

OGC Vector Tiles Pilot 2
Summary Engineering Report

Publication Date: 2020-07-07

Approval Date: 2020-06-26

Submission Date: 2020-04-27

Reference number of this document: OGC 19-088r2

Reference URL for this document: <http://www.opengis.net/doc/PER/VTP2-summary>

Category: OGC Public Engineering Report

Editors: Gobe Hobona, Terry Idol

Title: OGC Vector Tiles Pilot 2: Summary Engineering Report

OGC Public Engineering Report

COPYRIGHT

Copyright © 2020 Open Geospatial Consortium. To obtain additional rights of use, visit <http://www.opengeospatial.org/>

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 NON-INFRINGEMENT 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.

Table of Contents

1. Executive Summary	7
2. Introduction	9
2.1. Requirements & Research Motivation	9
2.2. Recommendations	10
2.3. Document contributor contact points	10
2.4. Foreword	11
3. References	12
4. Terms and definitions	13
4.1. Abbreviated terms	15
5. Overview	16
6. Context	17
6.1. Scenario	17
6.2. Use Cases	18
6.2.1. Filtering Use Case	18
6.2.2. Metadata Use Case	18
7. Architecture	19
7.1. OGC API - Features standard	19
7.2. OGC Web Feature Service (WFS) standard	20
7.3. OGC Web Map Tile Service (WMTS) standard	20
7.4. OGC GeoPackage standard	20
7.5. OGC API - Maps draft specification	20
7.6. OGC API - Tiles draft specification	21
7.7. VTP2 Filtering Language proof of concept	21
7.8. VTP2 Metadata model proof of concept	21
8. Vector Tiles GeoPackage Model	22
8.1. Vector Tiles	22
8.2. Portrayal Information	23
8.3. Metadata	24
8.4. Semantic Annotations	25
8.4.1. Styles	27
8.4.2. Stylable Layer Sets	28
8.5. Attributes	28
8.6. Attributes with Related Tables	30
8.7. Tile Matrix Sets	31
9. Implementations	33
9.1. Services and Data Producers	33
9.1.1. Ecere D103 Features, Tiles and Styles API	33
9.1.2. Ecere D106 GeoPackage Producer	39

9.1.3. GeoSolutions D100 Features API	43
9.1.4. GeoSolutions D102 Tiles API	49
9.1.5. interactive instruments D101 Features, Tiles and Styles API	53
9.1.6. Support for filtering	60
9.1.7. Support for additional tiling schemes	60
9.1.8. Image Matters D107 GeoPackage	64
9.1.9. Terranodo D100 Features API	66
9.2. Client applications	67
9.2.1. Ecere D105 OGC API Client and D106 GeoPackage visualization	67
9.2.2. GeoSolutions D104 Client	71
9.2.3. Skymantics D104 Client	74
10. Results	81
10.1. Considerations	81
10.1.1. Portrayal information	81
10.1.2. Shared or Disjoint APIs	82
10.1.3. Static API	82
10.1.4. Tile Cache vs. Tile Set	83
10.1.5. Out of bounds and empty tiles	83
10.1.6. Collections	84
10.2. Technology Integration Experiments	85
10.2.1. Data Exchange TIEs	86
10.2.2. Styles Encoding TIEs	87
10.2.3. Multi-Layer Tile TIEs	88
10.2.4. Multiple Projections TIEs	88
10.2.5. TileJSON (Metadata) TIEs	90
10.2.6. Filtering TIEs	91
11. Key Findings	93
11.1. Bounding box inconsistency	93
11.2. Online and offline access to multi-layer vector tiles can be supported by OGC APIs and GeoPackage	93
11.3. The absence of a way to describe schemas creates difficulty for data transfer from an OGC API to a GeoPackage	94
11.4. There is a need for generic collections that do not advertise much about their contents ...	94
11.5. There is a need for multi-layer tiles without having explicit collections	94
11.6. The need to test larger datasets	95
11.7. Proposed GeoPackage extensions are ready for progression	95
11.8. Consistency of schemas across some OGC resources could be improved	95
12. Recommendations	96
12.1. Key recommendations	96
12.1.1. Publish schemas through the Tiles and Features APIs	96
12.1.2. Development of a standard for Tile Set Metadata	96

12.1.3. Introduce additional concepts in the OGC Symbology Conceptual Core Model	96
12.1.4. OGC API - Tiles should provide direction on multi-layer vector tiles schemas	96
12.1.5. OGC API - Tiles should allow for multi-layer tiles that are without explicit collections .	97
12.1.6. TileJSON editors should register a media type for the specification	97
12.1.7. OGC API - Common should allow for collections that do not advertise their content types	98
12.1.8. Development of Mapbox Vector Tiles and TileJSON as OGC Community Standards	98
12.2. Future Work	98
Appendix A: Proposed GeoPackage Extensions (Informative)	100
A.1. Vector Tiles Extension	100
A.2. Mapbox Vector Tiles Extension	102
A.3. GeoJSON Vector Tiles Extension	104
A.4. GeoPackage Portrayal Extension	105
A.5. GeoPackage Semantic Annotation Extension	108
A.6. Vector Tiles Attributes Extension	110
A.7. GeoPackage Tile Matrix Set Extension	112
Appendix B: Additional Screenshots	116
B.1. GeoSolutions D101 Feature Server	116
B.1.1. GeoSolutions D104 Client	116
B.1.2. Ecere D103 Features, Tiles and Styles API	117
B.2. Ecere D103 Client	121
B.3. interactive instruments	123
Appendix C: Revision History	127
Appendix D: Bibliography	128

Chapter 1. Executive Summary

Consistent online and offline support for data access is important, particularly in environments where networks have Denied, Degraded, Intermittent or Limited (DDIL) connectivity. In some cases, end-users of geospatial data have to operate in such environments over long periods of time. Humanitarian agencies, the military, and first responders are some of the groups that work in such challenging operational environments.

