15/daraa/collections/TransportationGroundCrv/tiles/WebMercatorQuad/11/827/1229?f=mvt&properties=ZI005_FNA,ZI001_SDV,RIN_ROI,RTY&filter=INTERSECTS(geometry%2CPOLYGON((36.1%2032.6%2C36.15%2032.6%2C36.15%2032.65%2C36.1%2032.65%2C36.1%2032.6)))%20AND%20F_CODE%3D%27AP030%27%20AND%20RIN_ROI%3C4%20AND%20ZI001_SDV%20AFTER%202013-01-01 Use  **File** **Property** / **Value** / **Evaluation** / **Comment** Size (uncompressed) / 1 KiB (1001 Bytes) / 💚 Content-Type / application/vnd.mapbox-vector-tile / 💚 Number of Layers / 1 / 💚 **Layer Details** **#** / **name** / **features** / **points** / **#Keys** / **#Values** / **keys** / **keyStringLength** / **valueTypes** / **valueStringLength** 1 / TransportationGroundCrv / 19 / 97 / 4 / 14 / ZI001_SDV, RTY, RIN_ROI, ZI005_FNA / 28 / string 12, number 2 / 235

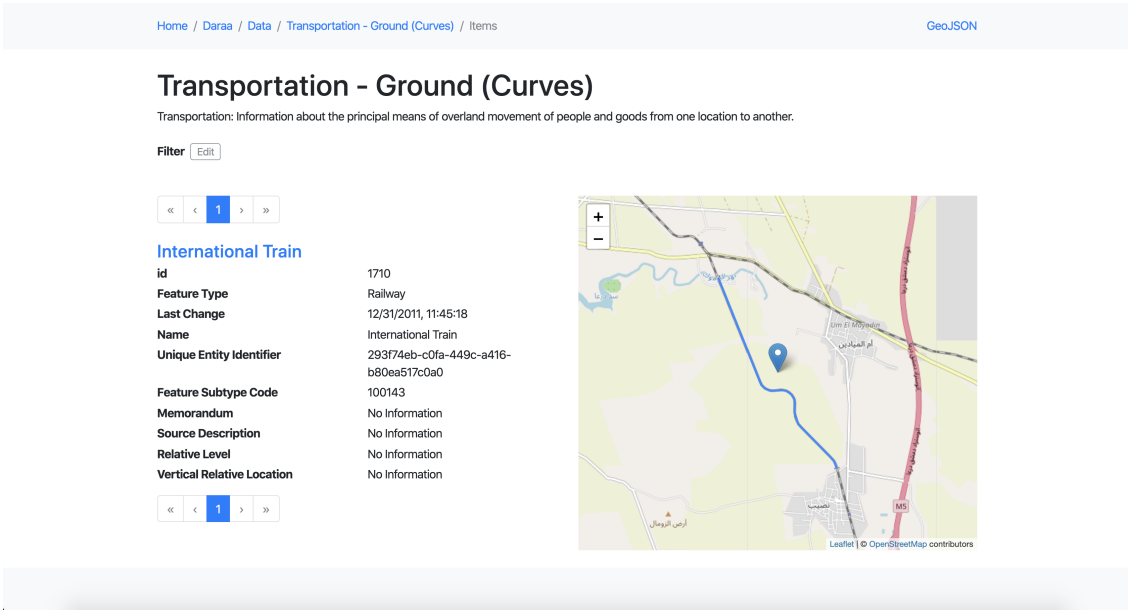
Fetch the local roads in the eastern part of the city that were updated during 2011 or 2012; only return selected attributes	
CQL:	`INTERSECTS(geometry,POLYGON( (36.1 32.6,36.15 32.6,36.15 32.65,36.1 32.65,36.1 32.6) )) AND F_CODE='AP030' AND RIN_ROI<4 AND ZI001_SDV AFTER 2013-01-01`
Features:	link:https://services.interactive-instruments.de/t15/daraa/collections/TransportationGroundCrv/items?limit=200&properties=ZI005_FNA,ZI001_SDV,RIN_ROI,RTY&filter=INTERSECTS(geometry%2CPOLYGON36.1%2032.6%2C36.15%2032.6%2C36.15%2032.65%2C36.1%2032.65%2C36.1%2032.6)%20AND%20F_CODE%3D%27AP030%27%20AND%20RIN_ROI%3D5%20AND

Tile:	link:https://services.interactive-instruments.de/t15/daraa/collections/TransportationGroundCrv/tiles/WebMercatorQuad/11/827/1229?f=mvt&properties=ZI005_FNA,ZI001_SDV,RIN_ROI,RTY&filter=INTERSECTS(geometry%2CPOLYGON36.1%2032.6%2C36.15%2032.6%2C36.15%2032.65%2C36.1%2032.65%2C36.1%2032.6)%20AND%20F_CODE%3D%27AP030%27%20AND%20RIN_ROI%3D5%20AND%20ZI001_SDV%20DURING%202011-01-01/2013-01-01[Request Link]

Fetch the ground transportation features last updated on 12/31/2011 at 11:45:18 UTC	
CQL:	`ZI001_SDV TEQUALS 2011-12-31T11:45:18Z`

%20ZI001_SDV%20DURING%202011-01-01/2013-01-01[Request Link]

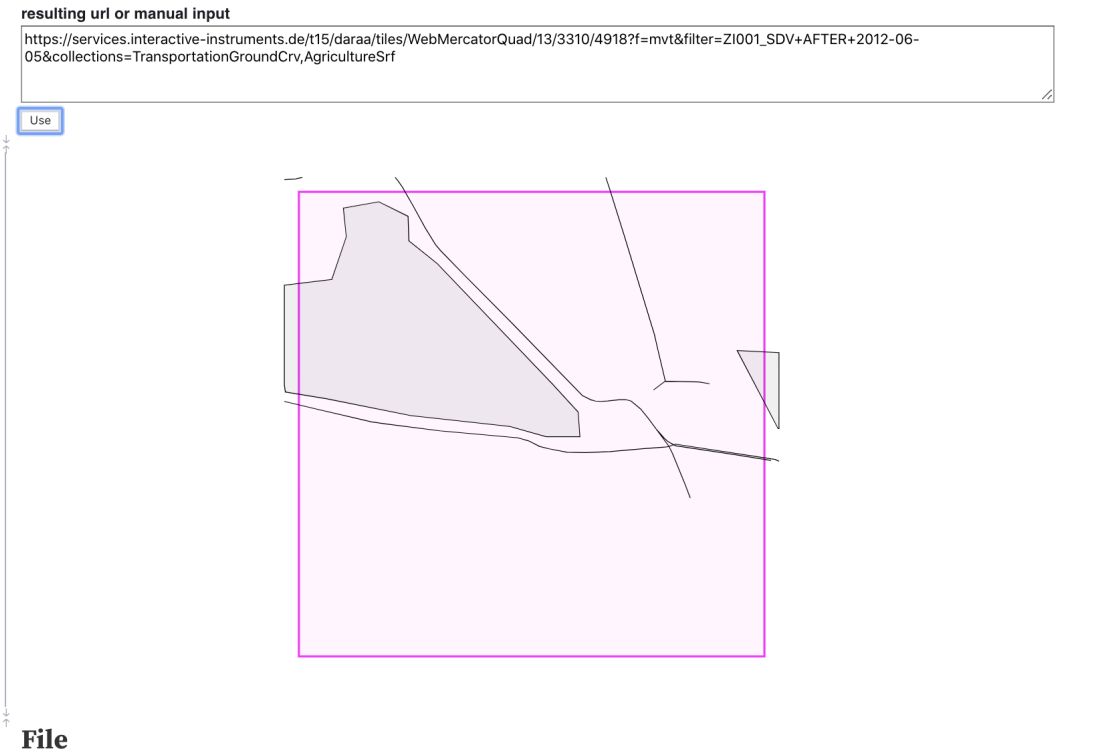
Features:	Request Link [https://services.interactive-instruments.de/t15/daraa/collections/TransportationGroundCrv/items?filter=ZI001_SDV+TEQUALS+2011-12-31T11:45:18Z]



In a filter on a multi-layer vector tile ldproxy applied the filter on each collection. This implied that the list of collections in the tile has to be restricted to a small set with shared queryables. Otherwise an error was returned since the predicate could not be evaluated for all layers/collections. This is discussed in more detail in [Results].

Fetch a multi-layer tile with Transportation (Ground) curves and Agricultural surfaces updated since 06/05/2012	
CQL:	`ZI001_SDV AFTER 2012-06-05`

Tile:	Request Link [https://services.interactive-instruments.de/t15/daraa/tiles/WebMercatorQuad/13/3310/4918?f=mvt&filter=ZI001_SDV+AFTER+2012-06-05&collections=TransportationGroundCrv,AgricultureSrf]



File

Property	Value	Evaluation	Comment
Size (uncompressed)	2 KiB (1121 Bytes)	💚	
Content-Type	application/vnd.mapbox-vector-tile	💚	
Number of Layers	2	💚	

Layer Details

#	name	features	points	#Keys	#Values	keys	keyStringLength	valueTypes	valueStringLength
1	TransportationGroundCrv	5	61	9	19	F_CODE, ZI001_SDV, ZI005_FNA, UFI, FCSUBTYPE, ZI006_MEM, RTY, RIN_ROI, ZI016_WTC	64	string 13 number 6	315
2	AgricultureSrf	2	23	4	5	F_CODE, ZI001_SDV, UFI, FCSUBTYPE	27	string 4 number 1	97

All the examples above used CQL in the Text variant. The following is a JSON example:

Fetch all Transportation (Ground) curves and Agricultural surfaces updated since 06/05/2012	
CQL:	`{"after":{"property":"ZI001_SDV","value":"2015-09-01"}}`

Features:	Request Link [https://services.interactive-instruments.de/t15/daraa/collections/TransportationGroundCrv/items?filter-lang=cql-json&filter=%7B%22after%22%3A%7B%22property%22%3A%22ZI001_SDV%22%2C%22value%22%3A%222015-09-01%22%7D%7D]

Home / Daraa / Data / Transportation - Ground (Curves) / Items GeoJSON

Transportation - Ground (Curves)

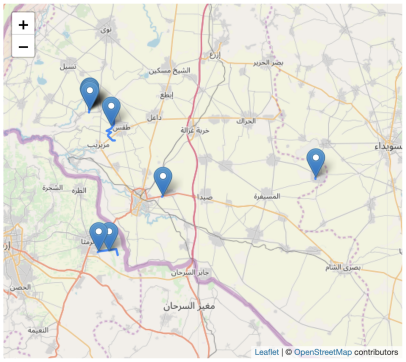
Transportation: Information about the principal means of overland movement of people and goods from one location to another.

Filter Edit

« ‹ 1 2 3 4 5 › »

No Information

id	75
Feature Type	Road
Last Change	09/10/2015, 14:56:40
Name	No Information
Roadway Type	Road
Route Designation Type	Local
Unique Entity Identifier	4ba385c0-844c-4d8d-a7b6-84242714c16c
Feature Subtype Code	100152
Memorandum	No Information
Source Description	No Information
Road Weather Restriction	All-weather
Route Designation	No Information
Relative Level	No Information
Vertical Relative Location	No Information



No Information

id	99

The API was also tested with the GeoSolutions client, where various filters were applied on the vector tiles of the Transportation (Ground) curves. The following are two screenshots of sample filters:

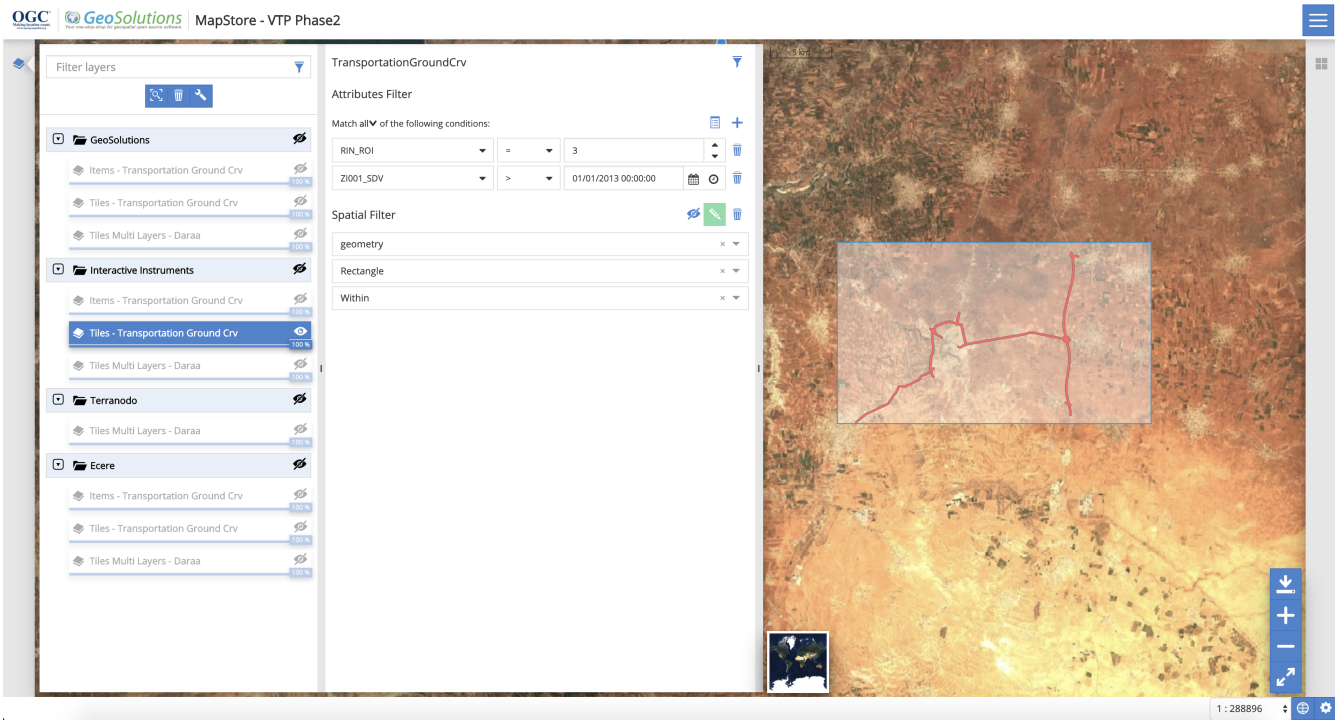


Figure 9. Filtering Transportation (Ground) curves using the GeoSolutions client (example 1)

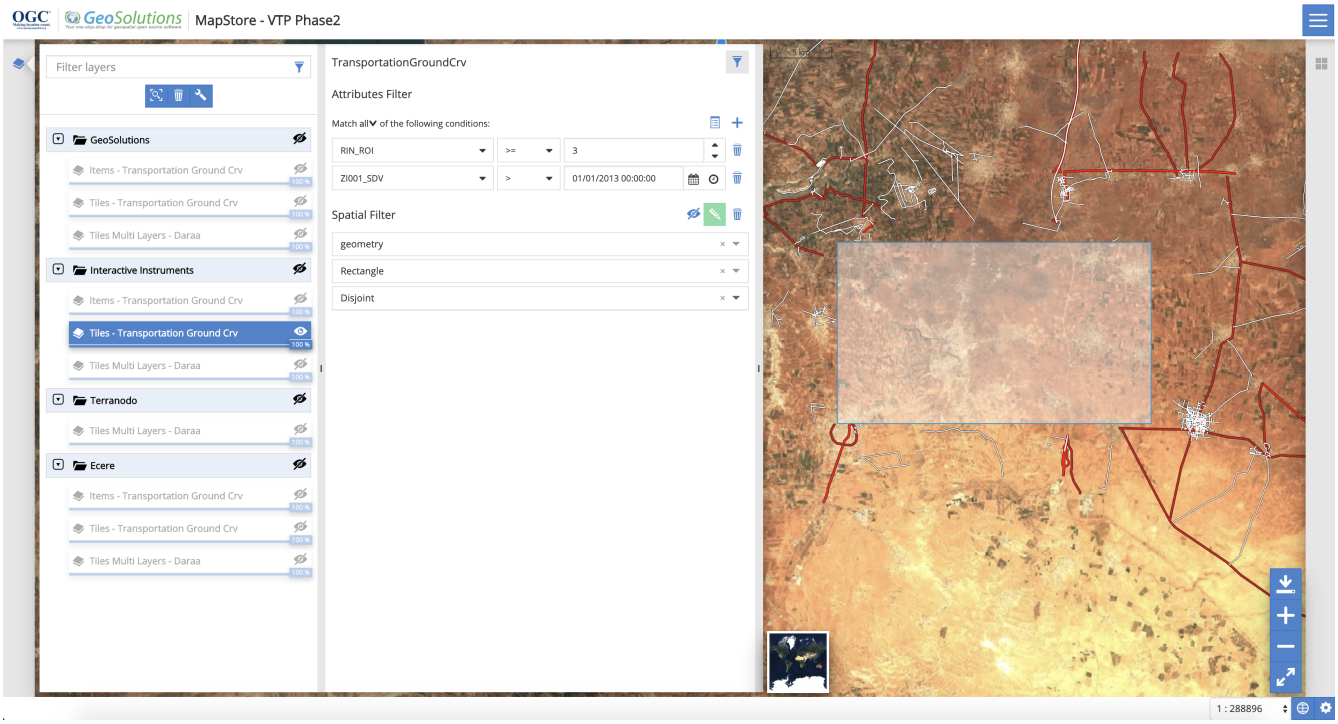


Figure 10. Filtering Transportation (Ground) curves using the GeoSolutions client (example 2)

11.4. GeoSolutions D102 OGC API - Tiles

GeoSolutions worked on a filtering extension for the existing GeoServer OGC API - Tiles. During the VTP2 activity, the OGC API - Tiles has been extended to advertise queryables for vector collections, as shown in the following screenshot.

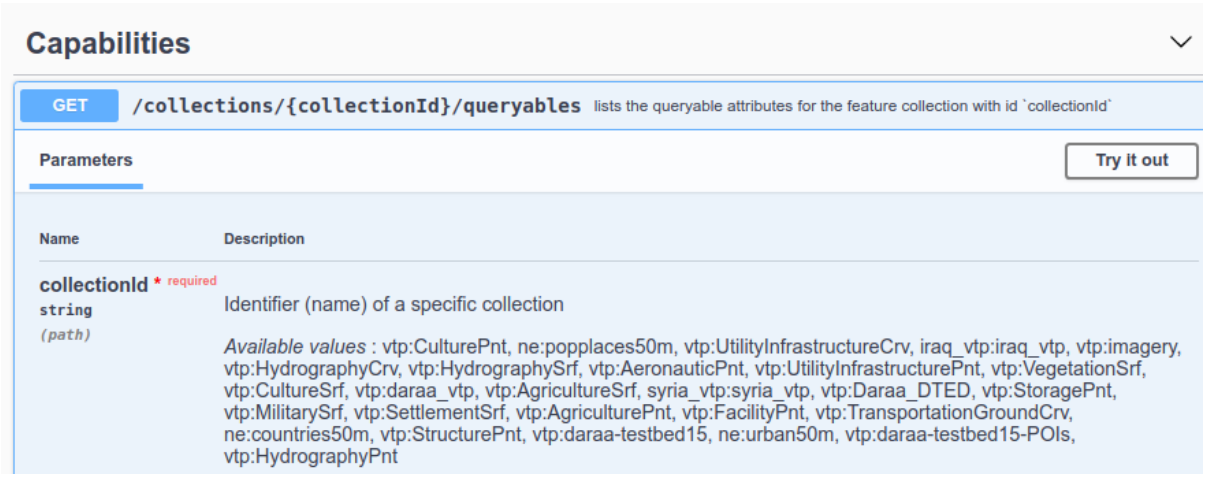


Figure 11. API has been extended with Queryables concept. The collectionId parameter lists only the names of collections actually exposing queryables.