The OGC Vector Tiles Pilot Phase 2 (VTP2) Interoperability Initiative 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. This document interchangeably uses the terms "tiled feature data" and "vector tiles" to refer to the approach of tiling vector feature data. The objectives of the VTP2 initiative were as follows:

- a. Metadata: Describe stored tile caches in necessary detail, using the NSG Metadata Foundation (NMF), to enable usage and updating of each tile cache without analyzing each individual tile.
- b. Filtering Language: Develop a filtering language for vector tiles, then implement and exercise the filtering language on clients and servers.
- c. GeoPackage: Using [GeoPackage](https://portal.opengeospatial.org/files/12-128r15) [https://portal.opengeospatial.org/files/12-128r15] technology, develop solutions for robust offline/online usage scenarios by associating tiled vector feature tables in GeoPackages with attribute tables to allow applying similar filters to GeoPackages as being applied to the draft [OGC API - Tiles](https://github.com/opengeospatial/OGC-API-Tiles/tree/master/core) [https://github.com/opengeospatial/OGC-API-Tiles/tree/master/core] specification.
- d. Style Sharing: Develop online/offline symbol and style sharing for content in vector tiles. In other words, enable a user to request pre-rendered tiled vector data by defining a particular style while online and then be able to render the data on the client-side when the connection is lost.
- e. Experimentation: Test structuring content in vector tiles using the following Tile Matrix Sets delivered by tile and feature servers and implemented in GeoPackages: EPSG:3395 and EPSG:4326 as specified in Annex D of the [OGC Two Dimensional Tile Matrix Set standard](http://docs.opengeospatial.org/is/17-083r2/17-083r2.html) [http://docs.opengeospatial.org/is/17-083r2/17-083r2.html].

The various APIs used in this pilot are from the emerging suite of OGC API standards. The first of the OGC API standards to be approved by the OGC Membership and the OGC Technical Committee (TC) is the [OGC API - Features](http://docs.opengeospatial.org/is/17-069r3/17-069r3.html) [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html] standard. The TC is composed of individuals representing organizations that are duly recognized members in good standing of the OGC [1]. The main function of the TC is to provide an open, collaborative forum for professional discussion related to the consensus development and/or evaluation, approval, and revision of OGC international standards. The pilot also explored the draft OGC API - Tiles and [OGC API - Styles](https://github.com/opengeospatial/ogcapi-styles) [https://github.com/opengeospatial/ogcapi-styles] specifications. Feature and tile servers that implement these APIs were deployed. The data encodings used in the pilot included Mapbox Vector Tiles (MVT), GeoJSON, and GeoPackage.

Whereas the feature and tile servers allowed the pilot participants to explore online support for vector tiles, GeoPackage allowed the pilot participants to explore offline support. The use of GeoPackage potentially offers applications a container that could carry tiled datasets in a portable

fashion that allows use when an application is offline.

The VTP2 initiative concluded that an architecture that enables consistent online and offline support of vector tiles can be implemented using the suite of draft and approved OGC API specifications and the GeoPackage standard. This was demonstrated by the successful implementation of the VTP2 pilot architecture, which included implementations of the OGC API - Features standard, OGC API - Tiles draft specification, OGC API – Styles draft specification and the OGC GeoPackage standard.

The following recommendations, which are described in more detail in [Section 12](#), were identified as a result of the work completed in the VTP2 initiative:

1. OGC API - Tiles and OGC API - Features should be extended to allow for publishing schemas.
2. The Styled Layer Descriptor (SLD) Standards Working Group (SWG), by virtue of being responsible for the OGC Symbology Conceptual Core Model, should introduce the additional concepts: sprite, stylable layer set, stylesheet, style encoding, symbol, and symbol content.
3. A standard for Tile Set Metadata should be developed by an appropriate SWG.
4. OGC API - Tiles should provide direction on multi-layer vector tiles schemas, ideally as an optional extension.
5. OGC API - Tiles should allow for multi-layer tiles that are without explicit collections.
6. TileJSON editors should register a media type for the specification.
7. OGC API - Common should allow for collections that do not advertise their content types.
8. The originators of the TileJSON and MVT specifications should be encouraged to submit the specifications into the OGC for consideration as OGC Community Standards.

[1] While it is preferable to remove the attributes from the tiles when using the Vector Tiles Attributes Extension, it is not mandatory to do so. If the GeoPackage producer is not capable of modifying the vector tiles to remove the attributes, it is acceptable to leave them in place.

[2] For consistency with other implementations, it is expected that the `gpkext_vt_fields` table will still be populated when this extension is in use.

Chapter 2. Introduction

This OGC Engineering Report (ER) provides a summary of the research and findings from Phase 2 of the OGC Vector Tiles Pilot (VTP2). The goal of VTP2 was to deliver a consistent, interoperable online/offline architecture for vector tiles based on feature and tile servers, as well as GeoPackage. All Application Programming Interface (API) implementations and service types deployed in the pilot were implemented to support the prototype vector tile metadata model and filtering language. These were two essential work items of VTP2. The feature and tile servers included implementations of the OGC API – Features standard and the draft OGC API – Tiles specification. The feature and tile servers provided support for a variety of Coordinate Reference Systems (CRS). This ER provides an overview of each of the components, their implementation decisions and the challenges faced.

The VTP2 participants intend to use the results of the work in VTP2 to inform the development of OGC APIs, GeoPackage, and web service standards to enable consistent use both online and offline, particularly in DDIL environments. Such consistent use of tiled feature data online and offline will improve interoperability and usability of geospatial applications. Therefore, the value of the VTP2 work to organizations is expected to be in the efficiencies and productivity that comes from greater interoperability and usability.

2.1. Requirements & Research Motivation

The motivation for performing the VTP2 pilot was the increasing need to enable consistent access to geospatial resources such as tiled feature data (i.e. vector tiles), styles and metadata in networked environments through Web APIs, and in DDIL environments using GeoPackage.

A detailed description of the requirements was presented in the [Call for Participation](https://portal.ogc.org/files/89870) [https://portal.ogc.org/files/89870]. In brief, the requirements addressed by the work presented in this Summary ER were to:

- Design a metadata model that can describe fundamental aspects of the tiles, such as date, creator, source, tiling scheme (tile matrix set), space partitions, and styles;
- Design a vector tiles filtering language, with filters similar to OGC Filter Encoding 2.0 or CQL (Common Query Language), and with a degree of consistency between GeoPackage, tile and feature servers;
- Develop solutions for robust offline/online usage scenarios using GeoPackage technology and develop methods to associate vector tile tables in GeoPackages with attribute tables;
- Develop online/offline symbol and style sharing for vector tiles e.g. a user can request pre-rendered tiled vector data by defining a particular style while online. As network connectivity is lost, the user can then request data from the GeoPackage and render the GeoPackage content client-side;
- Evaluate GeoPackages with vector tile data for the ability to support portrayal and attributes in an offline environment;
- Design methods to store symbols in easily accessible online locations such as the Styles API and Amazon S3 ‘buckets’ and offline resource folders or GeoPackage tables;

- Design methods to retrieve and apply style information based on the emerging OGC API - Styles specification; and
- Conduct experimentation involving tests of vector tiles using the [WorldMercatorWGS84Quad](http://www.opengis.net/def/tilematrixset/OGC/1.0/WorldMercatorWGS84Quad) [http://www.opengis.net/def/tilematrixset/OGC/1.0/WorldMercatorWGS84Quad] and [WorldCRS84Quad](http://www.opengis.net/def/tilematrixset/OGC/1.0/WorldCRS84Quad) [http://www.opengis.net/def/tilematrixset/OGC/1.0/WorldCRS84Quad] tile matrix sets referenced to the [EPSG:3395](http://www.opengis.net/def/crs/EPSSG/0/3395) [http://www.opengis.net/def/crs/EPSSG/0/3395] and [EPSG:4326](http://www.opengis.net/def/crs/EPSSG/0/4326) [http://www.opengis.net/def/crs/EPSSG/0/4326] Coordinate Reference Systems respectively.

2.2. Recommendations

In addition to the [recommendations](#) listed in the Executive Summary, the following were identified by the participants as areas for future work in OGC Innovation Program initiatives such as pilots and testbeds:

- Exploring the use of Collections (some of which has already started in the SWGs)
- Variable Width Tile matrices (two examples being [GNOSISGlobalGrid](http://schemas.opengis.net/tms/1.0/json/examples/GNOSISGlobalGrid.json) [http://schemas.opengis.net/tms/1.0/json/examples/GNOSISGlobalGrid.json] and [CDBGlobalGrid](http://maps.ecere.com/geoapi/tileMatrixSets/CDBGlobalGrid) [http://maps.ecere.com/geoapi/tileMatrixSets/CDBGlobalGrid]). One of the reasons for looking into variable width tile matrices is that currently one needs four Tile Matrix Sets (TMS) to cover the Earth pole-to-pole, when using current non-WebMercator TMS. It would therefore be useful to have a Tile Matrix Set that can cover the Earth pole-to-pole but that has better accuracy than WebMercator.
- Use cases that do not only focus on visualization, such as the ability to re-construct geometry in support of data analysis. In this regard, a compact format that can be streamed, for example [FlatGeoBuff](https://github.com/bjornharrtell/flatgeobuf) [https://github.com/bjornharrtell/flatgeobuf], could be investigated. For Mapbox vector tiles, future use cases could include data extraction that is geared towards analysis and editing rather than visualization and interactivity.
- Implementation of tiled data as part of processing workflows (e.g. in parallel and/or distributed processing environments).
- Better specification of the CRS used in tiled GeoJSON (CRS84 vs the tile matrix set's own CRS) needs to be analyzed.
- Experimentation with FlatGeobuf should occur. FlatGeobuf is a variation of geobuf geared towards simple features and focusing on performance and compactness.
- Development of beta compliance tests for OGC API - Tiles, OGC API - Styles, OGC API - Maps, and others that are needed. The compliance tests would be implemented as executable test suites for running in TEAM Engine - OGC's validator.
- Testing larger datasets

2.3. Document contributor contact points

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

Contacts

Name	Organization	Role
Gobe Hobona	OGC	Editor
Terry Idol	OGC	Editor
Jeff Harrison	AGC	Contributor
Jeff Yutzler	Image Matters	Contributor
Clemens Portele	interactive instruments	Contributor
Sergio Taleisnik	Skymantics	Contributor
Logan Stark	Skymantics	Contributor
Andrea Aime	GeoSolutions	Contributor
Stefano Bovio	GeoSolutions	Contributor
Jérôme Jacovella-St-Louis	Ecere Corporation	Contributor
Patrick Dion	Ecere Corporation	Contributor
Carl Reed	Carl Reed & Associates	Contributor