Figure 12. HTML representation for single layer vector collection, with links to the queryables, as well as map and data tiles.

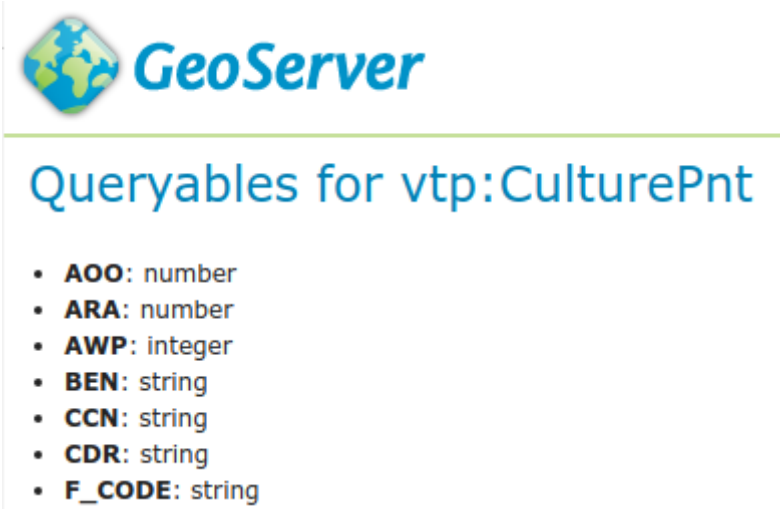


Figure 13. HTML representation for the above layer queryeables (excerpt, the layer in question has 73 queryables).

Then, the "data tiles" endpoint has been extended to allow filtering by means of the `filter` and `filter-lang` parameters. At the time of writing, the only allowed value for `filter-lang` is `cql-text`.

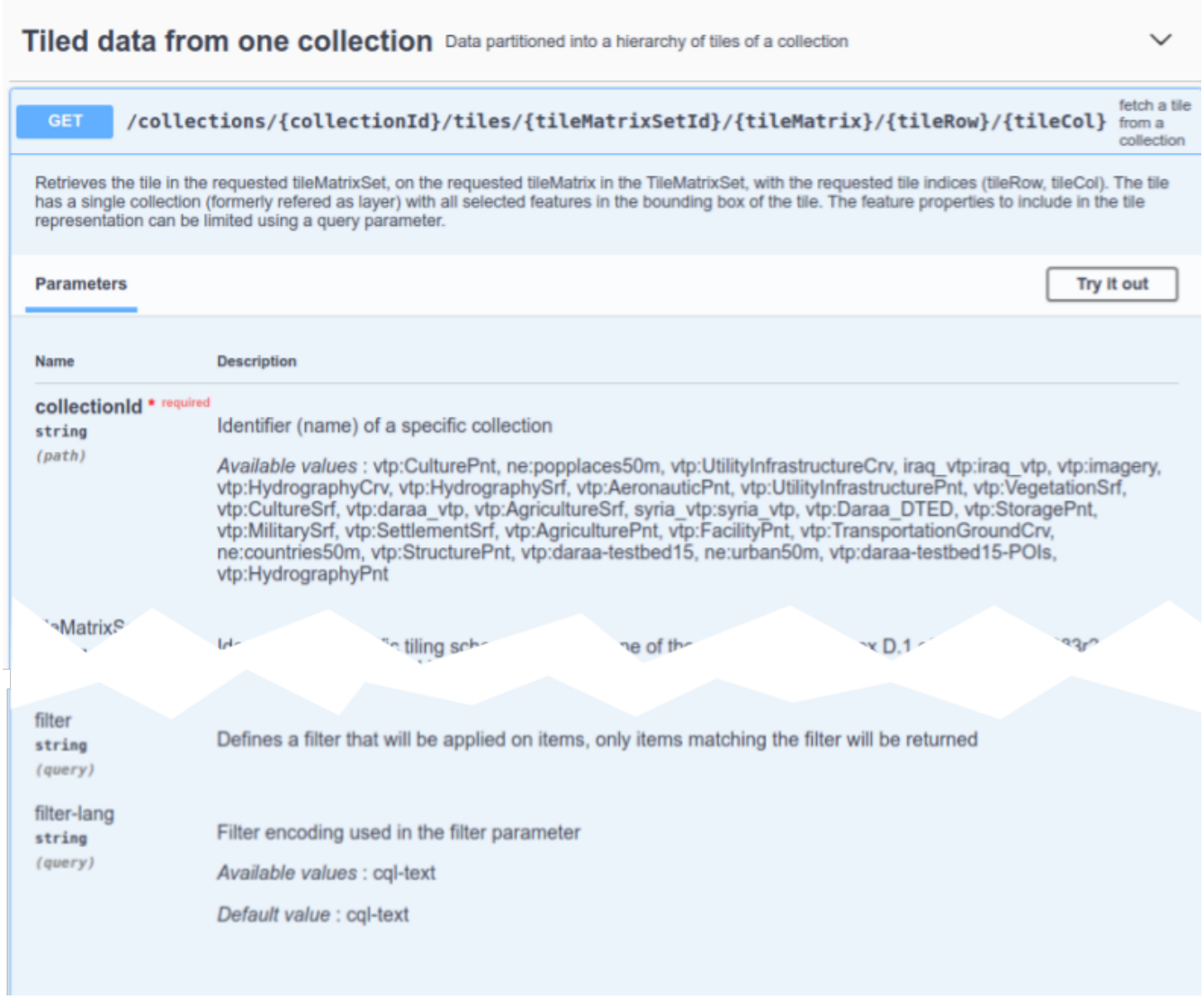


Figure 14. HTML representation for the above layer queryeables (excerpt, the layer in question has 73 queryables).

The implementation of `cql-text` is based on GeoServer Extended CQL [https://github.com/geotools/geotools/blob/master/modules/library/cql/ECQL.md], providing a full set of filtering operators, as well as a comprehensive list of filter [https://docs.geoserver.org/stable/en/user/filter/function_reference.html] functions covering basic math, string and date handling, as well as geometry manipulation.

A sample filtered tile request follows, reformatted and URL decoded for readability:

- `https://vtp2.geo-solutions.it/geoserver/ogc/tiles/` (service base)
- `collections/vtp:TransportationGroundCrv/tiles/WebMercatorQuad/14/6618/9833?` (single tile access)
- `filter=(RIN_ROI = '3')&` (the filter)
- `filter-lang=cql-text` (filter language is CQL)

Various tile filtering visual examples are available in the GeoSolutions D104 client section.

11.5. Ecere D103 Features & OGC API - Tiles

Ecere developed capabilities for server-side filtering in its GNOSIS Map Server, and exposed it in both the OGC API - Features and OGC API - Tiles implementations. The implementation included support for comparison of attribute values, geometry intersection with a bounding box, as well as logical and arithmetic operations. The current implementation supports expressions as defined in CMSS, the native styling language of GNOSIS tools. Support is planned for the CQL filtering language, but a parser for it could not be implemented during the pilot. Filtering Cascading Map Style Sheets (CMSS) was made available instead.

The CMSS styling language was first proposed as an encoding for a styling conceptual model in OGC Testbed 14, and documented in the Portrayal, and CityGML and Augmented Reality (http://docs.opengeospatial.org/per/18-025.html#_expressions) Engineering Reports. CMSS expressions serve both for selectors, deciding whether to apply symbology rules, and for styling property values within symbolizers. Just like with selectors, the expressions used for filtering resolve to a single boolean value, if the value is true then the feature is included in the results.

The syntax for CMSS expressions is mostly compatible with the eC programming language [http://ec-lang.org], a superset of the C language adding support for object orientation, as well as with ECON [http://ec-lang.org] (eC Object Notation). Like the bitwise C operator, the symbol `|` represents an OR logical operation, while `&` represents AND. However, textual 'or' and 'and' are also supported. Negation is represented by the `!` operator, and inequality by `!=`.

11.5.1. Examples of filtering expressions used with the OGC API - Features

From Natural Earth countries, return only features whose *name* property is either *Canada* or *United States*:

CMSS Expression: `name = 'Canada' | name = 'United States'`

Encoded filtered request URL:

https://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countries/items?filter-lang=cmss&filter=name=%27Canada%27|name=%27United%20States%27

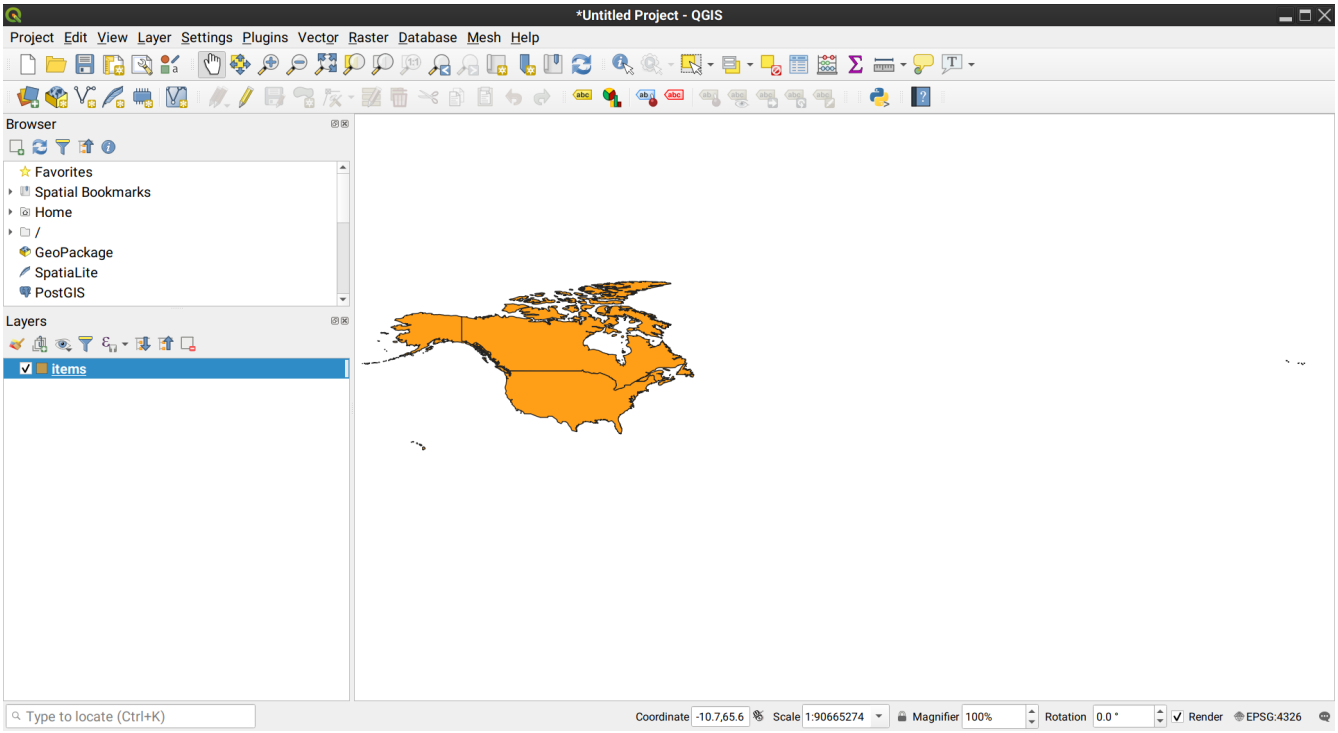


Figure 15. Filtering country features by name

From Daraa OpenStreetMap / Topographic Data Store's ground transportation linear features, return only those whose name property (*ZI005_FNA*) is not *No Information*:

CMSS Expression: `ZI005_FNA != 'No Information'`

Encoded filtered request URL:

https://maps.ecere.com/geoapi/collections/vtp/Daraa2/TransportationGroundCrv/items?filter-lang=cmss&filter=ZI005_FNA!=%27No%20Information%27

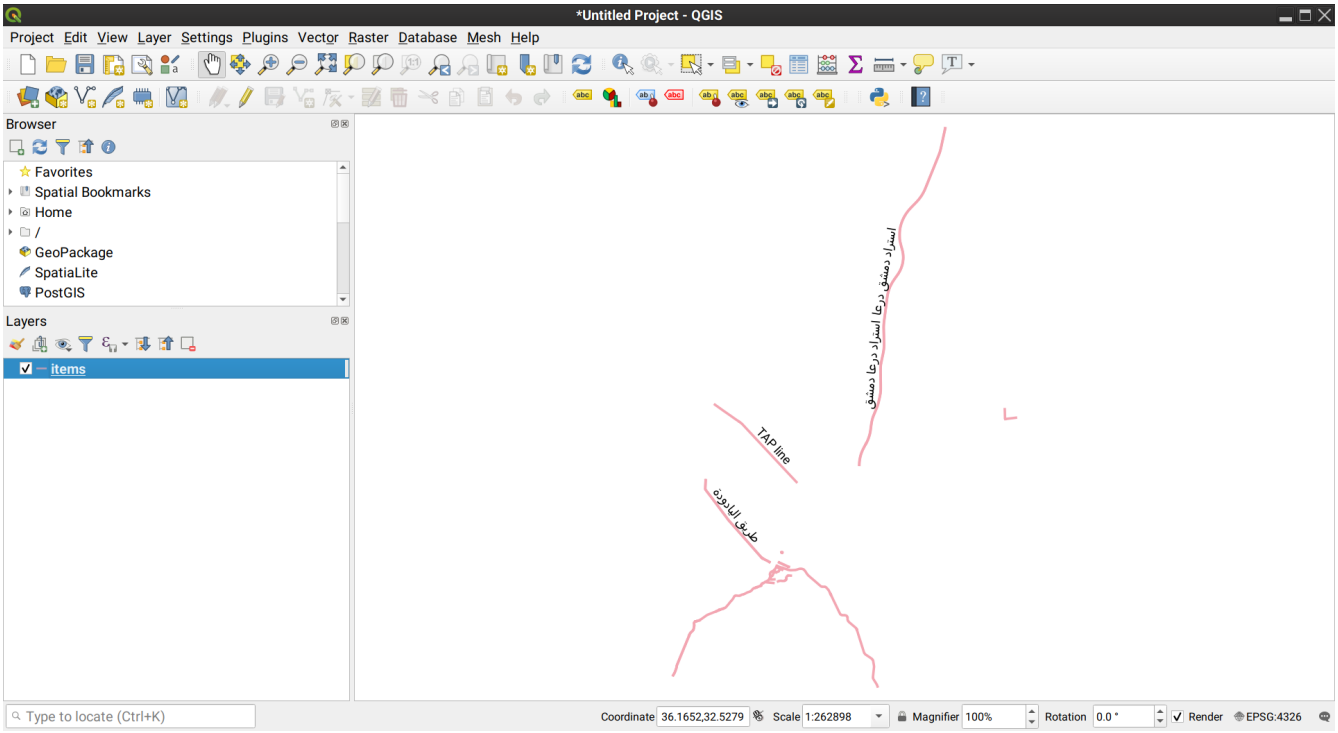


Figure 16. Filtering out road features with name set to 'No Information'

CMSS currently does not feature full regular expressions, but supports the text comparision operators: *Contains* (~), *Begins with* (^) and *Ends with* ($).

From Natural Earth countries, return only features whose *name* property contains *ain*:

CMSS Expression: `name ~ 'ain'`

Encoded filtered request URL:

https://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countries/items?filter-lang=cmss&filter=name~%27ain%27

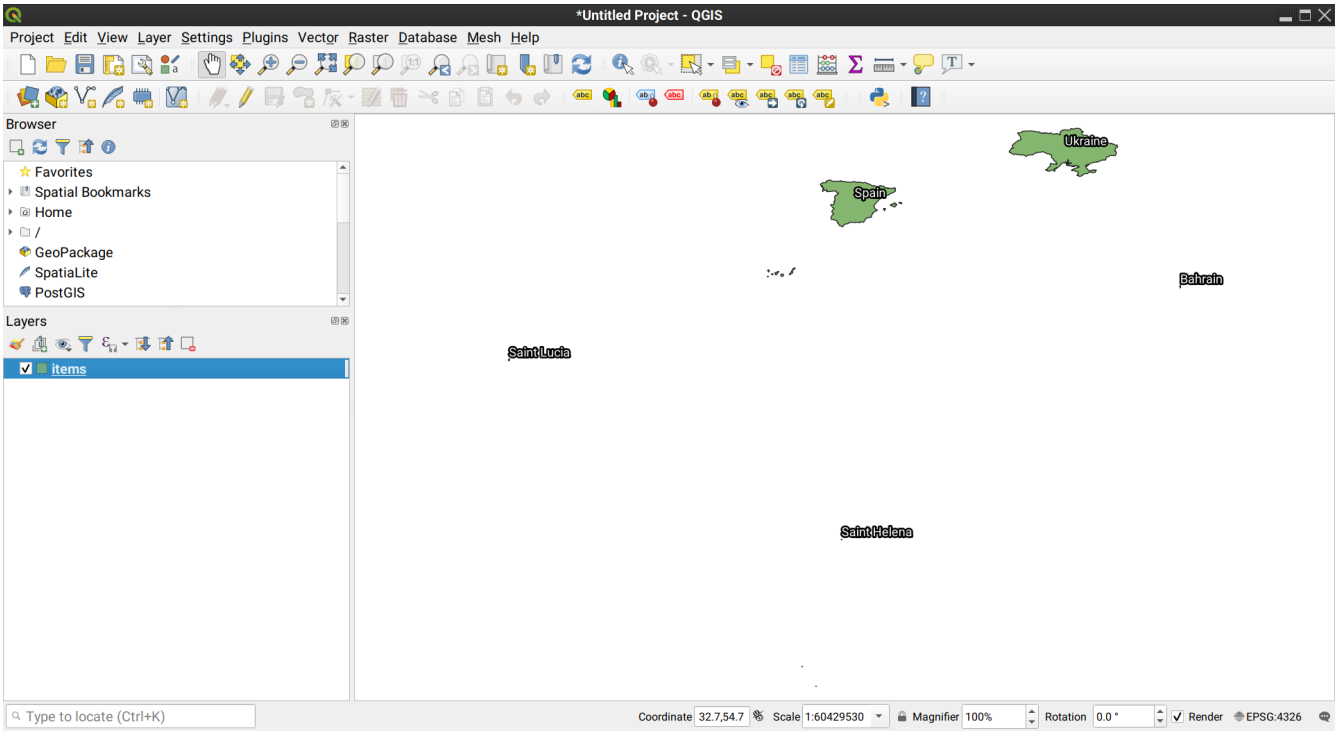


Figure 17. Filtering country features whose names contain a sub-string

From Natural Earth countries, return only features whose estimated population (*pop_est*) property is greater than 50 million:

CMSS Expression: `pop_est > 50M`

Encoded filtered request URL:

https://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countries/items?filter-lang=cmss&filter=pop_est%3E50M

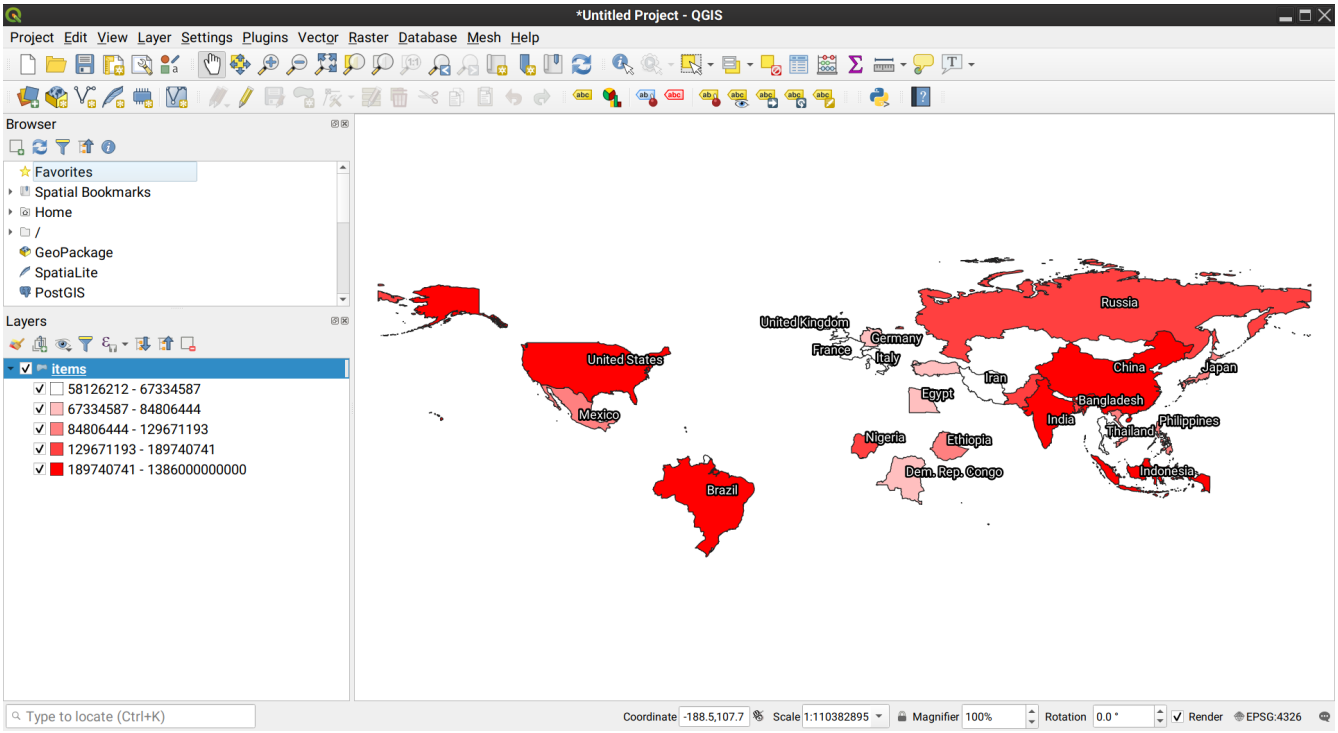


Figure 18. Filtering out country features below a population threshold

The 'intersects' function was implemented to check intersection with geometry.

Currently, only support for checking intersection between a record's geometry and a bounding box (specified as `{ { south latitude, west longitude }, { north latitude, east longitude } }`) has been implemented.

From Natural Earth countries, return only features whose *continent* property is *Anctarctica*, or whose geometry intersects with a bounding box around 45°N, 75°W:

CMSS Expression: `continent = 'Antarctica' | intersects(rec.geom, { { 45, -75 }, { 45.01, -74.99 } })`

Encoded filtered request URL:

https://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countries/items?filter-lang=cmss&filter=continent=%27Antarctica%27|intersects%28rec.geom,{{45,-75},{45.01,-74.99}}%29

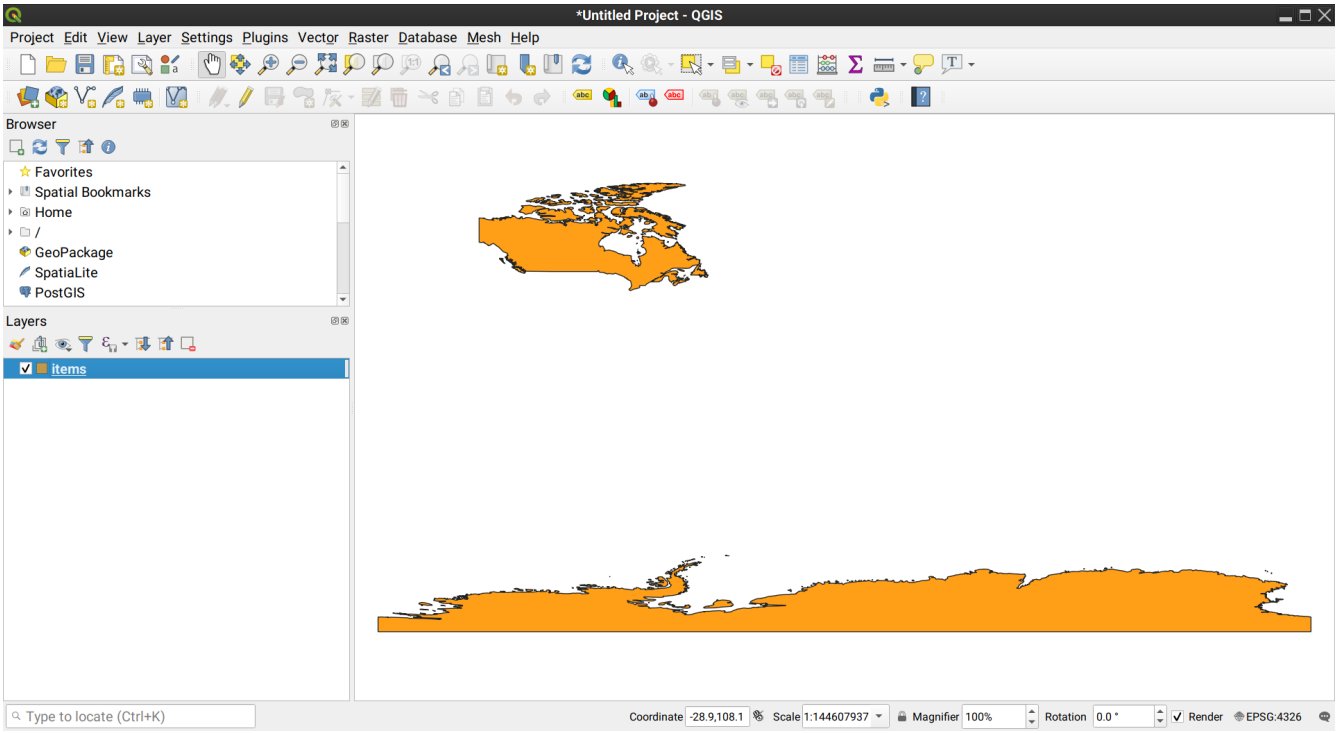


Figure 19. Filtering out features whose geometry lies outside of a bounding box

Support for `lyr.geom` to test whether any of the layers geometries intersect with each other, is also planned, as this might be useful for complex styling scenarios with special rules for overlapping geometry.

11.5.2. Use of filtering for vector tiles

Allowing clients requesting vector tiles to specify custom filters can reduce bandwidth and processing requirements, providing a flexible mechanism to make the same data suitable for different purposes and styles. Spatial operations may be less useful in conjunction with tiles, which already limit the response to a specific spatial extent, but are still available in the OGC API - Tiles for consistency.

Examples of filtering expressions used with the OGC API - Tiles

From Natural Earth countries, return a MapBox vector tile containing only features whose `name` property is either *Canada* or *United States*. Use the WebMercatorQuad tile matrix set, zoom level 0, row 0, column 0.

CMSS Expression: `name = 'Canada' | name = 'United States'`

Encoded filtered request URL:

https://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countries/tiles/WebMercatorQuad/0/0/0.mvt?filter-lang=cmss&filter=name=%27Canada%27|name=%27United%20States%27

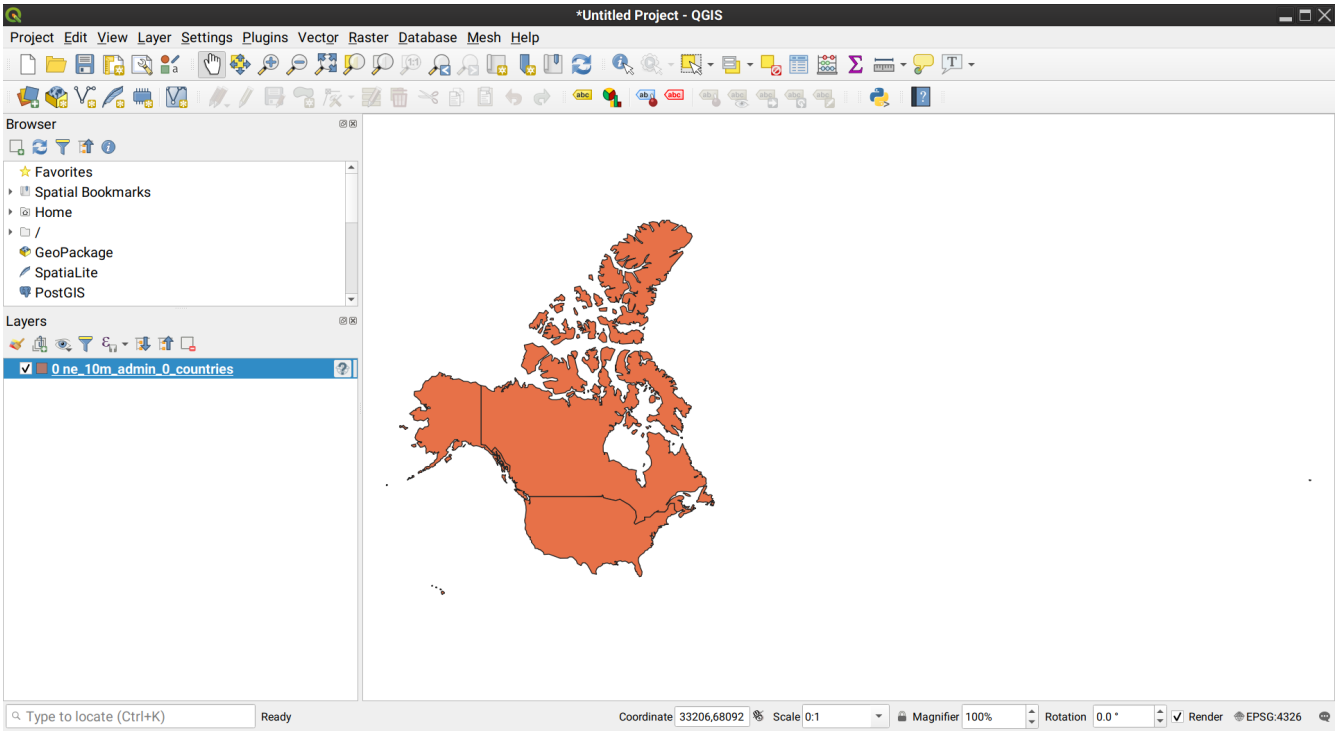


Figure 20. Filtering country features by name from a tile request

From Daraa OpenStreetMap / Topographic Data Store's ground transportation linear features, return only those whose name property `ZI005_FNA` is not *No Information*. Use the WebMercatorQuad tile matrix set, zoom level 9, row 206, column 307.

CMSS Expression: `ZI005_FNA != 'No Information'`

Encoded filtered request URL:

http://maps.ecere.com/geoapi/collections/vtp/Daraa2/TransportationGroundCrv/tiles/WebMercatorQuad/9/206/307.mvt?filter-lang=cmss&filter=ZI005_FNA!=%27No%20Information%27

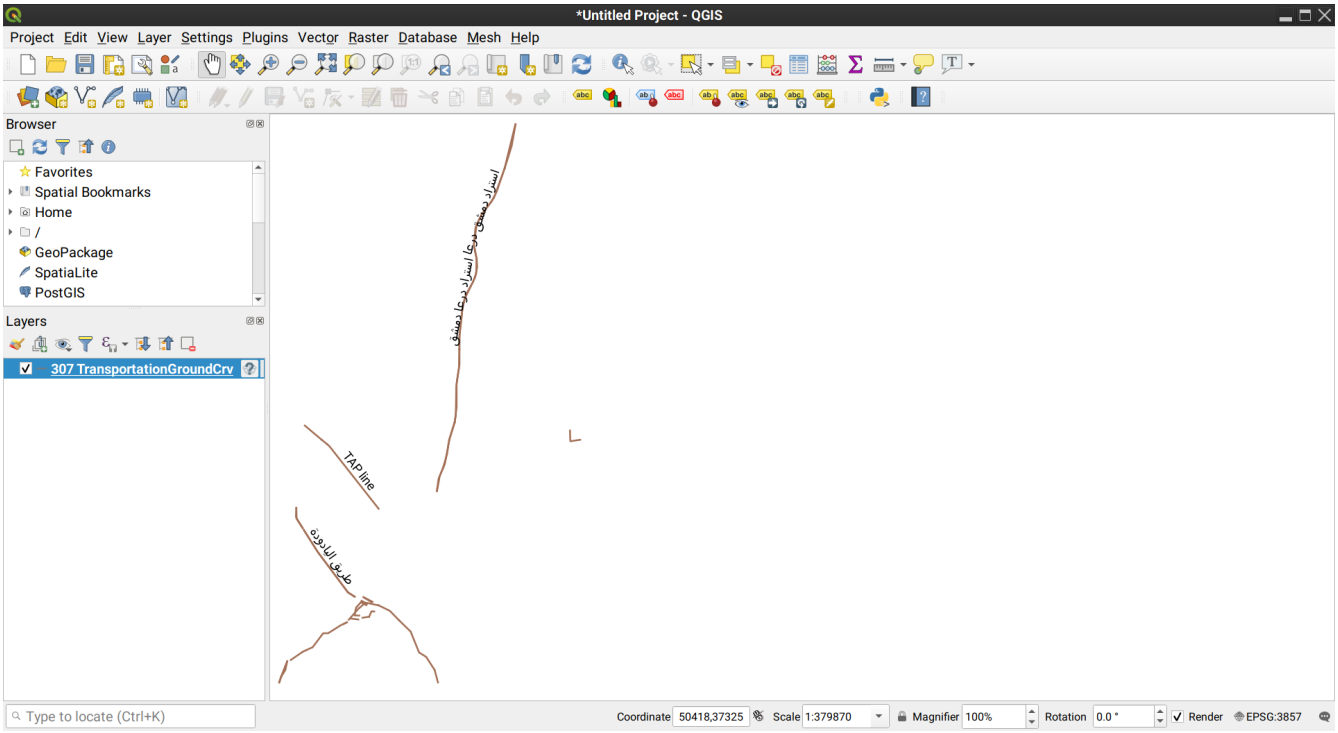


Figure 21. Filtering out road features with name set to 'No Information' from a tile request

11.5.3. Multi-layer and scale-based filtering (OGC API - Tiles)

The ability to select which layers to include inside multi-layer tiles also helps to reduce the size of produced tiles. CMSS already supported simple expressions' syntax e.g. as enclosed in a single selector within [ ] or being assigned to a property In addition, the syntax has been extended to include one or more selectors enclosed within square brackets, as well as a layer identifier selector prefixed by the # symbol. This syntax mirrors the full CMSS syntax as used for styling, but excludes the symbolizer and nested rules enclosed in curly braces which normally follow these selectors. Alternatively, the pre-defiend `lyr.id` expression can be compared with a string literal, and used in conjunction with an `or`operation to match multiple layers. Additional selectors can be combined with `and` operations, with operands contained within parentheses.

Multi-layer filtering is particularly useful in conjunction with a scale-based selector, making it possible to re-use the same tiles template for all zoom levels.

CMSS pre-defines the `viz.sd` expression to refer to the scale denominator. In the context of the OGC API - Tiles filtering, it represents the scale denominator associated with a specific zoom level (tile matrix) being requested. The scale denominators associated with each zoom level of a tile matrix set are well-defined by the OGC Tile Matrix Standard, and tile matrix sets defined following this specification, according to a standard 0.28 mm/pixel equivalence. Referencing a scale denominator rather than a zoom level ensures that a filter or style is re-usable between different tile matrix sets where zoom levels correspond to different scales.

Examples of multi-layer and scale-based filtering

From Daraa OpenStreetMap / Topographic Data Store, retrieve all features of the agriculture polygons layer (AgricultureSrf) and of the settlement polygons layer (SettlementSrf). Use the WebMercatorQuad tile matrix set, zoom level 8, row 103, column 153.