2.4. 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 12-080r2, OGC OWS Context Conceptual Model 1.0 Standard (2014) [https://portal.opengeospatial.org/files/?artifact_id=55182]
- OGC: OGC 12-128r15, OGC GeoPackage 1.2.1 Standard (2018) [<https://www.geopackage.org/spec120/index.html>]
- OGC: OGC 07-057r7, OGC® OpenGIS Web Map Tile Service Implementation Standard (2010) [http://portal.opengeospatial.org/files/?artifact_id=35326]
- OGC: OGC 17-069r3, OGC API - Features - Part 1: Core (2019) [<http://docs.opengeospatial.org/is/17-069r3/17-069r3.html>]
- OGC: OGC 05-078r4, Styled Layer Descriptor, Version 1.1 (2007) [http://portal.opengeospatial.org/files/?artifact_id=22364]
- OGC: OGC 15-120r5, Volume 0: Primer for the OGC CDB Standard: Model and Physical Data Store Structure (2018) [<https://portal.opengeospatial.org/files/15-120r5>]
- OGC: OGC 15-113r5, Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure (2018) [<https://portal.opengeospatial.org/files/15-113r5>]
- OGC: OGC 16-005r3, Volume 2: OGC CDB Core: Model and Physical Structure: Informative Annexes (2018) [<https://portal.opengeospatial.org/files/16-005r3>]
- IETF: RFC-7946 The GeoJSON Format (2016) [<https://tools.ietf.org/html/rfc7946>]

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) [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- **portrayal**
the act of rendering digital data into a visual form
- **queryable**
a property that can be queried
- **<portrayal> sprite**
an image representing a symbol; multiple sprites may be stored together in a collection as sub-images
- **stylable layer set**
a collection of styles designed to be used within the same domain
- **style**
a sequence of rules of symbolizing instructions to be applied by a rendering engine on one or more features and/or coverages
- **style encoding**
specification to express a style as one or more files
- **stylesheet**
representation of a style in a machine-readable form
- **style metadata**
essential information about a style in order to support users in discovering and selecting styles for rendering their data and for visual style editors to create user interfaces for editing a style
- **symbol**
a graphical representation of a concept that is rendered during portrayal
- **symbol content**
representation of a symbol in machine-readable form
- **symbol encoding**
specification to express one or more sprites in one or more files
- **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).

NOTE

For the purposes of this OGC ER, a tile is a small rectangular representation of geographic data, often part of a set of such elements, covering a tiling scheme and sharing similar information content and graphical styling. A tile can be uniquely defined in a tile matrix by one integer index in each dimension. Tiles are mainly used for fast transfer (particularly in the web) and easy display at the resolution of a rendering device. Tiles can be grid based pictorial representations, coverage subsets, or feature based representations (e.g., vector tiles).

• tile matrix

a grid tiling scheme that defines how space is partitioned into a set of conterminous tiles at a fixed scale.

NOTE

A tile matrix constitutes a tessellation of the space that resembles a matrix in a 2D space characterized by a matrix width (columns) and a matrix height (rows).

• tile matrix set

a tiling scheme composed by collection of tile matrices defined at different scales covering approximately the same area and has a common coordinate reference system.

• tile set

set of tiles - a collection of subsets of the space being partitioned.

NOTE

For the purposes of this OGC ER, a tile is a series of actual tiles contain data and following a common tiling scheme.

• tiling scheme

a scheme that defines how space is partitioned into individual tiled units. A tiling scheme defines the spatial reference system, the geometric properties of a tile, which space a uniquely identified tile occupies, and reversely, which unique identifier corresponds to a space satisfying the geometric properties to be a tile.

NOTE

A tiling scheme is not restricted to a coordinate reference system or a tile matrix set and allows for other spatial reference systems such as DGGS and other organizations including irregular ones.

• vector tile

a tile that contains vector information that has been simplified at the tile scale resolution and clipped at the tile boundaries.

• Web API

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

NOTE

See [Best Practice 24: Use Web Standards as the foundation of APIs](https://www.w3.org/TR/dwbp/#APIHttpVerbs) [https://www.w3.org/TR/dwbp/#APIHttpVerbs] (W3C Data on the Web Best Practices) for more detail.

4.1. Abbreviated terms

- **API** Application Programming Interface
- **CRS** Coordinate Reference System
- **DDIL** Denied, Degraded, Intermittent or Limited connectivity networks
- **OGC** Open Geospatial Consortium
- **WFS** Web Feature Service
- **WMS** Web Map Service
- **WMTS** Web Map Tile Service
- **VT** Vector Tiles, Vector Tiling, Vectiles

Chapter 5. Overview

[Section 6](#) presents the context for the pilot, including the scenario and use cases.

[Section 7](#) presents the architecture and outlines the standards and draft specifications used by the pilot.

[Section 8](#) presents the Vector Tiles GeoPackage Model implemented in the VTP2 pilot.

[Section 9](#) describes the products that were deployed by testbed participants in order to implement the architecture described in Section 7.

[Section 10](#) presents the results of the pilot, including considerations and technology integration experiments.

[Section 11](#) presents the findings of the pilot.

[Section 12](#) presents the recommendations and ideas for future work.

[Appendix A](#) presents GeoPackage extensions proposed by the VTP2 initiative.

[Appendix B](#) presents additional screenshots for some of the components deployed by the VTP2 participants.

[Appendix C](#) presents the revision history of this document.

[Appendix D](#) presents the bibliography.

Chapter 6. Context

The sets of tiled feature data organized through vector tiling are colloquially referred to as vector tiles. Vector tiles may consist of one or more layers and may be accompanied by styling information that allows a client application to render the tiled feature data consistently across tiles. By virtue of containing feature data, vector tiles allow client applications to modify how geographic features are presented. This contrasts with map tiles which are generated through pre-rendering on the server-side thereby preventing client-side modification of the presentation. Consequently, client applications can adapt vector tiles to display device characteristics such as resolution and color-hue management. Client applications can also adapt requests for vector tiles sent to servers to consider bandwidth constraints. These are some of the benefits of vector tiles.