CMSS Expression (using selectors syntax): `#AgricultureSrf#SettlementSrf`

Encoded filtered request URL:

https://maps.ecere.com/geoapi/collections/vtp/Daraa2/tiles/WebMercatorQuad/8/103/153.mvt?filter-lang=cmss&filter=%23AgricultureSrf%23SettlementSrf

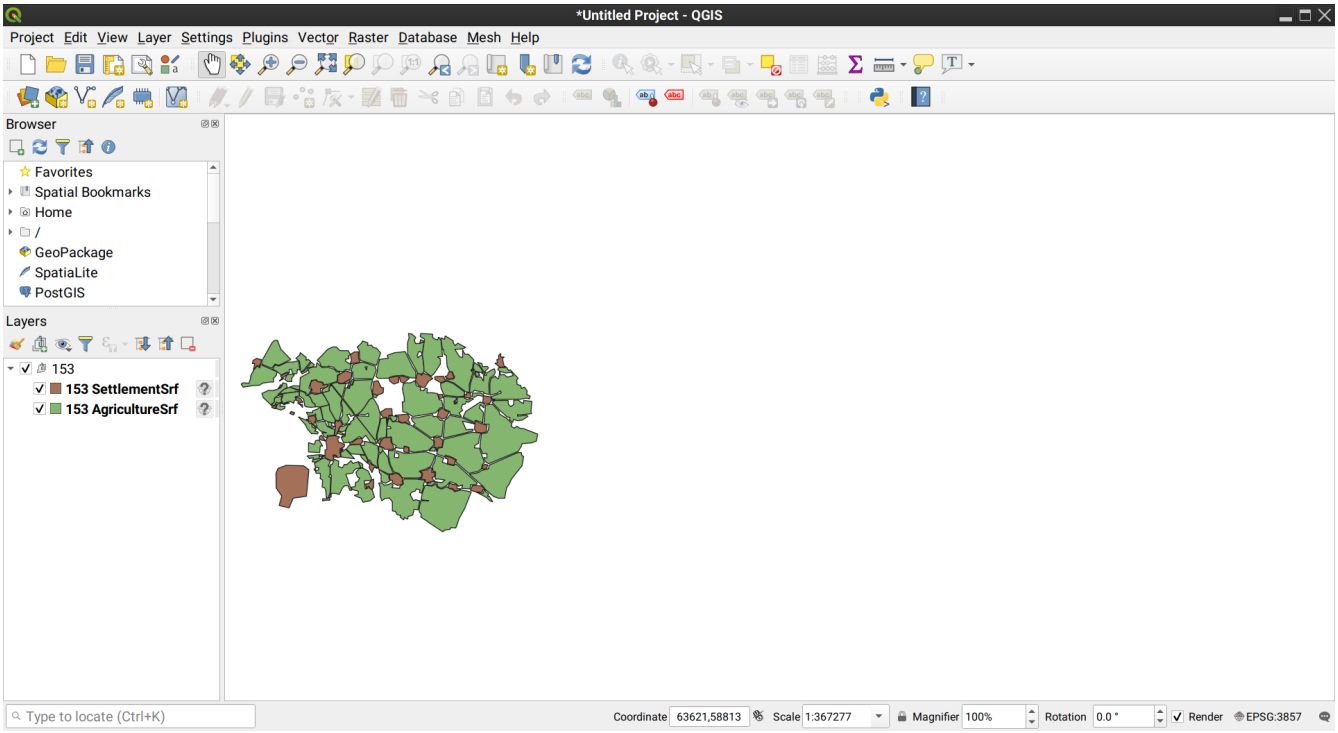


Figure 22. Multi-layer filter to only include specific layers

---

From Daraa OpenStreetMap / Topographic Data Store, retrieve:

- All features of the agriculture polygons layer (AgricultureSrf).
- All features of the settlement polygons layer (SettlementSrf).
- All features of the ground transportation linear features whose name property `ZI005_FNA` is not *No Information*.
- All features of the culture points layer (CulturePnt), but only for zoom levels more detailed than 1:2,000,000 scale.

Use the WebMercatorQuad tile matrix set, zoom level '0', row 206, column 307.

CMSS Expression (using selectors syntax):

```
#AgricultureSrf #SettlementSrf #TransportationGroundCrv[ZI005_FNA != 'No Information']
#CulturePnt[viz.sd < 2M]`
```

Encoded filtered request URL:

https://maps.ecere.com/geoapi/collections/vtp/Daraa2/tiles/WebMercatorQuad/9/206/307.mvt?filter-lang=cmss&

---

filter=%23AgricultureSrf%23SettlementSrf%23TransportationGroundCrv%5BZI005_FNA!=%27No%20Information%27%5D%23CulturePnt%5Bviz.sd%3C2M%5D

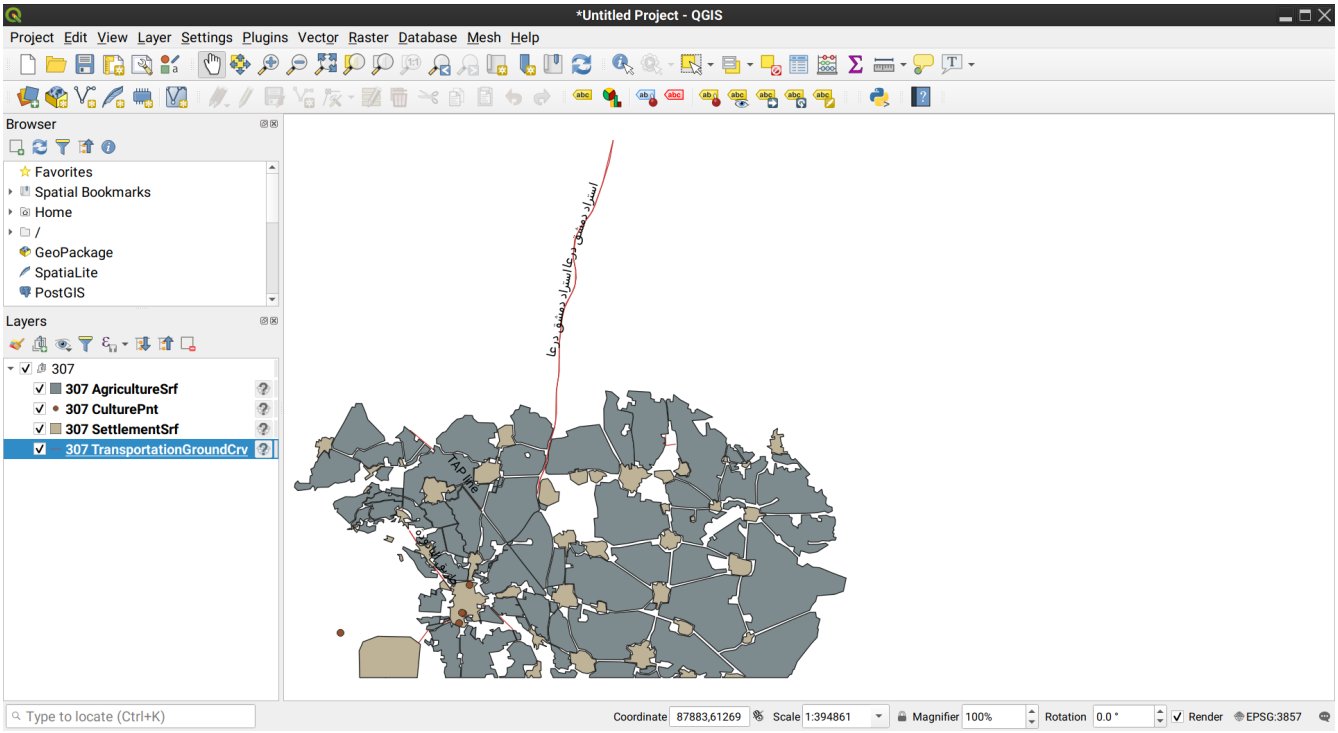


Figure 23. Complex multi-layer filter combining layer selection, name attribute condition and scale-based filtering

If the same filter is used for WebMercatorQuad zoom level 8 (scale is 1:2,183,915), the culture points, which were included at level 9, will be omitted:

https://maps.ecere.com/geoapi/collections/vtp/Daraa2/tiles/WebMercatorQuad/8/103/153.mvt?filter-lang=cmss&
filter=%23AgricultureSrf%23SettlementSrf%23TransportationGroundCrv%5BZI005_FNA!=%27No%20Information%27%5D%23CulturePnt%5Bviz.sd%3C2M%5D

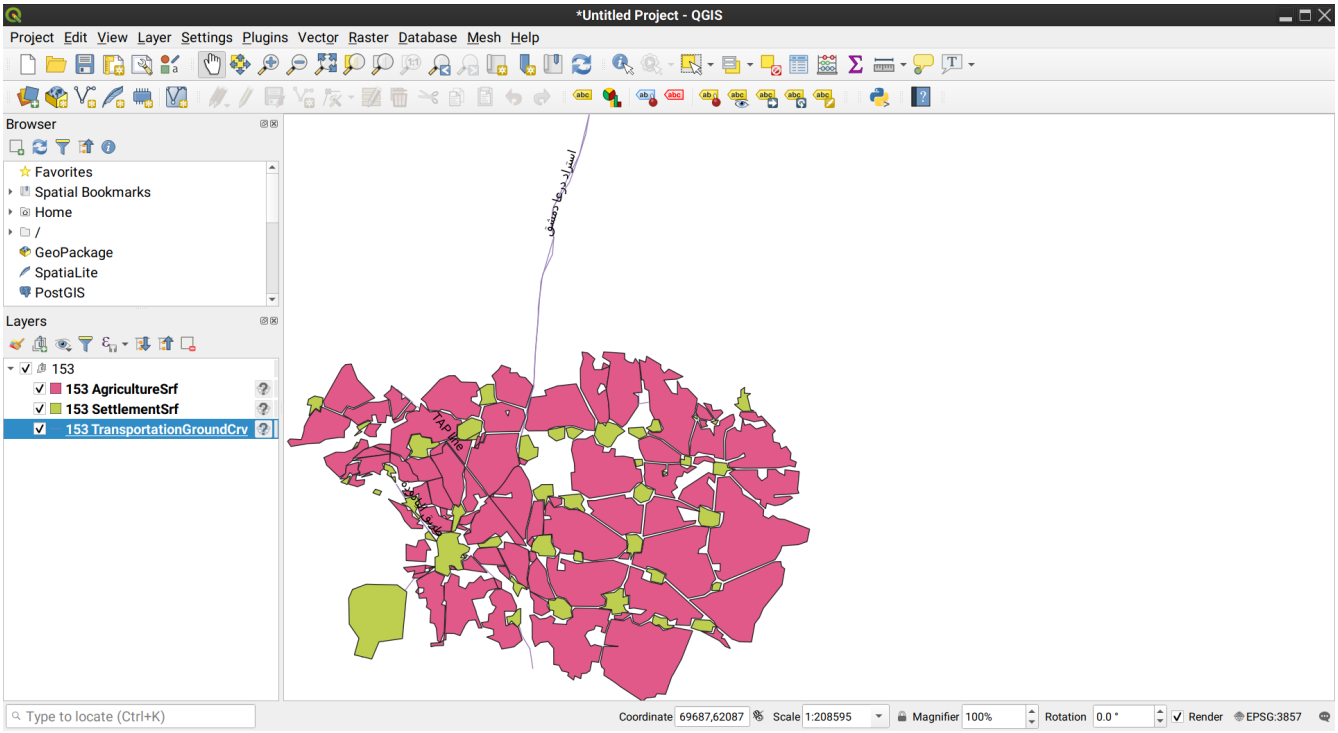


Figure 24. Identical multi-layer filter used at a different zoom level, showing the culture point features omitted

An equivalent expression not using the selectors syntax could be written as:

CMSS Expression (not using selectors):

```
( lyr.id = 'AgricultureSrf' or lyr.id = 'SettlementSrf' or (lyr.id =
'TransportationGroundCrv' and ZI005_FNA != 'No Information') or (lyr.id = 'CulturePnt'
and viz.sd < 2M) )`
```

Encoded filtered request URL:

http://maps.ecere.com/geoapi/collections/vtp/Daraa2/tiles/WebMercatorQuad/9/206/307.mvt?filter-lang=cmss&
filter=%28lyr.id=%27AgricultureSrf%27%20or%20lyr.id=%27SettlementSrf%27%20or%20%28lyr.id=%27TransportationGroundCrv%27%20and%20ZI005_FNA!=%27No%20Information%27%29%20or%20%28lyr.id=%27CulturePnt%27%20and%20viz.sd%3C2M%29%29

From OpenStreetMap dataset of Washington, D.C., return all features from natural, waterways, railways, landuse, other areas, other lines, boundaries, roads, buildings and amenities layers. Use the WorldMercatorWGS84Quad tile matrix set, zoom level 10, row 392, column 293.

CMSS Expression:

```
#natural #waterways #railways #landuse #otherAreas #otherLines #boundaries #roads
#buildings #amenities
```

Encoded filtered request URL:

https://maps.ecere.com/geoapi/collections/osm/dc/tiles/WorldMercatorWGS84Quad/10/392/293.mvt?filter-lang-cmss&filter=%23natural%23waterways%23railways%23landuse%23otherAreas%23otherLines%23boundaries%23roads%23buildings%23amenities

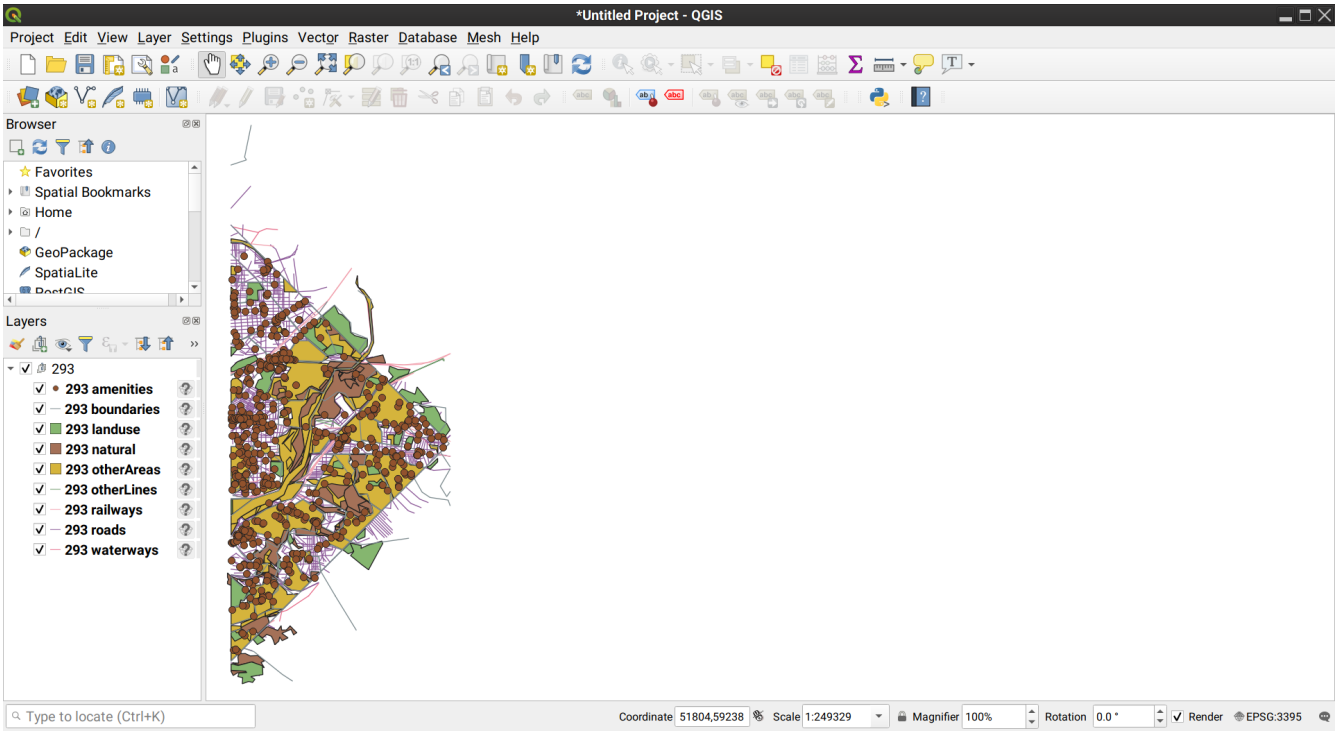


Figure 25. An OpenStreetMap tile requested with a filter to omit layers with large number of points

As a comparison, the same tile without a filter would include an extra 7000 points for the house numbers layer and 3000 for other points, each point containing attribute informations, resulting in a particularly big tile for that scale:

https://maps.ecere.com/geoapi/collections/osm/dc/tiles/WorldMercatorWGS84Quad/10/392/293.mvt

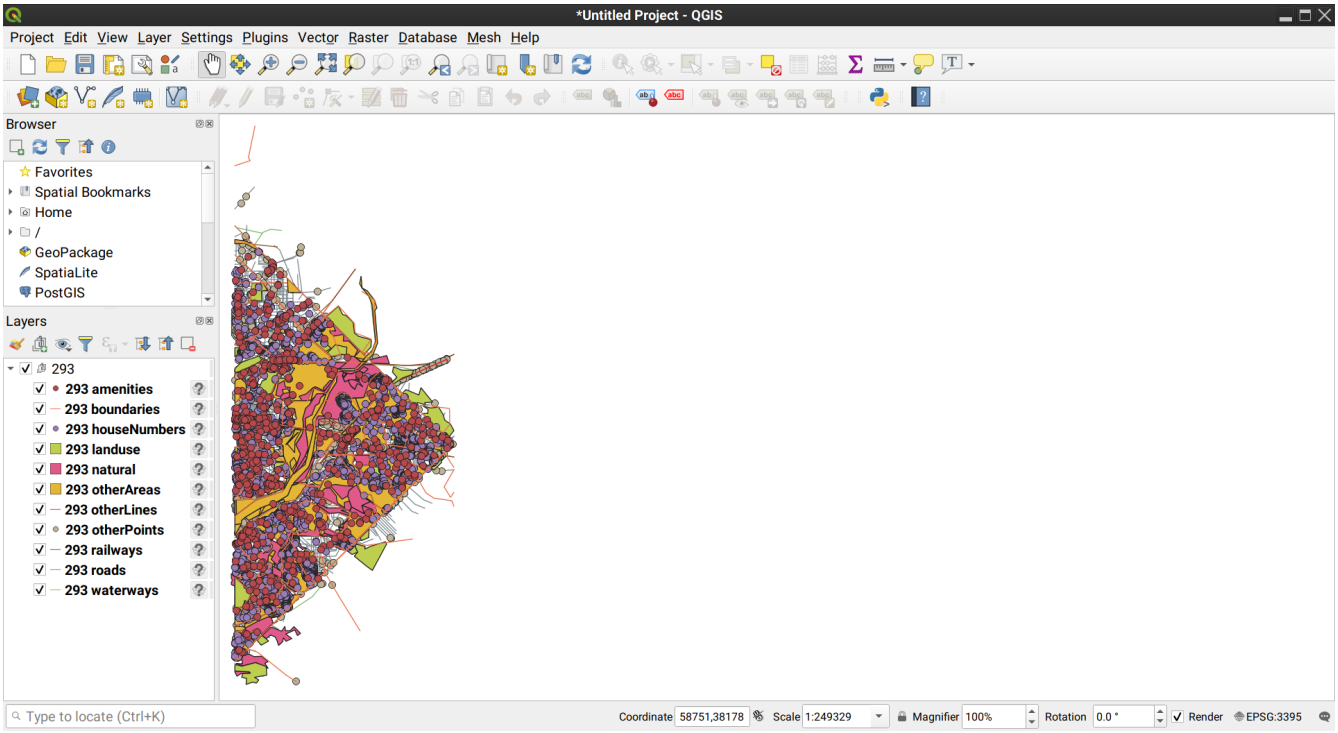


Figure 26. The same OpenStreetMap tile requested without the filter, showing the large number of points

11.5.4. POST requests to the OGC API - Tiles

A client would often make numerous requests for tiles using the same filter, especially when using multi-layer tiles and scale-based filtering. These filters may also be complex, which might pose problems to encode them entirely as a URL query parameter. As an alternative, Ecere considered using a POST request to a OGC API - Tiles end-point, whose content would be the filter expression. The response to this request would be a templated URL corresponding to the filter, which would not be a public resource, but potentially only available to a client having made that request. This would allow the server to only parse and process the filtering expression once, and may facilitate the configuration and/or caching of the tiles. A URL template returned this way might expire after a certain time of no tile request coming in, at which point the client would need to re-submit the template again to obtain a new valid template. Such a POST request could also be used with the OGC API - Tiles to specify complex multi-layer input data layers, styles, and even daisy-chained processing workflows. This approach was discussed and presented at the January 2020 OGC Coverage & Analytics Code Sprint.

As an example, a client could potentially submit a POST request to this URL (or an alternative version including a more complex filter in the payload):

http://maps.ecere.com/geoapi/collections/vtp/Daraa2/tiles?filter-lang=cmss&filter=%23TransportationGroundCrv%5BZI005_FNA!=%20No%20Information'%5D%23AgricultureSrf%23SettlementSrf%23CulturePnt%5Bviz.sd%3C2M%5D

and the server would return a templated link which may look like:

http://maps.ecere.com/geoapi/collections/vtp/Daraa2/tiles/{tileMatrixSet}/{tileMatrix}/{tileRow}/{tileCol}?savedConfig=a123b456c890ad123

When a client accesses a tile using that template, the requested filter would be included in the tile

result calculation.

11.5.5. Filtering driven by styles

For visualization use cases, filters may be tightly coupled with the styles displaying the data. Thus, the server could automatically determine the proper filter from the referenced style (from an associated Styles API), or by the client posting a complete style (e.g. encoded as CMSS, SLD/SE or Mapbox GL). In order to turn an encoded style into a filter, the service would filter out any geometry not being drawn. For example, in the case of CMSS (where everything is drawn by default, even without a style), the decision would be based on properties such as visibility, opacity, and label/marker. The server could also, optionally, omit any attribute information not useful for that style.

11.5.6. Retrieving attributes separately from geometry

In the GNOSIS Map Tiles encoding of vector tiles, attributes are not included in the result (in the GNOSIS Data Store, they are stored separately in a SQLite file, along with a spatial index). Currently, when the GNOSIS Client requests tiles as GNOSIS Map Tiles from the GNOSIS Map Server, the attributes are requested on demand separately, for the specific properties and features required. The request uses the query parameters `featureid` to specify which features to include, and `properties` to specify which properties to include.

In addition to being useful for vector tiles encodings formats not defining attribute, requesting the attributes separately can greatly reduce the size of the data to transmit, especially for large detailed features, spread across multiple tiles, having numerous associated attributes.

11.5.7. CMSS Expressions Syntax

A summary of the CMSS expression syntax is presented here. See the CityGML and Augmented Reality Engineering Report [http://docs.opengeospatial.org/per/18-025.html#_expressions] for a more detailed overview of CMSS as a styling language.

Types of expressions

type	example
identifier	FEATCODE
text	'Parking'
integer	10
real	3.14159
object	{ hour = 16, minutes = 30 } Circle { radius = 5 }
member	viz.sd
list	[1, 2, 3]
operation	viz.sd > 10M
variable	@colorScheme

Types of identifiers

id or example	description
F_CODE	Example of a data attribute
null	Unset value
true	Example of an enumeration value
lyr	The data layer (collection)
lyr.id	Identifier for the layer
lyr.fc	Feature class (e.g. vector, coverage, imagery)
lyr.vt	Vector type (e.g. points, lines, polygons)
lyr.geom	Layer geometry
viz	Visualization attributes
viz.sd	Scale denominator
viz.time	Visualization time
viz.date	Visualization date
viz.timeOfDay	Visualization time of day
rec	The record
rec.id	Record id
rec.geom	Record geometry
records	List of all records in the layer

Operators

Logical	AND (&), OR (|), NOT (!)
Comparison	Equal (=), Not equal (!=), Greater (>), Lesser than (<), Greater or Equal (>=), Lesser or Equal (\<\=)
Text Comparison	Contains (~), Starts with (^), Ends with ($), does not contain (!~), does not start with (!^), does not end with (!$)
Arithmetic	Addition (+), Subtraction (-), Multiplication (*), Division (/), Integer division (div), Remainder (%)
Priority	parentheses () for prioritizing
Conditional	if ? then : else
in	'in' to check if left-side expression is within a list
Function call	()

Functions

Text manipulation	strlwr, strupr, format, subst	implemented
Geometry operation	area, length, centroid	not yet implemented

Spatial operations	intersects, contains, within, withinRadius…	only intersects(rec.geom, bbox) implemented
Iteration	to iterate within a list as part of an expression — thought of as an exercise for rules based on overlapping features e.g. [featcode = 'bay' & iterate(or, records, it.featcode = 'ocean' & intersects(it, rec.geom))]	not yet implemented

For temporal support, time attributes can be compared against either visualization attributes (e.g. viz.time) or fixed time values.

11.6. Ecere D105 Client

Ecere provided a client able to visualize tiled vector features served by OGC API - Tiles, based on its GNOSIS Cartographer application. The client can apply styles, whether loaded locally, created using the visual style editing interface, or accessed via the Styles API. Features will be automatically filtered based on the style in use, but this is still currently being done on the client side. As discussed in the server section, a filter can automatically be determined from a style, by considering its selectors and visualization properties such as opacity and visibility. The client could either reference a style, or POST a style to the service to specifically ask for pre-filtered geometry and attributes based on that style. Alternatively, the client could itself have determined that filter suitable for the style in use and submit that filter in its requests to the service. That capability was planned but was not yet implemented during the pilot. A disadvantage of requesting pre-filtered tiles, rather than doing the filtering on the client side, is that the client would need to discard tiles and retrieve a different version when changing between styles.

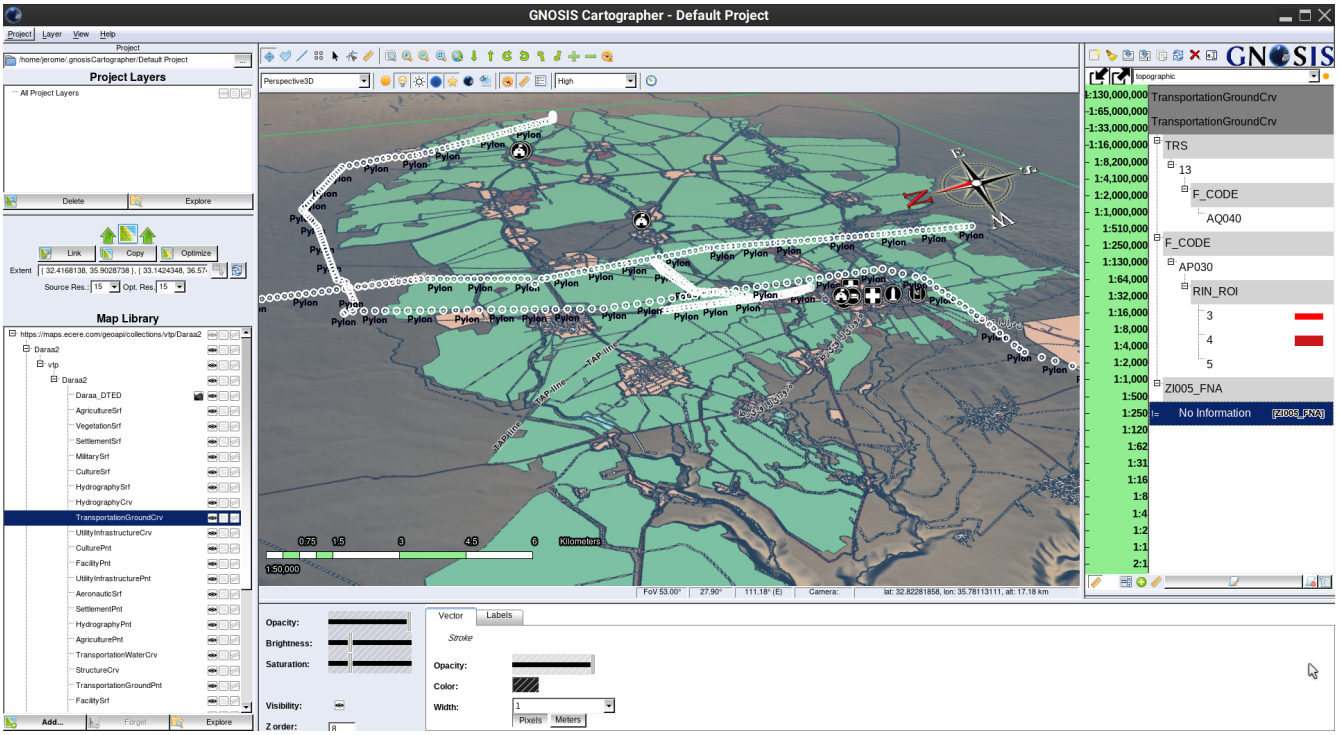


Figure 27. A screenshot of GNOSIS Cartographer showcasing the styling rules which could be used to automatically determine a filter

11.7. Ecere D107 GeoPackage producer & client

Ecere produced a number of GeoPackages for the pilot, as well as a client capable of visualizing GeoPackages produced by other participants. These GeoPackages produced using Ecere's GNOSIS Cartographer tool made use of the "Attributes Tables" extension, which is also supported by the visualization client. This extension is useful for performing filtering when the client accesses the tile data (e.g. accessing fewer encoded tile blobs based on SQL queries). This optimization was not yet implemented in the client during the pilot. The attributes table extension also greatly reduced the size of GeoPackages compared to embedding attributes within the tiles themselves, and provided fast access to querying or updating the attribute information for a given feature. A filter could also be applied when generating a GeoPackage to minimize its size, but this capability has not yet been implemented either. See the GeoPackage section for more details, and an explanation of the SQL queries pictured below.

```
sqlite> select attributes_TransportationGroundCrv.id, UFI, F_CODE, FCSUBTYPE,  ZI005_FNA
from attributes_TransportationGroundCrv inner join rtree_attributes_TransportationGroundC
rv_vector_tiles on attributes_TransportationGroundCrv.id = rtree_attributes_Transportatio
nGroundCrv_vector_tiles.id where ZI005_FNA != 'No Information' and maxLat > 326083233 and
 maxLon > 360899582 and minLat < 326097047 and minLon < 360987994;
id          UFI                                   F_CODE      FCSUBTYPE   ZI005_FNA
----------  ------------------------------------  ----------  ----------  ----------
1442        36d6beb3-0a35-4ec3-a2f6-6b526ea1f659  AP030       100152      شارع الأردن
3178        204af963-3d18-402d-8cfe-ff4195e992c7  AP030       100152      شارع الزيمي
sqlite> select base_id, zoom_level, tile_row, tile_column from mapping_table_Transportati
onGroundCrv inner join tiles_Daraa2 on mapping_table_TransportationGroundCrv.base_id = ti
les_Daraa2.id inner join attributes_TransportationGroundCrv on mapping_table_Transportati
onGroundCrv.related_id = attributes_TransportationGroundCrv.id inner join rtree_attribute
s_TransportationGroundCrv_vector_tiles on attributes_TransportationGroundCrv.id = rtree_a
ttributes_TransportationGroundCrv_vector_tiles.id where ZI005_FNA != 'No Information' and
 maxLat > 326083233 and maxLon > 360899582 and minLat < 326097047 and minLon < 360987994
and zoom_level = 16;
base_id     zoom_level  tile_row    tile_column
----------  ----------  ----------  -----------
7962        16          26518       39339
8059        16          26519       39338
8060        16          26519       39339
8161        16          26520       39337
8162        16          26520       39338
8162        16          26520       39338
8163        16          26520       39339
sqlite>
```

Figure 28. SQL queries performed at an SQLite prompt inside a multi-layer WorldMercator GeoPackage produced by Ecere, which could be used for filtering

11.8. Skymantics D104 Client

Skymantics selected Unity 3D to develop a map client with augmented reality capabilities. This software was combined with Mapbox SDK and Mapbox Studio to provide a framework for map utilization.

11.8.1. Use of CQL Unsuccessful

The implementation of CQL filters within these software applications proved unsuccessful for the following reasons:

- Unity 3D allows the creation of C# scripts that could be used to command filtering and styling over maps generated on Unity 3D. When generating maps with Mapbox SDK and Mapbox Studio, the actual rendering of the map is encapsulated by Mapbox SDK and therefore the Unity C# scripts have no power over the maps.
- Mapbox SDK does not allow Unity to interact with the filtering and styling tools found on Mapbox Studio. All those tools must be used manually in the Mapbox Studio application.
- Vector tile data can only be uploaded to Mapbox Studio as a dataset or a tile set, therefore discarding any possibility of CQL filtering in Mapbox Studio.
- Tile sets were fetched from the tile servers before being stored in files, and only afterwards loaded into Mapbox Studio. A CQL filter could have indeed been added to the GET requests. This approach would have been impractical for end users, because it was impossible to initiate from Unity 3D the automatic fetch of tile sets, the upload of those tile sets into Mapbox Studio, and the render in Unity of the updated filtered map.

11.8.2. Filtering Without CQL

In order to provide filtering support into Unity and Mapbox, an alternative approach was implemented. Skymantics generated styles with pre-defined filters in Mapbox Studio, and then made those styles available in Unity.

Mapbox provides their own filtering tools within the Mapbox Studio environment. In the Mapbox Style Specification, a filter is a property at the layer level that determines which features should be rendered in a style layer. Filters are written as expressions, providing fine-grained control over which features to include: the style layer only displays the features that match the filter condition that the style defines.

Mapbox Studio allows users to create styles by adding multiple features to them. Entire tile sets, comprising one or several layers, can be added to a style, therefore creating a Mapbox style made up of a single or multiple layers. As shown on Figure 29, an Agriculture tile set and a Hydrography tile set were loaded into a single style as two separate layers; Agriculture was defined by the color green and Hydrography as dark blue.

Mapbpox Studio features a variety of filtering and styling tools. Once the layers have been added to a style, these tools can be applied to the style. In Figure 29, a filter based on values is being applied to a style in Mapbox Studio.

The final step of the process is to load the style into Unity. Mapbox Studio makes the styles created by the user available through a URL that is called by the Mapbox SDK in Unity. As seen in Figure 31, the URL is loaded into Unity and the style is automatically rendered, as previously visualized in Mapbox Studio.

This approach had the disadvantage of providing a static solution to filtering. The styles had to be pre-generated in Mapbox Studio before making them available in Unity, an end user would not be able to dynamically apply filters while using the mobile application.

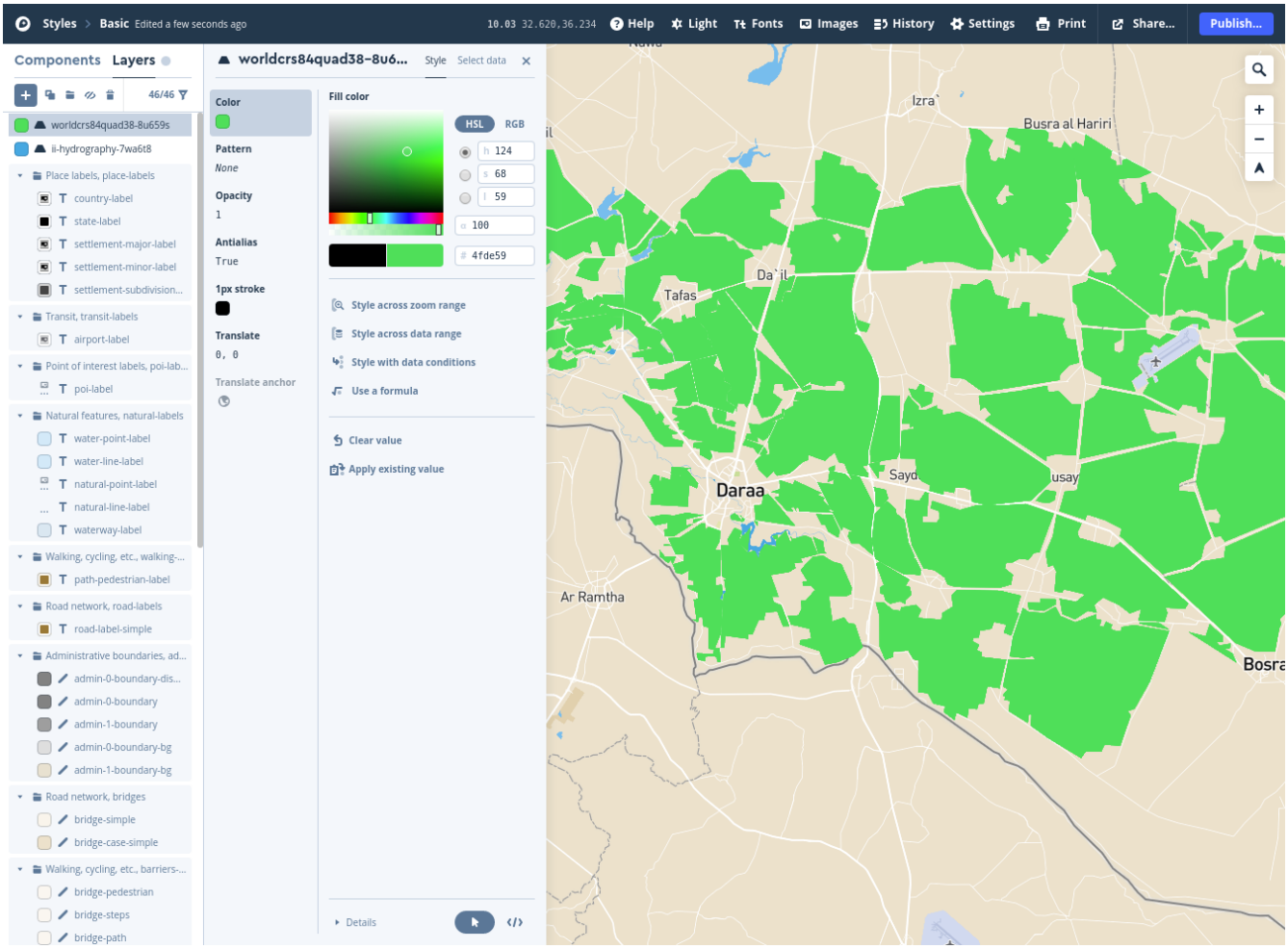


Figure 29. Style created in Mapbox Studio comprising two layers.