The primary goal of Phase 1 of the Vector Tiles Pilot (VTP1) was to define candidate extensions to existing OGC standards as a way to advance the use of tiled feature data as part of the OGC standards baseline. More specifically, VTP1 participants developed proofs of concept for extending OGC API - Features (at the time called WFS3), WMTS and GeoPackage. An extension to VTP1 developed a set of possible extensions to GeoPackage 1.2 that provide mechanisms for storing and retrieving vector tiles in a GeoPackage. The data encodings used in VTP1 and its extension included Mapbox Vector Tiles (MVT) and GeoJSON. VTP1 applied a use case on the visualization of feature data on a client. The three main scenarios considered by VTP1 were consumption of tiled feature data by a web client, a desktop client and a mobile client [2].

VTP2 had the goal of delivering a consistent, interoperable online/offline architecture consisting of feature and tile servers, and GeoPackage-producing components that could publish, display and query vector tiles.

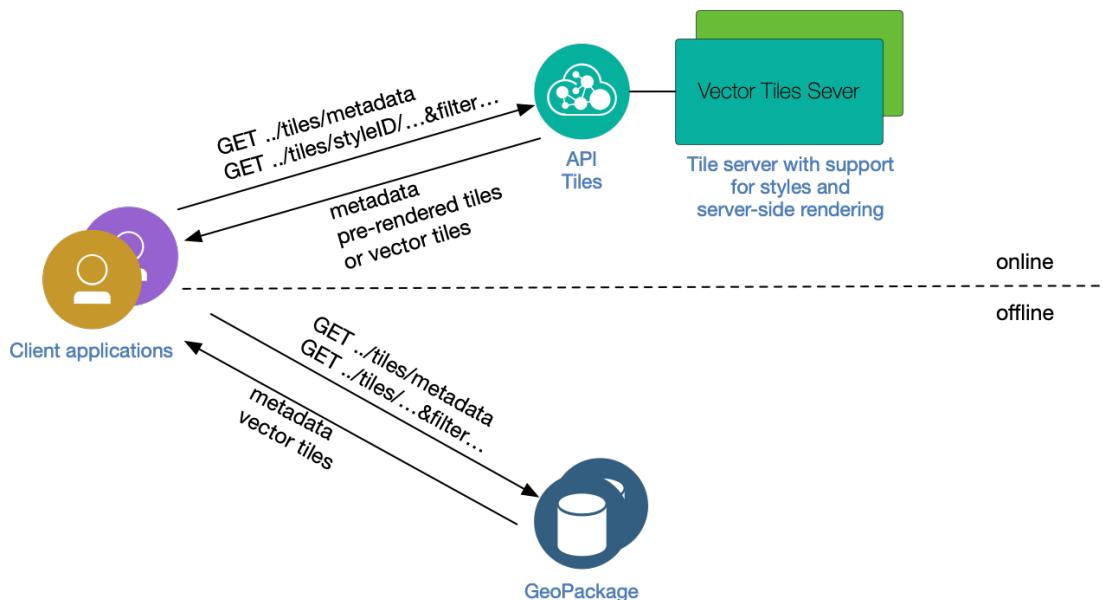


Figure 1. VTP2 scenario

6.1. Scenario

The VTP2 Initiative implemented the scenario described below.

A truck driver is tasked with delivering humanitarian supplies to a refugee camp in a country with

limited or intermittent communications infrastructure.

Before departing from the dispatch center, the truck driver prepares the maps and data they will need for navigating to the refugee camp. From a collection of tiled feature data for the entire world, the truck driver acquires the subset of tiles covering the route to the camp and the area around the camp. The truck driver excludes the tiles and feature attributes that have no information. The truck driver also excludes data from more than 6 months ago. Having acquired the tiled feature data, the truck driver then stores the data in a portable external drive that they can carry with them.

6.2. Use Cases

Vector Tiles can be used to support several different use cases. Taking the truck driver mentioned in the scenario above as an end-user, this section presents use cases addressed by the participants of the VTP2 initiative.

6.2.1. Filtering Use Case

An end-user has a very large data set with more than 700 different feature types, the end-user requires Filters and Filter Builders to constrain the types of features that go into the vector tile sets that may be used offline and overlaid on raster tile imagery.

The Filters and Filter Builders are used to constrain the types of features, remove attributes that consistently have 'no information', set the bounding box or area, indicate a date/time range and select output formats (MVT and/or GeoJSON). The filter would then be sent to the API to generate vector tile sets that may be used offline and overlaid on raster tile imagery.

6.2.2. Metadata Use Case

Tile Set Metadata is used to describe the content of the tile sets received from the API. The end user requires access to the tile sets in a complete offline environment and utilizing tile set metadata to quickly review the main elements and information about those tile sets without having to parse each tile of the tile set.

Chapter 7. Architecture

To support the objectives of this initiative, the following types of components were built by the participants: feature servers, tile servers, GeoPackage producers, and Client Applications. The feature servers were exposed through implementations of OGC API - Features, whereas the tile servers were exposed through implementations of OGC API - Tiles. The following subsections describe each of the components. [Figure 2](#) shows the architecture designed for the pilot.