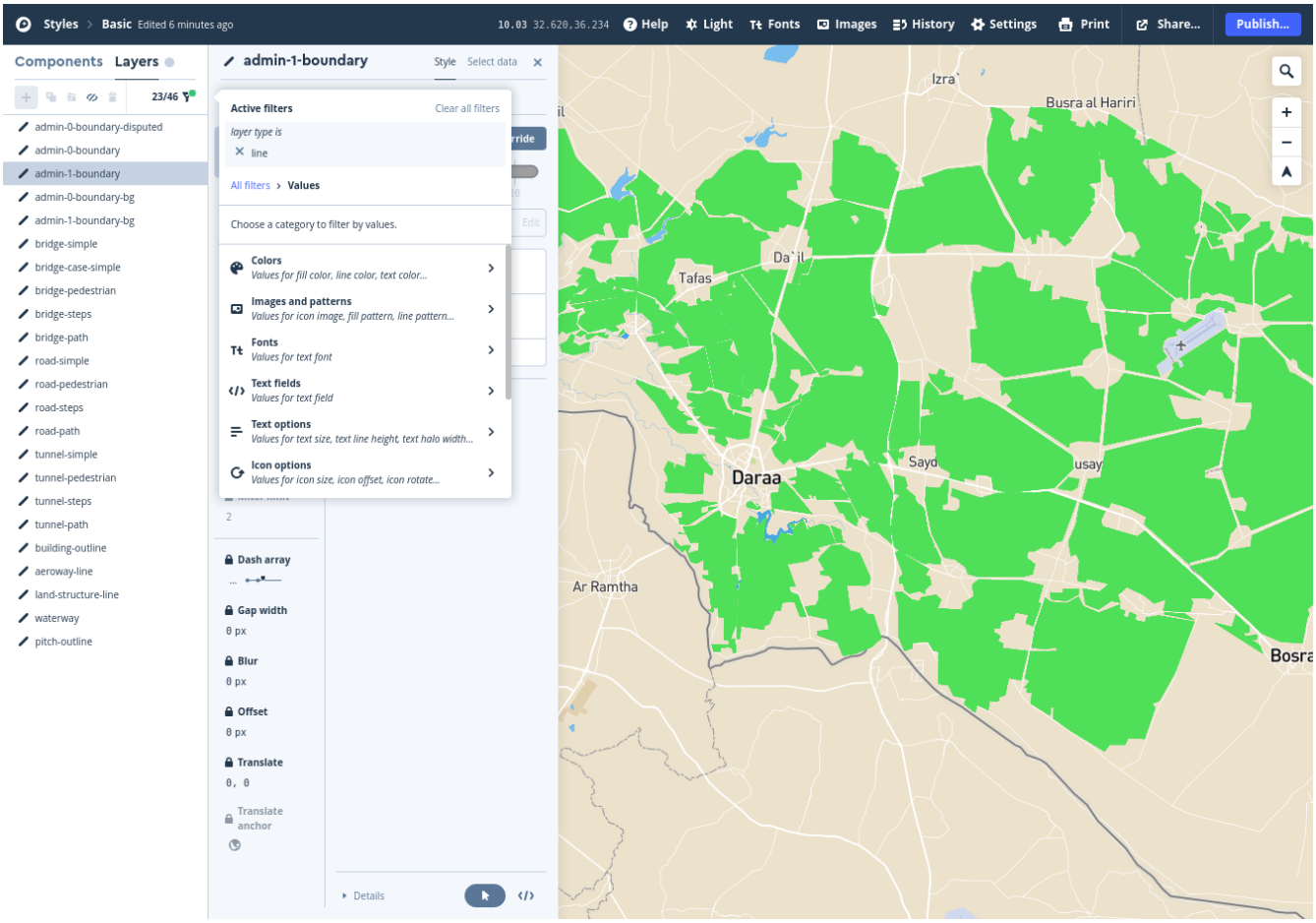


Figure 30. Filter being applied to a style in Mapbox Studio.

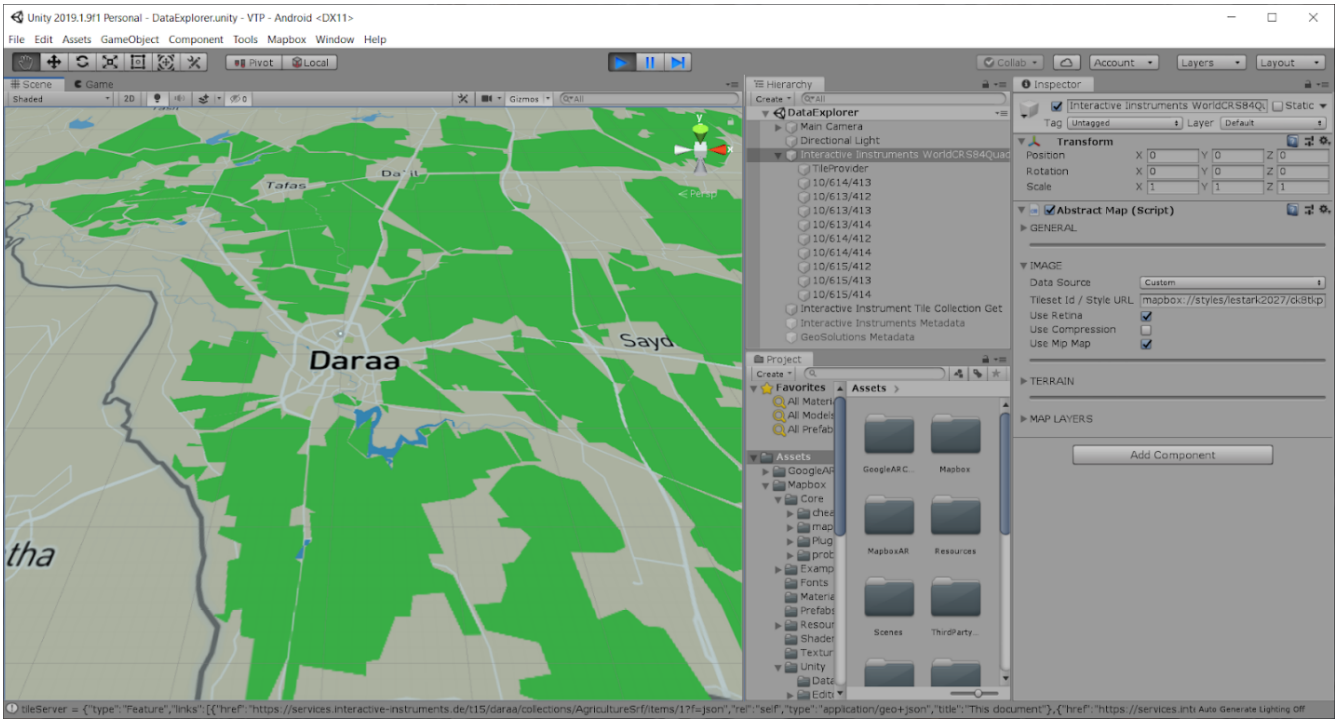


Figure 31. Style URL loaded and map rendered in Unity.

11.8.3. Future Work

The filtering approach implemented in this Pilot could be further expanded by anticipating in advance a large number of combinations of layers, styles, and filters that an end user would be

requiring, and making them easily available in the augmented reality mobile application. One Mapbox style per each combination would be created in Mapbox Studio, and made available in Unity by means of individual *scenes* the end user would select in the mobile app. Each Unity *scene* would access a specific Mapbox style URL previously created.

In order to provide end users with full filtering capabilities while using the augmented-reality mobile application, maps should be entirely generated in Unity without utilizing Mapbox SDK nor Mapbox Studio. A custom *slippy* map would need to be created in Unity, and control buttons would command C# scripts that effectively filter and style the map being rendered. This approach was indeed considered, but the workload involved would not fit the time limits of the Pilot.

11.9. GeoSolutions D104 Client

The deliverable implemented for the vector tile client was a web application built with MapStore [https://mapstore.geo-solutions.it/], an open source web-based Geographic Information System (GIS) framework.

- live demo [http://demo.vtp2.geo-solutions.it/mapstore/index.html#/]
- repository [https://github.com/geosolutions-it/ogc-vector-tiles-vtp/tree/master/vtp2]

The following components show a complete filtering workflow:

- Layers panel to list all the layers added to the map and also to provide layer related tools
- filter builder panel to apply attribute, temporal or spatial filters to a selected layer (it was available only after selecting a layer in the list).
- Map to render and support OGC vector tiles layers.
- Catalog panel to display all layers available on an OGC API service where each card represents a collection that could be displayed on map.

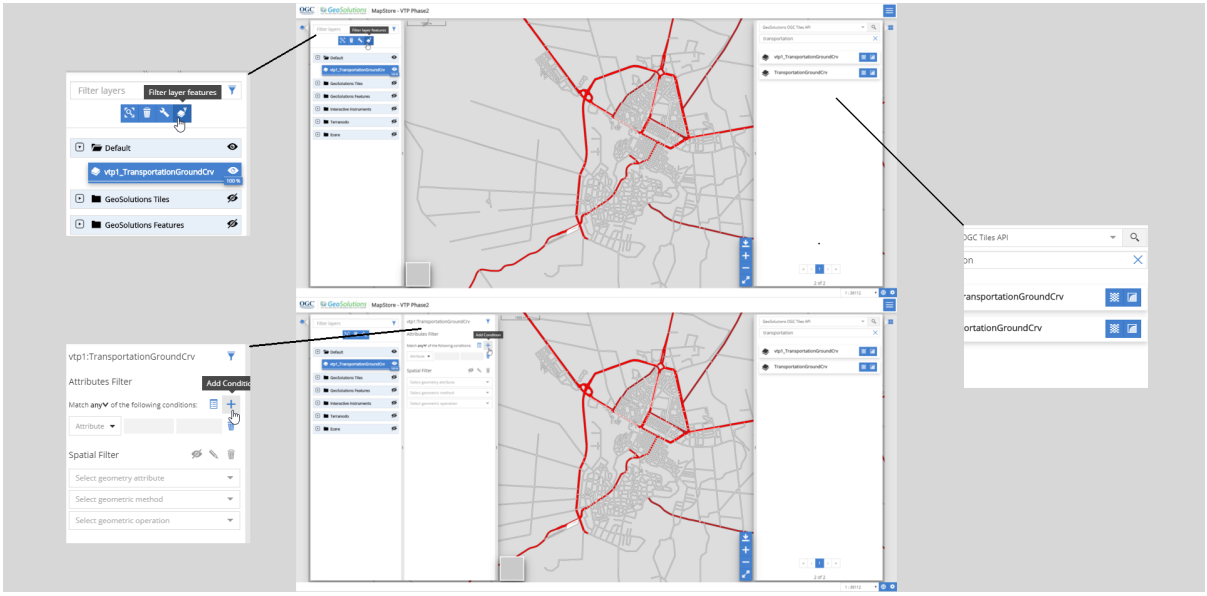


Figure 32. MapStore client with steps to activate the filter builder: add collection from an OGC OGC API - Tiles server (right box), select collection and enable filter panel (top left box) and interact with the filter builder (bottom left box)

The filter created by the Filter Builder component was a CQL filter directly included in the tiles request, as query parameter. The client thus expected to get server side filtered tiles.

The Transportation Ground Curve layer was used to test TIEs.

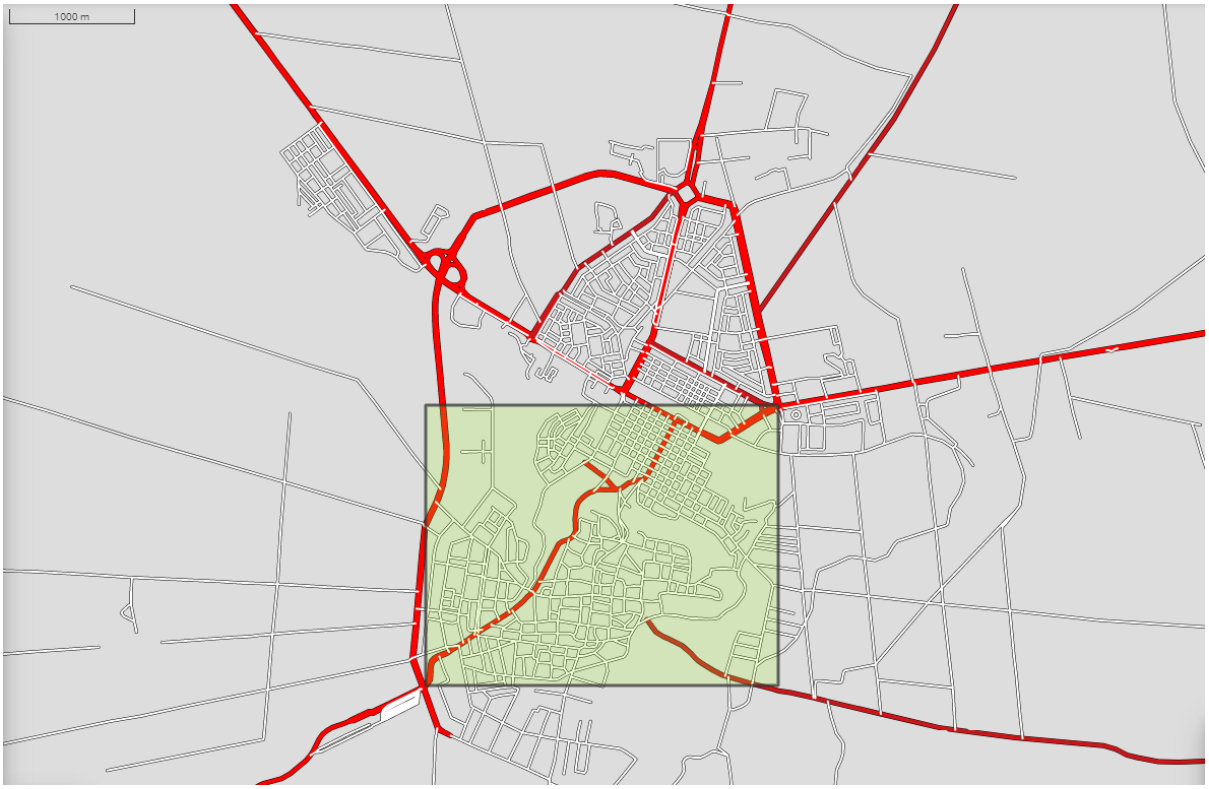


Figure 33. Transportation Ground Curve layer without filters and highlighted the polygon used in the spatial filter

Below some screenshots of filter applied to different services using the same layers.

11.9.1. Attribute filter

CQL Filter:

```
(RIN_ROI = '3')
```

Result:

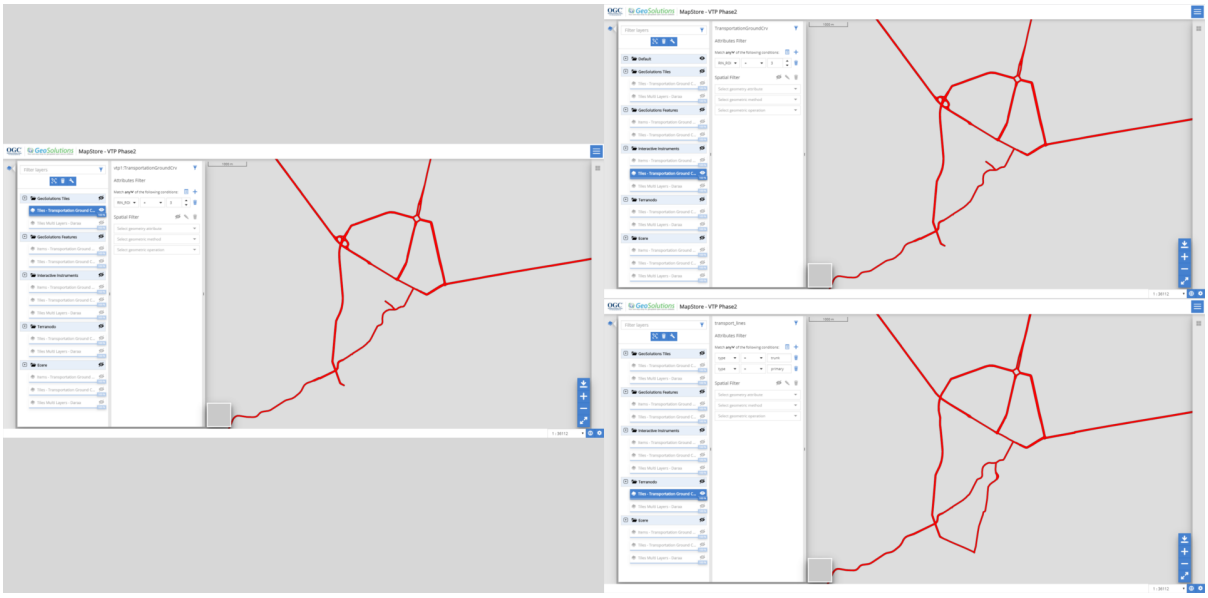


Figure 34. MapStore client with attribute filter applied to Transportation Ground Curve layer from different services: GeoSolutions (left), Interactive Instruments (top right) and Terranodo (bottom right)

11.9.2. Spatial filter

CQL Filter:

```
(WITHIN(geometry,POLYGON((36.08639717102051 32.605362330628395, 36.116266250610344
32.605362330628395, 36.116266250610344 32.625532858348336, 36.08639717102051
32.625532858348336, 36.08639717102051 32.605362330628395))))
```

Result:

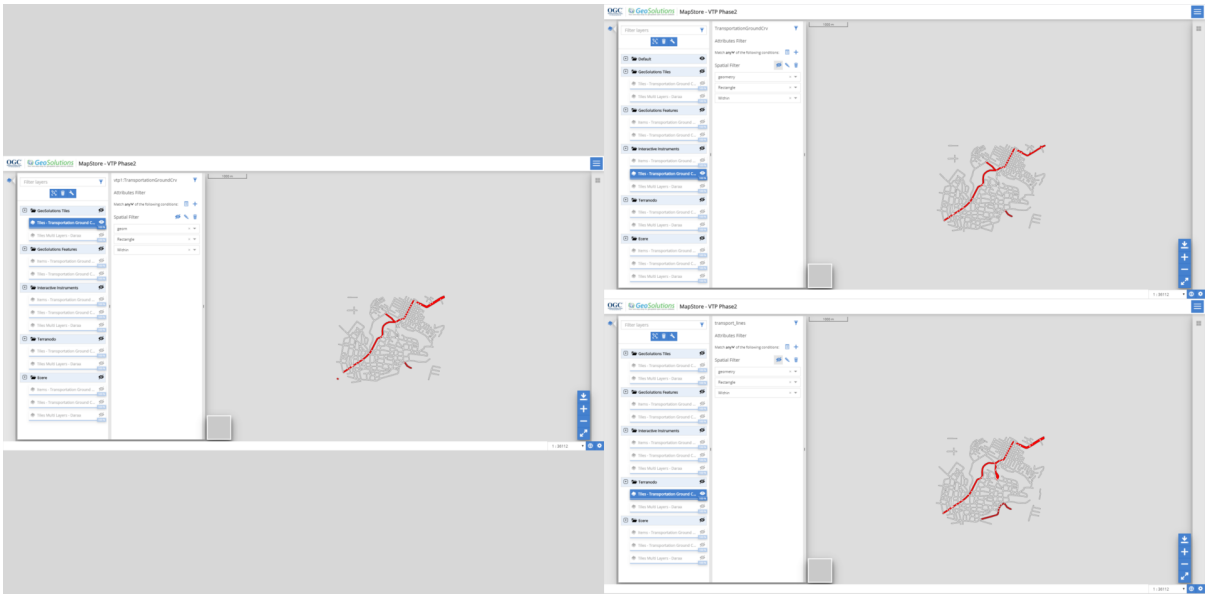


Figure 35. MapStore client with WITHIN spatial filter applied to Transportation Ground Curve layer from different services: GeoSolutions (left), Interactive Instruments (top right) and Terranodo (bottom right)

CQL Filter:

```
(DISJOINT(geometry,POLYGON((36.08639717102051 32.605362330628395, 36.116266250610344
32.605362330628395, 36.116266250610344 32.625532858348336, 36.08639717102051
32.625532858348336, 36.08639717102051 32.605362330628395))))
```

Result:

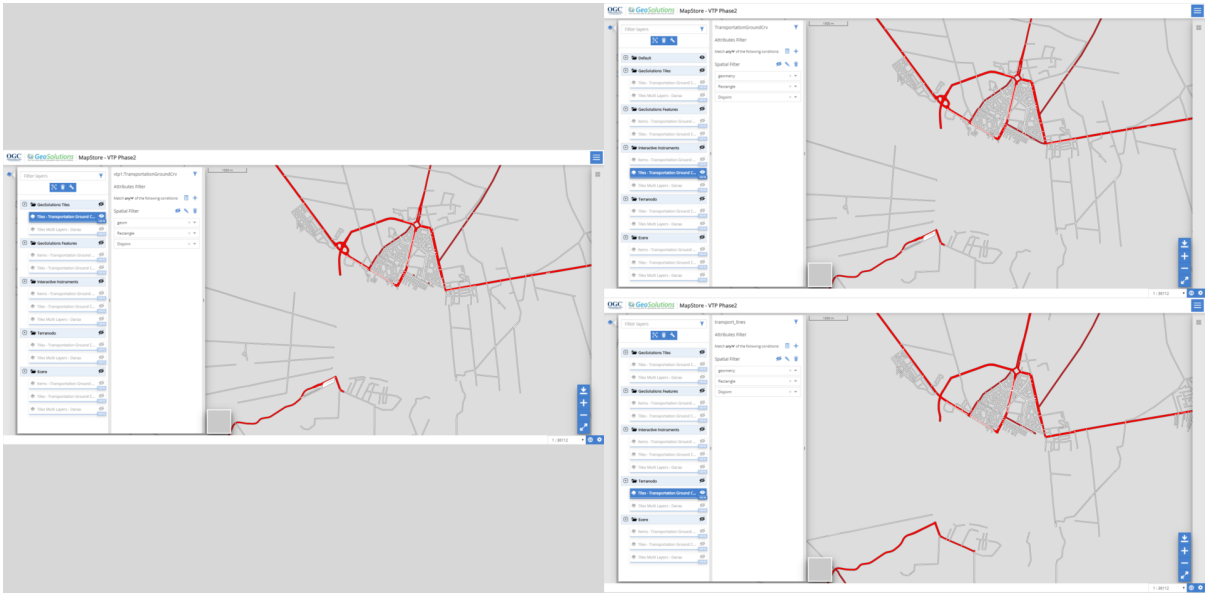


Figure 36. MapStore client with DISJOINT spatial filter applied to Transportation Ground Curve layer from different services: GeoSolutions (left), Interactive Instruments (top right) and Terranodo (bottom right)

11.9.3. Temporal filter

CQL Filter:

```
(ZI001_SDV AFTER 2014-01-01T00:00:00.000Z)
```

Result:

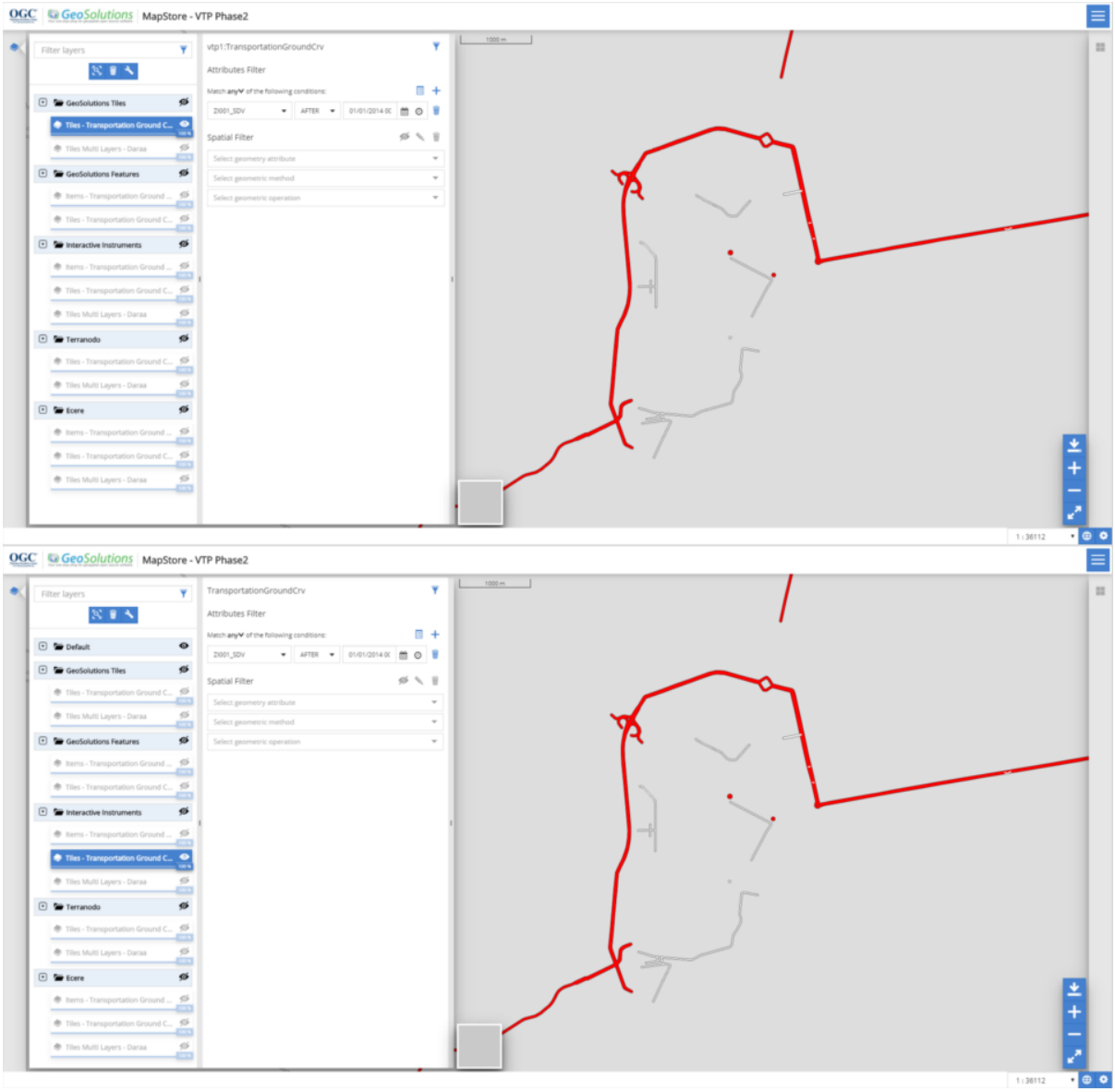


Figure 37. MapStore client with AFTER temporal filter applied to Transportation Ground Curve layer from different services: GeoSolutions (top) and Interactive Instruments (bottom)

11.9.4. Mixed filters

CQL Filter:

```
(RIN_ROI = '3') AND (WITHIN(geom,POLYGON((36.06425285339356 32.59450435638428,
36.122875213623054 32.59450435638428, 36.122875213623054 32.65407085418701,
36.06425285339356 32.65407085418701, 36.06425285339356 32.59450435638428))))
```

Result:

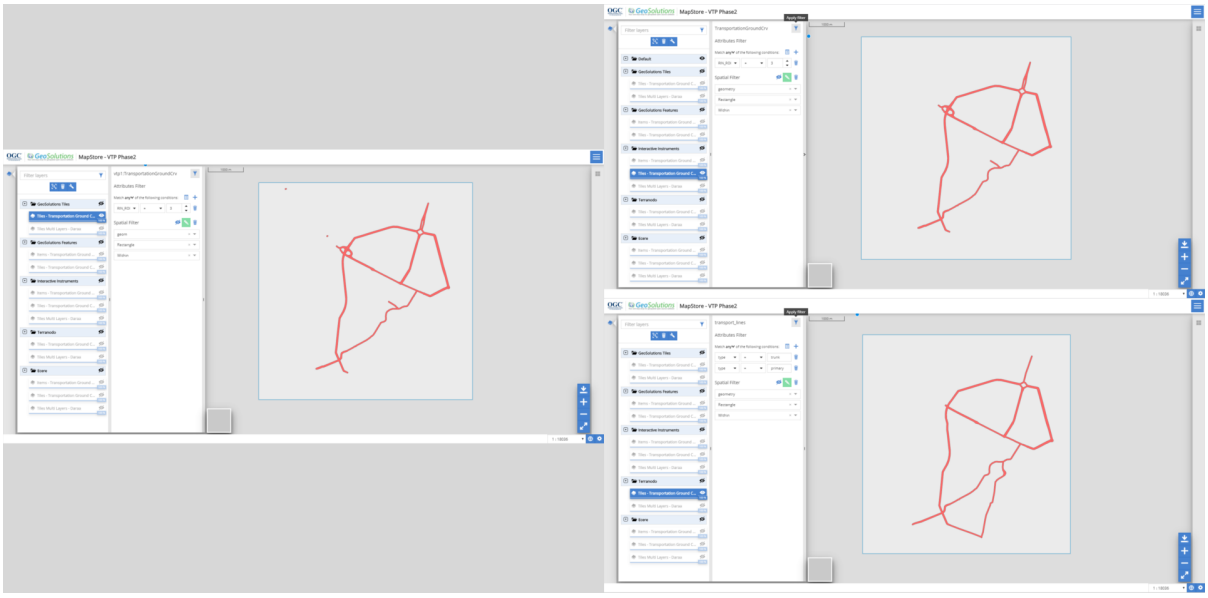


Figure 38. MapStore client with attribute and spatial filters applied to Transportation Ground Curve layer from different services: GeoSolutions (left), Interactive Instruments (top right) and Terranodo (bottom right)

NOTE	Terranodo server used a different dataset so the attribute filter applied was (type='trunk' OR type='primary') instead of (RIN_ROI = '3')

Chapter 12. Results and findings

12.1. Issues Encountered

12.1.1. CQL language issues

During the development and tests of CQL parsers the following issues were raised.

12.1.2. Quoted identifiers

In the original CQL it's possible to double quote an identifier to avoid conflicts with reserved words and issues with special characters being part of the names.

The current draft OGC API - Features CQL extension BNF does not contain such support, it was suggested [https://github.com/opengeospatial/ogcapi-features/issues/332] to add it back.

12.1.3. ENVELOPE constructor

An issue [https://github.com/opengeospatial/ogcapi-features/issues/334] was found with the `ENVELOPE` construct, used to express rectangular geometries and used in spatial filters. In particular, the axis order in the BNF uses a `west`, `east`, `north`, `south` convention, not used in other OGC protocols.

Discussion is ongoing about possibly replacing the `ENVELOPE` construct with a simpler `BBOX` spatial filter operator.

12.1.4. EXISTS operator

The catalog specification is not clear [https://github.com/opengeospatial/ogcapi-features/issues/335] about the meaning of the `EXISTS` operator, in particular, on whether the property being checked has a value, or not, or whether it is present in the schema of the data itself. For schemaless datasets, it may not even be possible to tell apart the two situations.

12.2. Findings

This section collects the findings of the Pilot activity.

12.2.1. Client versus Server-side filtering

A recurring topic in the Pilot has been is side executing the filters, two places are considered:

- Server side, during the production of the tiles or the GeoPackage
- Client side, while using the tiles

Is it to be noted that it makes sense to speak of a "filtering language" in the context of an API request, while working client side the filtering will depend on the data delivered and the tools used to read it. A few examples follow:

- In the case of a Mapbox Vector Tiles or GeoJSON file, the client side library will often provide programmatic filtering abilities in the language of choice (e.g., JavaScript)
- In the case of a GeoPackage, the client will perform filtering either directly in SQL, if attributes are available, or programmatically, after parsing the vector tiles, if the attributes are still encoded as part of the tile body.

12.2.2. Client-side control of contents and default filtering

The typical vector tiles delivery mechanisms available today assume that all filtering has to be done on the server side in advance, taking into account multiple factors:

- Data density, avoiding layers that would have too dense data for the current zoom level.
- Common styling needs, removing attributes that are typically not used in client-side styles, and that are not often needed for "info" displays.
- Data generalization based on the zoom level, considering multiple approaches, such as geometry simplification, union, selection based on importance.

This helps keep the vector tiles small compared to a server side rendered equivalent, while still allowing the client to have some latitude in rendering and eventual client side filtering.

Allowing the client to provide filters and projections to the server side allows for other avenues.

For multi-layer base maps a complex configuration is needed server side, reporting for each layer:

- The range of visibility, removing the layer at certain zoom levels, be it because it is spatially too dense, or not enough.
- The desired filtering on a zoom level by zoom level basis (e.g., including only highways at low zoom levels, while providing the entire data set at higher ones).
- Switching between different data sources depending on the zoom level (to leverage previous generalization work that might have simplified geometries, collapsed them, or simple pre-filtered them for performance reasons)
- The desired set of attributes, which will vary depending on the intended application.

This kind of configuration is better performed server-side, and would be the driver for an eventual advance seeding of the tiles. Trying to apply it on the client side would require either:

- A large configuration document to be sent along each request
- The creation of a stateful configuration, similar in principle to Stored Queries, that the client would then refer to by id, in subsequent tile requests.

Overlay layers can instead use a more dynamic filtering, provided the client as a query parameter in tile requests, depending on the current needs.

12.2.3. Filter capabilities and desired minimum filtering operator set

Regardless of the specific filter language encoding, filtering languages include a rich set of operators. This rich set helps expressiveness, but can hinder the implementation of the parser and

filter engine.

It is thus desirable to advertise the capabilities of the filtering engine, allowing the implementation of a sub-set of filters.

Within the confines of VTP2 no capabilities mechanism HAS been mandated, instead, a minimum set of filters has been agreed upon among implementors, thereby providing a suitable balance between filter expressiveness and ease of implementation.

12.3. Recommendations

12.3.1. CQL CRS geometry support

CQL currently supports geometry literals without any specification of their coordinate reference system. We recommend adding explicit support for it, either in the CQL own syntax, or as a separate request parameter.

The syntax embedding would allow for a more free form expression of filters, with a potential to have multiple spatial filters using different coordinate reference system literals.

The separate request parameter would instead imply that all literals in the filter are expressed in the same CRS. It would be however simpler to implement, and more in line with the `bbox-crs` request parameter already suggested in the OGC API - Features CRS extension.

12.3.2. Filter capabilities support

A mechanism to advertise the set of supported filtering operators is needed, in order to avoid unnecessary implementation efforts of servers-side, while allowing clients to determine which subset of filters is supported.

Ideally, this should be part of the OGC API - Features filtering extension.

12.3.3. Support for complex filtering

The current support for filters as query parameters in `GET` requests is simple, well suited for link sharing, and generally well understood.

However, there are evident situations where it would stop working due to practical URL lenght limits:

- Spatial filters with long, elaborate geometries
- Complex filters combining many conditions
- Filters including manually chosen/removed features, by identifier
- Multi-layer filters scenarios, where more than one filter needs to be specified.

A mechanism to support these use cases should be developed, e.g., stateless `POST` requests in `form-urlencoded` format using the same parameters, or stateful `POST` requests creating of stored queries that may be referred to using a filter identifier.

12.4. Future work

12.4.1. Selection of returned attributes

While the filtering allows to reduce the number of returned items, it does not support reducing the number of attributes returned along with each item.

Developing such support would require:

- Advertising a list of "returnable" properties, structurally similar to the queryables, but with a different purpose.
- Adding request parameters to support the property selection.

Similarly to filters, the selection of properties could lead to excessively long URLs, requiring usage of the `POST` method to carry the request.

In this context, a full fledged JSON based filtering and property selection (querying) support could be developed too.

12.4.2. JSON based filtering languages

While VTP2 experimented with CQL-text, there is an evident desire for a JSON based filtering language. The language can be functionally equivalent, while the JSON structure would remove the need of a full fledged text parser.

The encoding would find its natural position in larger JSON documents, while also being useful in allowing quicker implementation of servers, and on the client-side as well, when filters are programmatically built through graphical user interfaces, rather than directly entered by a human being.

12.4.3. Explore multi-layer tile filtering and querying support

As noted in the Single and multi-layer filtering section, with the exception of the Ecere Tiles API implementation (see Chapter 8 - implementations), this pilot focused on filtering single-layer tiles.

Multi-layer tiles filtering has been marked as a future-work item. In particular, in order to support multi-layer tiles filtering the following topics need to be researched in greater depth:

- Expressing a short list of filters in GET requests, eventually as a `cql-text` grammar extension.
- Expressing list of filters not fitting the practical length of a URL as a POST request (possibly in combination with multi-collection download extension for the OGC Features API)
- Advertising queryables of a multi-layer vector tiles outside of the OGC API - Features tile building blocks applications, for example, in a stand alone OGC API - Tile implementation.