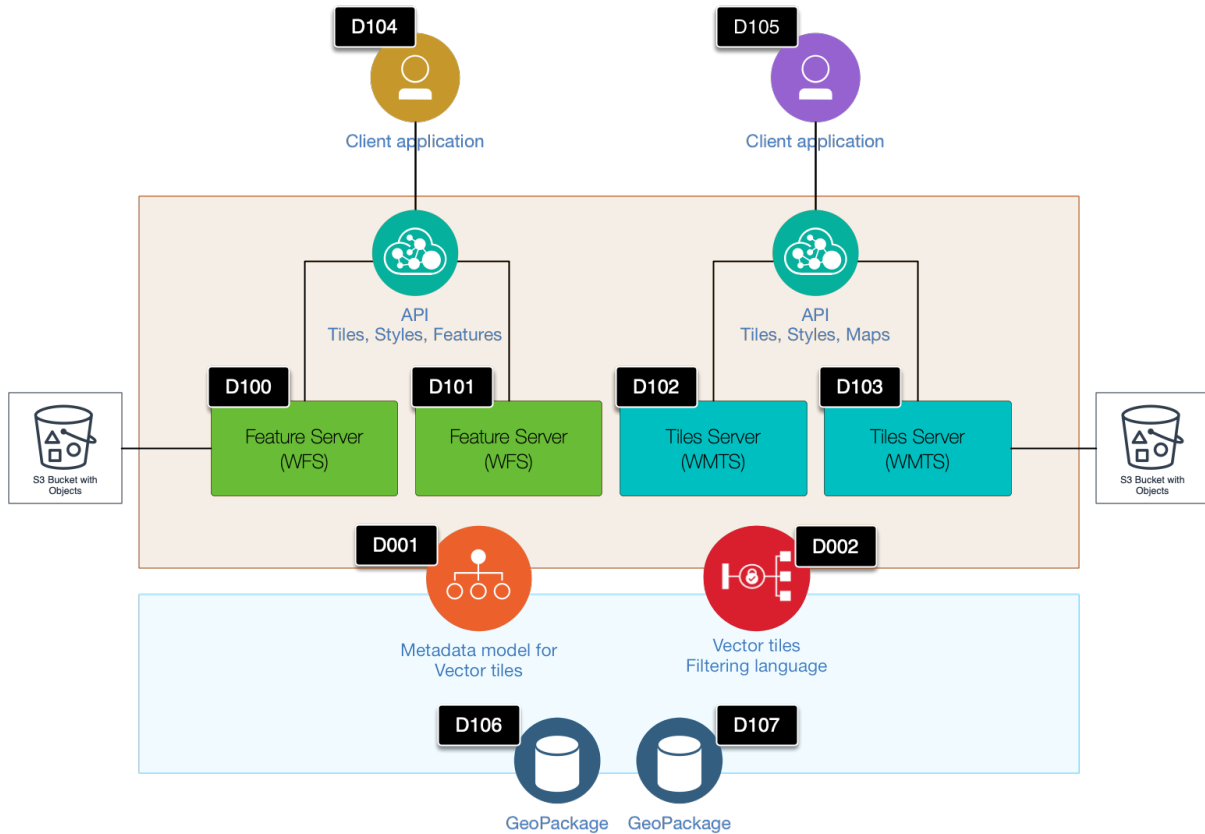


Figure 2. The OGC Vector Tiles Pilot Phase 2 architecture

The components shown in [Figure 2](#) are based on the standards and draft specifications described in the following subsections.

7.1. OGC API - Features standard

The OGC API - Features standard specifies API building blocks to create, modify and query features on the Web. Part 1 of the standard specifies the core capabilities and is restricted to fetching features where geometries are represented in the WGS 84 Coordinate Reference System with axis order longitude-latitude. The core specifies mandatory capabilities that every implementing service has to support and is restricted to read-access to geospatial data. Amongst the capabilities is support for collections through a `/collections` path. The collections capabilities specified in the standard describe requirements only for collections consisting of features. That is, each collection considered by the standard is a feature collection. Further, each feature in a dataset is part of exactly one collection. At the time of writing this report, development of additional capabilities that address specific needs (e.g. [support for different CRS](http://docs.opengeospatial.org/DRAFTS/18-058.html) [http://docs.opengeospatial.org/DRAFTS/18-058.html]) is well underway. Other envisaged future capabilities include, for example, support for creating and modifying data, more complex data models, and richer queries. The OGC API - Features standard

builds on the Web Feature Service (WFS) standard and has previously been referred to as WFS 3.0.

7.2. OGC Web Feature Service (WFS) standard

The OGC Web Feature Service (WFS) standard specifies a service interface for discovery operations, query operations, locking operations, transaction operations for data and operations to manage stored, parameterized query expressions. Discovery operations allow the service to be interrogated to determine its capabilities and to retrieve the application schema that defines the feature types that the service offers. Query operations allow features or values of feature properties to be retrieved from the underlying data store based upon constraints, defined by the client, on feature properties. Locking operations allow exclusive access to features for the purpose of modifying or deleting features. Transaction operations allow features to be created, changed, replaced and deleted from the underlying data store. Stored query operations allow clients to create, drop, list and described parameterized query expressions that are stored by the server and can be repeatedly invoked using different parameter values.

7.3. OGC Web Map Tile Service (WMTS) standard

The OGC Web Map Tile Service (WMTS) standard specifies a service interface for providing digital maps using predefined image tiles. The purpose of a WMTS service is to serve maps divided in individual tiles, thereby improving performance and scalability. The standard allows an implementing service to advertise the tiles it has available through a standardized declaration common to all OGC web services. This declaration defines the tiles available in each layer (i.e. each type of content), in each graphical representation style, in each format, in each coordinate reference system, at each scale, and over each geographic fragment of the total covered area.

7.4. OGC GeoPackage standard

The OGC GeoPackage standard describes an open, standards-based, platform-independent, portable, self-describing, compact format for transferring geospatial information. The standard describes a set of conventions for storing the following within a SQLite database:

- Vector features
- Tile matrix sets of imagery and raster maps at various scales
- Extensions for handling specific types of resources such as tiled gridded coverage data and related tables

To be clear, a GeoPackage is the SQLite container and the GeoPackage Encoding Standard governs the rules and requirements of content stored in a GeoPackage container. The GeoPackage standard defines the schema for a GeoPackage, including table definitions, integrity assertions, format limitations, and content constraints. The required and supported content of a GeoPackage is entirely defined in the standard.

7.5. OGC API - Maps draft specification

The OGC API - Maps draft specification describes an API that can serve spatially referenced and

dynamically rendered electronic maps [3]. The specification describes the discovery and query operations of an API that provides access to electronic maps in a manner independent of the underlying data store. The query operations allow dynamically rendered maps to be retrieved from the underlying data store based upon simple selection criteria, defined by the client. The OGC API - Maps draft specification builds on the Web Map Service (WMS) standard (OGC 06-042 [http://portal.opengeospatial.org/files/?artifact_id=14416]).

7.6. OGC API - Tiles draft specification

The OGC API - Tiles draft specification describes an API building block that can enable other OGC API implementations to serve maps or tiled feature data divided into individual tiles [4]. The draft specification includes concepts originating from the WMTS standard although such concepts have been revised to allow for tiling other types of resources, not only maps. The OGC API - Tiles draft specification references the OGC Two Dimensional Tile Matrix Set (TMS) standard (OGC 17-083r2 [<http://docs.opengeospatial.org/is/17-083r2/17-083r2.html>]). The TMS standard defines the rules and requirements for a tile matrix set as a way to index space based on a set of regular grids defining a domain (tile matrix) for a limited list of scales in a CRS.

7.7. VTP2 Filtering Language proof of concept

Within the context of the architecture described in Figure 2 the initiative participants sought to design a filtering language to support the publication and use of vector tiles, within the context of the architecture described in Section 7. The filtering language is described in detail in the *OGC Vector Tiles Pilot 2: Vector Tiles Filtering Language Engineering Report* [5].

7.8. VTP2 Metadata model proof of concept

Within the context of the architecture described in Figure 2 the initiative participants sought to design a metadata model to support the publication and use of vector tiles, within the context of the architecture described in Section 7. Using the NSG Metadata Foundation (NMF), the metadata model was designed to describe stored tile caches in sufficient detail to allow usage and updating of a tile cache without needing to analyze each individual tile in the cache. The NMF defines the conceptual schema profile for specifying geospatial metadata in and for the US National System for Geospatial Intelligence (NSG) [6]. The metadata model is described in detail in the *OGC Vector Tiles Pilot 2: Tile Set Metadata Engineering Report* [7].

Chapter 8. Vector Tiles GeoPackage Model

The VTP2 CFP states a requirement for robust online/offline use scenarios based on the GeoPackage Standard. This includes the following:

- A metadata model for the tiles, tile matrix set, and styles;
- Methods to associate vector tile tables in GeoPackages with attribute tables to allow applying similar filters to GeoPackages as being applied to the OGC API - Tiles;
- Online/offline symbol and style sharing.

The following subsections describe how this information can be stored in a GeoPackage. [Appendix A](#) documents draft GeoPackage extensions to support the approaches described here.

8.1. Vector Tiles

The following demonstrates how tiled feature data, such as vector tiles, can be inserted directly into a GeoPackage and have the attributes remain embedded in the vector tiles. This approach requires the least amount of server-side processing to produce the GeoPackage. As illustrated in [Figure 3](#), an application would apply the following process:

1. Ensure required tables are present:
 - `gpkg_extensions` as per the [GeoPackage Extension Mechanism](http://www.geopackage.org/spec121/#_extensions) [http://www.geopackage.org/spec121/#_extensions]
 - `gpkgext_vt_layers` as per `gpkgext_vt_layers`
 - `gpkgext_vt_fields` as per `gpkgext_vt_fields`
2. Populate `gpkg_extensions` with:
 - references to all tables mentioned above as per [gpkg_extensions Table Rows](#)
 - reference to the [MVT Extension](#) and/or [GeoJSON Extension](#) as needed
3. Add a row to `gpkg_contents` for each user-defined vector tiles table with a `data_type` of "vector-tiles".
4. Add a row to `gpkgext_vt_layers` for each vector tiles layer, referencing the `table_name` from `gpkg_contents`.
5. Add a row to `gpkgext_vt_fields` for each field referenced in the vector tiles, referencing the `layer_id` from `gpkgext_vt_layers`.
6. Add a row to the user-defined vector tiles table for each vector tile.