Appendix A: CQL BNF

The BNF describes the syntax of the CQL text language, in its full form (not limited to the subset of operators showcased in the Vector Tiles Pilot 2).

```
#
# MODULE:  cql.bnf
# PURPOSE: A BNF grammar for the Common Query Language (CQL).
# HISTORY:
# DATE         EMAIL                     DESCRIPTION
# 13-SEP-2019  pvretano[at]cubewerx.com  Initial creation
# 28-OCT-2019  pvretano[at]cubewerx.com  Initial checkin into github.
#

#============================================================================#
# A CQL filter is a logically connected expression of one or more predicates.
#============================================================================#
cqlFilter = booleanValueExpression;

booleanValueExpression = booleanTerm | booleanValueExpression "OR" booleanTerm;

booleanTerm = booleanFactor | booleanTerm "AND" booleanFactor;

booleanFactor = ["NOT"] booleanPrimary;

booleanPrimary = predicate
| leftParen cqlFilter rightParen;

#============================================================================#
#  CQL supports scalar, spatial, temporal and existence predicates.
#============================================================================#
predicate = comparisonPredicate
          | spatialPredicate
          | temporalPredicate
          | existencePredicate
          | inPredicate;

#============================================================================#
# A comparison predicate evaluates if two scalar expression statisfy the
# specified comparison operator.  The comparion operators include an operator
# to evaluate regular expressions (LIKE), a range evaluation operator and
# an operator to test if a scalar expression is NULL or not.
#============================================================================#
comparisonPredicate = binaryComparisonPredicate
                    | propertyIsLikePredicate
                    | propertyIsBetweenPredicate
                    | propertyIsNullPredicate;

binaryComparisonPredicate = scalarExpression comparisonOperator scalarExpression;
```

```
propertyIsLikePredicate =  scalarExpression "LIKE" regularExpression;

propertyIsBetweenPredicate = scalarExpression "BETWEEN"
scalarExpression "AND" scalarExpression;

propertyIsNullPredicate = scalarExpression "IS" ["NOT"] "NULL";

#
# A scalar expression is the property name, a chracter literal, a numeric
# literal or a function/method invocation that returns a scalar value.
#
scalarExpression = propertyName
| characterLiteral
| numericLiteral
| function
| arithmeticExpression;

# NOTE: This is just a place holder for a regular expression
#       We want to be able to say stuff like "<prop> LIKE 'Toronto%'" where
#       the '%' character means "match zero or more characters".
regularExpression = characterLiteral;

comparisonOperator = eq | neq | lt | gt | lteq | gteq;

neq = lt gt;

gteq = gt eq;

lteq = lt eq;

#============================================================================#
# A spatial predicate evaluates if two spatial expressions satisfy the
# specified spatial operator.
#============================================================================#
spatialPredicate =  spatialOperator leftParen geomExpression comma geomExpression
rightParen;

# NOTE: The buffer operators (DWITHIN and BEYOND) are not included because
#       these are outside the scope of a "simple" core for CQL.  These
#       can be added as extensions.
#
spatialOperator = "EQUALS" | "DISJOINT" | "TOUCHES" | "WITHIN" | "OVERLAPS"
| "CROSSES" | "INTERSECTS" | "CONTAINS";

# A geometric expression is a property name of a geometry-valued property,
# a geometric literal (expressed as WKT) or a function that returns a
# geometric value.
#
geomExpression = propertyName
| geomLiteral
| function;
```

```
#============================================================================#
# A temporal predicate evaluates if two temporal expressions satisfy the
# specified temporal operator.
#============================================================================#
temporalPredicate = temporalExpression temporalOperator
temporalExpression [temporalExpression];

temporalExpression = propertyName
| temporalLiteral
| function;

temporalOperator = "AFTER" | "BEFORE" | "BEGINS" | "BEGUNBY" | "TCONTAINS"
| "DURING" | "ENDEDBY" | "ENDS" | "TEQUALS" | "MEETS"
| "METBY" | "TOVERLAPS" | "OVERLAPPEDBY" | "ANYINTERACTS"
| "INTERSECTS";

#============================================================================#
# The existence predicate evalues whether the specified property exists
# in the current context. This predicate was added to accomodate the fact
# that OAPIF feature collections (and likely other specification) are
# heterogeneous with respect to schema.
#============================================================================#
existencePredicate = propertyName "EXISTS"
                   | propertyName "DOES" "NOT" "EXIST";

#============================================================================#
# The IN predicate
#============================================================================#
inPredicate = propertyName "IN" leftParen { characterLiteral |
                                            numericLiteral |
                                            geomLiteral |
                                            temporalLiteral |
                                            function } rightParen;

#============================================================================#
# Definition of a FUNCTION
# NOTE: How do we advertise which functions an implementation offer?
#       In the OpenAPI document I suppose!
#============================================================================#
function = identifier {argumentList};

argumentList = leftParen [positionalArgument]  rightParen;

positionalArgument = argument [ { comma argument } ];

argument = characterLiteral
| numericLiteral
| geomLiteral
| propertyName
| arithmeticExpression;
```

```
#============================================================================#
# An arithemtic expression is an expression composed of an arithmetic
# operand (a property name, a number or a function that returns a number),
# an arithmetic operators (+,-,*,/) and another arithmetic operand.
#============================================================================#
arithmeticExpression = arithmeticOperand arithmeticOperator arithmeticOperand;

arithmeticOperator = plusSign | minusSign | asterisk | solidus;

arithmeticOperand = propertyName
| numericLiteral
| function;

#============================================================================#
# Definition of NUMERIC literals
#============================================================================#
numericLiteral = unsignedNumericLiteral | signedNumericLiteral;

unsignedNumericLiteral = exactNumericLiteral | approximateNumericLiteral;

signedNumericLiteral = [sign] exactNumericLiteral | approximateNumericLiteral;

exactNumericLiteral = unsignedInteger [ period [ unsignedInteger ] ]
| period unsignedInteger;

approximateNumericLiteral = mantissa "E" exponent;

mantissa = exactNumericLiteral;

exponent = signedInteger;

signedInteger = [ sign ] unsignedInteger;

unsignedInteger = {digit};

sign = plusSign | minusSign;

#============================================================================#
# Definition of CHARACTER literals
#============================================================================#
characterLiteral = characterStringLiteral
                 | bitStringLiteral
                 | hexStringLiteral;

characterStringLiteral = quote [ {character} ] quote;

bitStringLiteral = "B" quote [ {bit} ] quote;

hexStringLiteral = "X" quote [ {hexit} ] quote;
```

```
propertyName = identifier;

identifier = identifierStart [ {identifierPart} ];

identifierStart = alpha [{octothorp|dollar|underscore|alpha|digit}];

identifierPart = alpha | digit;

character = alpha | digit | specialCharacter | quoteQuote;

quoteQuote = quote quote;

# NOTE: This production is supposed to be any alphabetic character from
#       the character set.
#
#       I use the A-Z, a-z range here as placeholders because:
#       (a) I have no idea how to indicate that alpha can be
#           any alphabetic UTF-8 character
#       (b) the validators I am using can only handle ASCII chars
#
alpha = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" |
        "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" |
        "W" | "X" | "Y" | "Z" |
        "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" |
        "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" |
        "w" | "x" | "y" | "z";

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";

specialCharacter = percent | ampersand | leftParen | rightParen | asterisk
| plusSign | comma | minusSign | period | solidus | colon
| semicolon | lt | gt | eq | questionMark | underscore
| verticalBar | doubleQuote ;

octothorp = "#";

dollar = "$";

underscore = "_";

doubleQuote = "\"";

percent = "%";

ampersand = "&";

quote = "'";

leftParen = "(";

rightParen = ")";
```

```
asterisk = "*";

plusSign = "+";

comma = ",";

minusSign = "-";

period = ".";

solidus = "/";

colon = ":";

semicolon = ";";

lt = "<";

eq = "=";

gt = ">";

questionMark = "?";

verticalBar = "|";

bit = "0" | "1";

hexit = digit | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f";

#============================================================================#
# Definition of TEMPORAL literals
#
# NOTE: Is the fact the time zones are supported too complicated for a
#       simple CQL?  Perhaps the "core" of CQL should just support UTC.
#============================================================================#
temporalLiteral = fullDate | fullDate "T" utcTime;

fullDate   = dateYear "-" dateMonth "-" dateDay;

dateYear   = digit digit digit digit;

dateMonth  = digit digit;

dateDay    = digit digit;

utcTime  = timeHour ":" timeMinute ":" timeSecond [timeZoneOffset];

timeZoneOffset = "Z" | sign timeHour;
```

```
timeHour   = digit digit;

timeMinute = digit digit;

timeSecond = digit digit [period digit {digit}];

#============================================================================#
# Definition of GEOMETRIC literals
#
# NOTE: This is basically BNF that define WKT encoding; it would be nice
#       to instead reference some normative BNF for WKT.
#============================================================================#
geomLiteral = pointTaggedText
            | linestringTaggedText
            | polygonTaggedText
            | multipointTaggedText
            | multilinestringTaggedText
            | multipolygonTaggedText
            | geometryCollectionTaggedText
            | envelopeTaggedText;

pointTaggedText = "POINT" pointText;

linestringTaggedText = "LINESTRING" lineStringText;

polygonTaggedText = "POLYGON" polygonText;

multipointTaggedText = "MULTIPOINT" multiPointText;

multilinestringTaggedText = "MULTILINESTRING" multiLineStringText;

multipolygonTaggedText = "MULTIPOLYGON" multiPolygonText;

geometryCollectionTaggedText = "GEOMETRYCOLLECTION" geometryCollectionText;

pointText = leftParen point rightParen;

point = xCoord yCoord [zCoord];

xCoord = signedNumericLiteral;

yCoord = signedNumericLiteral;

zCoord = signedNumericLiteral;

lineStringText = leftParen point {comma point} rightParen;

polygonText =  leftParen lineStringText {comma lineStringText} rightParen;

multiPointText = leftParen pointText {comma pointText} rightParen;
```

```
multiLineStringText = leftParen lineStringText {comma lineStringText} rightParen;

multiPolygonText = leftParen polygonText {comma polygonText} rightParen;

geometryCollectionText = leftParen geomLiteral {comma geomLiteral} rightParen;

envelopeTaggedText = "ENVELOPE" envelopeText;

envelopeText = leftParen westBoundLon comma eastBoundLon comma northBoundLat comma
southBoundLat [comma minElev comma maxElev] rightParen;

westBoundLon = signedNumericLiteral;

eastBoundLon = signedNumericLiteral;

northBoundLat = signedNumericLiteral;

southBoundLat = signedNumericLiteral;

minElev = signedNumericLiteral;

maxElev = signedNumericLiteral;
```

Appendix B: Queryables

The following is an extract from [2], including only the description of the Queryables extension for styling [http://docs.opengeospatial.org/per/19-010r2.html#get_queryables], used in VTP2 to expose the list of properties available for filtering.

A similar document is foreseen to be part of the OGC API - Features filtering extension, once it is ready.

B.1. Requirement Class "Queryables"

Requirements Class	
http://www.opengis.net/t15/opf-styles-1/{m_n}/req/queryables	
Target type	Web API
Dependency	OGC API - Features - Part 1: Core, conformance class "Core" [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html#rc_core]

B.1.1. Fetch the queryable properties of the features in a collection

This operation returns the list of queryable properties that can be used to filter features in a collection and supports clients in constructing expressions for selection criteria in queries on features in the collection.

The response is an object with a member `queryables`, which contains an array with a description of the queryable properties of the feature collection. "Queryable" means that the property may be used in query expressions, such as in a query extension to OGC API - Features or as part of a selection criteria in an OGC SLD/SE or Mapbox styling rule.

Often the list of queryables for a collection will be a subset of all available properties in the features and be restricted to those properties that are, for example, indexed in the backend datastore to support performant queries.

For each queryable property the following information is or may be provided:

- `id` (required) - the property name for use in expressions.
- `type` (required) - the data type of the property, one of
  - `string`
  - `uri`
  - `enum`
  - `number`
  - `integer`
  - `date`
  - `dateTime`
  - `boolean`
- `description` (optional) - a description of the property.

- `required` (optional) - indicator whether the property is always present in features.
- `mediaTypes` (optional) - in general, the representation of the queryables is meant to be independent of the feature encoding. However, this is not always the case. For example, length restrictions or namespace prefixes may result in different property identifiers for the same property. To support this, the definition of a queryable may be restricted to one or more feature encodings (media types).
- `pattern` (optional, only for "string" and "uri") - a regular expression to validate the values of the property.
- `values` (required, only for "enum") - an array of valid values of the property.
- `range` (optional, only for "number", "integer", "date" and "dateTime") - the range of valid values expressed as an array with two items. Open ranges can be expressed using null for the minimum or maximum value.

Note that this is not about providing a schema for the features in the collection. A schema provides a complete syntactic definition of a specific feature encoding, typically for validation purposes. Schema languages like XML Schema or JSON Schema are much richer and support more complex syntactic rules, but are also more complex to parse.

Requirement 1	**/req/queryables/op**
A	The server SHALL support the HTTP GET operation at the path `/collection/{collectionId}/queryables` for each collection.

Requirement 2	**/req/queryables/success**
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`.

B	The content of that response SHALL be based upon the OpenAPI 3.0 schema component "queryables", if the `itemType` of the collection is `feature`:
	
	queryables
	
	`type: object`
	`required:`
	`  - queryables`
	`properties:`
	`  queryables:`
	`  type: array`
	`  nullable: true`
	`  items:`
	`  oneOf:`
	`  - $ref: 'queryable-string'`
	`  - $ref: 'queryable-enum'`
	`  - $ref: 'queryable-number'`
	`  - $ref: 'queryable-boolean'`
	`  - $ref: 'queryable-date'`
	`  - $ref: 'queryable-dateTime'`
	
	queryable
	
	`type: object`
	`nullable: true`
	`required:`
	`  - id`
	`  - type`
	`properties:`
	`  id:`
	`  type: string`
	`  nullable: true`
	`  description: |-`
	`  the property name for use in expressions`
	`  title:`
	`  type: string`
	`  nullable: true`
	`  description: |-`
	`  the title of the property for presentation to a`
	`  human user`
	`  description:`
	`  type: string`
	`  nullable: true`
	`  description: |-`
	`  a description of the property`
	`  required:`
	`  type: boolean`
	`  nullable: true`
	`  default: false`

C	The `id` member of each queryable SHALL be unique.

Note that this requirements class does not specify any requirements on collections that are not feature collections.

Example 1. JSON encoding of queryables

```
{
  "queryables": [
    {
      "id": "name",
      "description": "the name of the vegetation area",
      "required": true,
      "type": "string",
      "example": "[A-Z0-9]{5}"
    },
    {
      "id": "type",
      "description": "the dominant characteristic of the vegetation area",
      "type": "enum",
      "values": [
        "grassland",
        "forest",
        "farmland"
      ]
    },
    {
      "id": "count",
      "description": "the number of cattle",
      "type": "integer",
      "range": [
        0,
        null
      ]
    },
    {
      "id": "fenced",
      "description": "indicator whether the area is walled or fenced",
      "type": "boolean"
    },
    {
      "id": "inspectionDate",
      "description": "the date of the last inspection",
      "type": "date",
      "range": [
        "2010-01-01",
        null
      ]
    },
```

```
    {
      "id": "lastUpdate",
      "description": "the date of the last update of the feature",
      "type": "dateTime",
      "range": [
        "2018-01-01T00:00:00Z",
        null
      ]
    }
  ]
}
```



```
        a regular expression to validate the values
        of the property
```


queryable-enum


```
allOf:
- $ref: 'queryable'
- type: object
  nullable: true
  required:
    - values
  properties:
    values:
      type: array
      nullable: true
      description: |-
        the list of values of the property
      items:
        type: string
```


queryable-number


```
allOf:
- $ref: 'queryable'
- type: object
  nullable: true
  properties:
    range:
      type: array
      nullable: true
      minItems: 2
      maxItems: 2
      items:
        type: number
        nullable: true
      description: |-
        a range of valid values; open range can be
        expressed using `null`
```

Appendix C: Filter capabilities

A filter capabilities document allows a server to enumerate the supported filter operators, as well as documenting the filter functions included for extensibility.

The following document is an excerpt of GeoServer 2.17.x filter capabilities, as implemented during the STAC and OGC API - Features and Catalogues Sprint [https://www.opengeospatial.org/projects/initiatives/ogcapi-featu].

The list of functions has been truncated to keep the example short, more information about GeoServer supported functions can be found in the filter functions reference [https://docs.geoserver.org/stable/en/user/filter/function_reference.html].


```
{
  "conformanceClasses": [
    "http://www.opengis.net/spec/cql/1.0/conf/core",
    "http://www.opengis.net/spec/cql/1.0/conf/spatial",
    "http://www.opengis.net/spec/cql/1.0/conf/temporal"
  ],
  "capabilities": [
    {
      "name": "logical",
      "operators": [ "and", "or", "not" ]
    },
    {
      "name": "comparison",
      "operators": [ "lt", "lte", "gt", "gte", "gt", "neq", "like", "between", "in" ]
    },
    {
      "name": "spatial",
      "operators": [
        "equals",
        "disjoint",
        "touches",
        "within",
        "overlaps",
        "crosses",
        "intersects",
        "contains"
      ]
    },
    {
      "name": "temporal",
      "operators": [
        "after",
        "before",
        "begins",
        "begunby",
        "tcontains",
```

```
        "during",
        "endedby",
        "ends",
        "tequals",
        "meets",
        "metby",
        "toverlaps",
        "overlappedby",
        "anyinteracts",
        "intersects"
      ]
    },
    {
      "name": "arithmetic",
      "operators": [ "+", "-", "*", "/" ]
    }
  ],
  "functions": [
    {
      "name": "Area",
      "returns": {
        "name": "area",
        "type": "number"
      },
      "arguments": [ { "name": "geometry", "type": "geometry" } ]
    },
    {
      "name": "Concatenate",
      "returns": {
        "name": "result",
        "type": "string"
      },
      "arguments": [ { "name": "text", "type": "string" } ]
    },
    {
      "name": "IEEEremainder",
      "returns": {
        "name": "remainder",
        "type": "number"
      },
      "arguments": [
        {
          "name": "dividend",
          "type": "integer"
        },
        {
          "name": "divisor",
          "type": "integer"
        }
      ]
    },
```

```
    {
      "name": "PolyLabeller",
      "returns": {
        "name": "result",
        "type": "geometry"
      },
      "arguments": [
        {
          "name": "polygon",
          "type": "geometry"
        },
        {
          "name": "precision",
          "type": "number"
        }
      ]
    },
    {
      "name": "abs",
      "returns": {
        "name": "abs",
        "type": "integer"
      },
      "arguments": [ { "name": "int", "type": "integer" } ]
    },
    {
      "name": "acos",
      "returns": {
        "name": "arc cosine",
        "type": "number"
      },
      "arguments": [ { "name": "value", "type": "number" } ]
    },
    {
      "name": "all",
      "returns": {
        "name": "return",
        "type": "string"
      },
      "arguments": []
    },
    {
      "name": "any",
      "returns": {
        "name": "return",
        "type": "string"
      },
      "arguments": []
    }
  ]
}
```

Appendix D: Revision History

Table 4. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
November 11, 2019	G. Hobona	.1	all	initial version
November 20, 2019	A. Aime	.2	1	added document number
December 10, 2019	A. Aime	.3	all	references, bibliography, chapter structure and UML model
January 7, 2020	A. Aime	.4	all	more work on document structure
January 21, 2020	A. Aime	.5	all	updated UML model, update findings based on latest meetings
February 7, 2020	A. Aime	.6	all	improved filter descriptions, added CQL examples
February 18, 2020	A. Aime	.7	6	expanding previous work section
February 26, 2020	A. Aime	.8	10	GeoSolutions server and client implementation s described
February 26, 2020	C. Portele	.9	10	interactive instruments server implementation described
March 19, 2020	G. Hobona	.10	various	updated bibliograpy and updated contributors list, fixed figure identifiers

Date	Editor	Release	Primary clauses modified	Descriptions
March 24, 2020	G. Hobona and A. Aime	.10	various	applied G. Hobona review feedback, expanded on reccomendations and future work
March 25, 2020	C. Reed and A. Aime	.11	various	applied C. Reese review feedback
March 28, 2020	C. Portele	.12	implementations	Remove contents shared with the summary ER
March 30, 2020	A. Aime	.13	various	Refine GeoSolutions implementation entries, merge findings and results
April 14, 2020	A. Aime	.13	various	Refine GeoSolutions implementation entries, merge findings and results
April 8, 2020	J. Yutzler	.14	GeoPackage filtering	First draft of the GeoPackage filtering section
April 10, 2020	J. Jacovella-St-Louis	.15	GeoPackage filtering, Implementations	Ecere components description
April 11, 2020	J. Johnson	.16	Implementations	Terranodo components description
April 14, 2020	S. Taleisnik	.16	Implementations	Skymantics components description

Appendix E: Bibliography

1. Portele, C.: OGC Testbed-14 Next Generation APIs: Complex Feature Handling Engineering Report. OGC 18-021,Open Geospatial Consortium, https://docs.opengeospatial.org/per/18-021.html (2019).
2. Portele, C.: OGC Testbed-15: Styles API Engineering Report. OGC 19-010r2,Open Geospatial Consortium, http://docs.opengeospatial.org/per/19-010r2.html (2019).