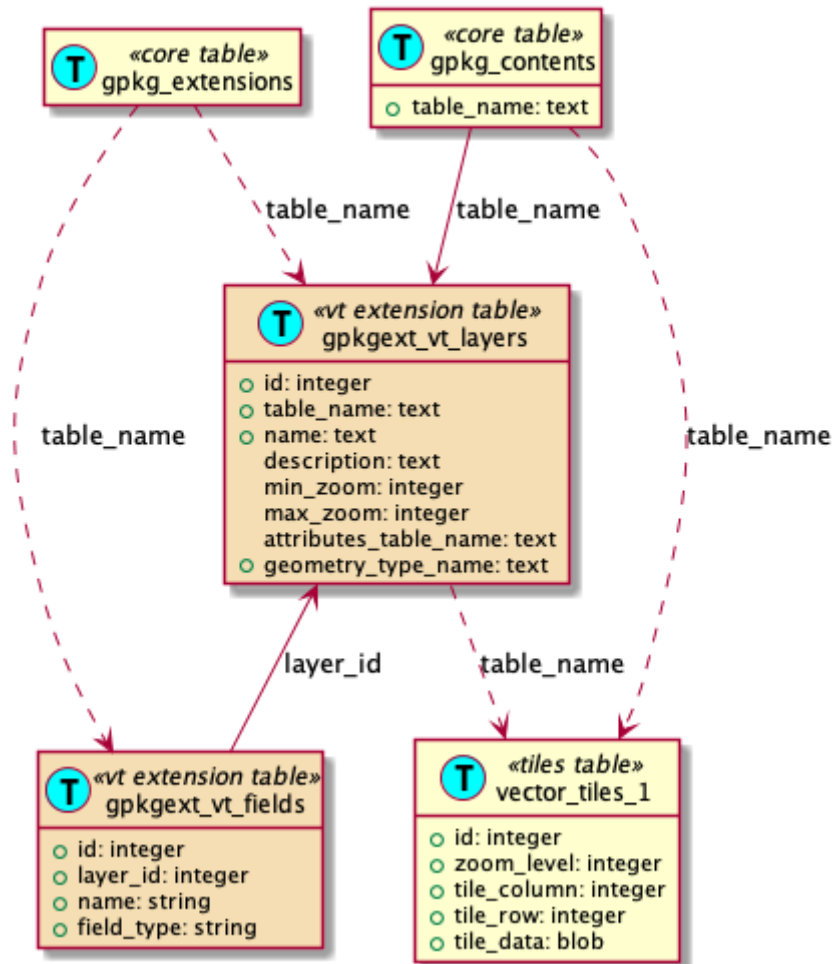


Figure 3. Vector Tiles with Embedded Attributes

NOTE

The need for the `gpkgext_vt_layers.geometry_type_name` column was identified at the end of the Pilot. This column was not in place during the TIEs.

8.2. Portrayal Information

This section describes extending the direct insertion of tiled feature data to provide the portrayal information (styles and symbols) needed to render the tiled data properly. In this approach, the client must couple the layers to the stylesheets. As illustrated in Figure 4, the process is as follows:

1. Ensure required tables are present as per [GeoPackage Portrayal Extension](#):
 - `gpkgext_styles`
 - `gpkgext_stylesheets`
 - `gpkgext_symbols`
 - `gpkgext_symbol_content`
2. Populate `gpkg_extensions` with references to all tables mentioned above.
3. Add a row to `gpkgext_stylesheets` for each stylesheet capable of rendering the layer.
4. Add a row to `gpkgext_styles` for each named style.

5. Add a row to `gpkgext_symbols` for each symbol used in the stylesheets.
6. Add a row to `gpkgext_symbol_content` for each file containing symbol data.
7. Add a row to `gpkgext_symbol_images` for each image, referencing rows in `gpkgext_symbols` and `gpkgext_symbol_content` and containing sprite information if needed.

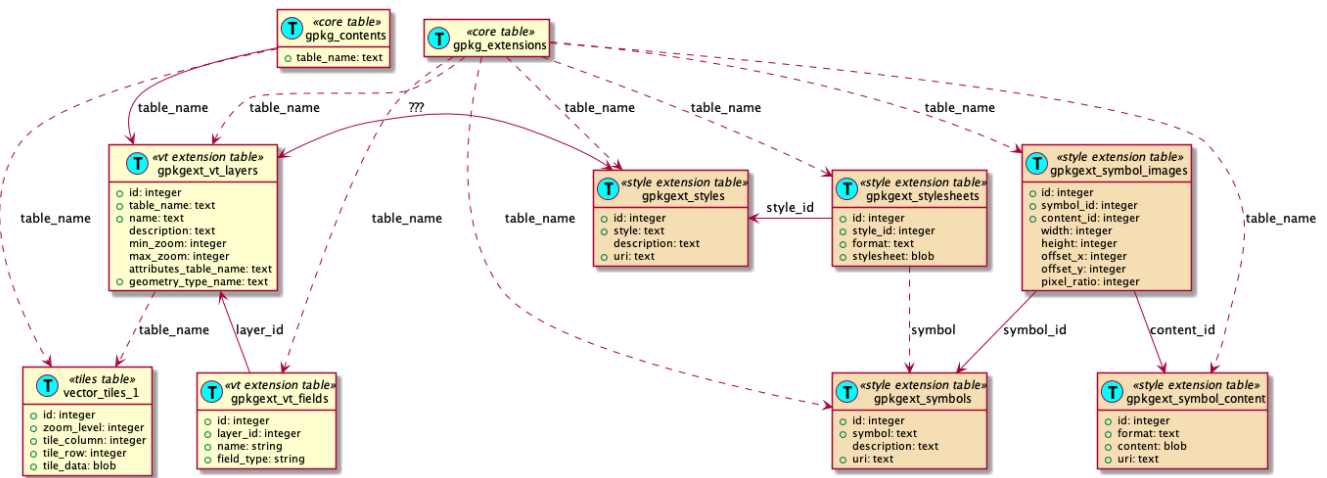


Figure 4. GeoPackage Portrayal with Vector Tiles

8.3. Metadata

This section is concerned with the metadata aspects of the Vector Tiles GeoPackage model. A wider discussion relating to metadata work conducted in VTP2 is presented in the *OGC Vector Tiles Pilot 2: Tile Set Metadata Engineering Report* [7]. The [GeoPackage Metadata Extension](http://www.geopackage.org/guidance/extensions/metadata.html) [http://www.geopackage.org/guidance/extensions/metadata.html] is used to add metadata for any business object (layer, feature, tile, style, symbol, etc.) to contents in a GeoPackage.

NOTE

All of the caveats regarding [metadata profiles](https://docs.opengeospatial.org/dp/19-047.html#_metadata_profiles) [https://docs.opengeospatial.org/dp/19-047.html#_metadata_profiles] apply here.

Through the following process, metadata is added to any GeoPackage business object:

1. Ensure required tables are present as per the [GeoPackage Metadata Extension](http://www.geopackage.org/spec121/#extension_metadata) [http://www.geopackage.org/spec121/#extension_metadata]
 - `gpkg_metadata`
 - `gpkg_metadata_reference`
2. Populate `gpkg_extensions` with references to all tables mentioned above.
3. Add a row to `gpkg_metadata` for each metadata document.
4. Add a row to `gpkg_metadata_reference` for each business object described by a metadata document, referencing the corresponding table name, column name, and row identifier.

Figure 5 illustrates a GeoPackage with style and symbol metadata.

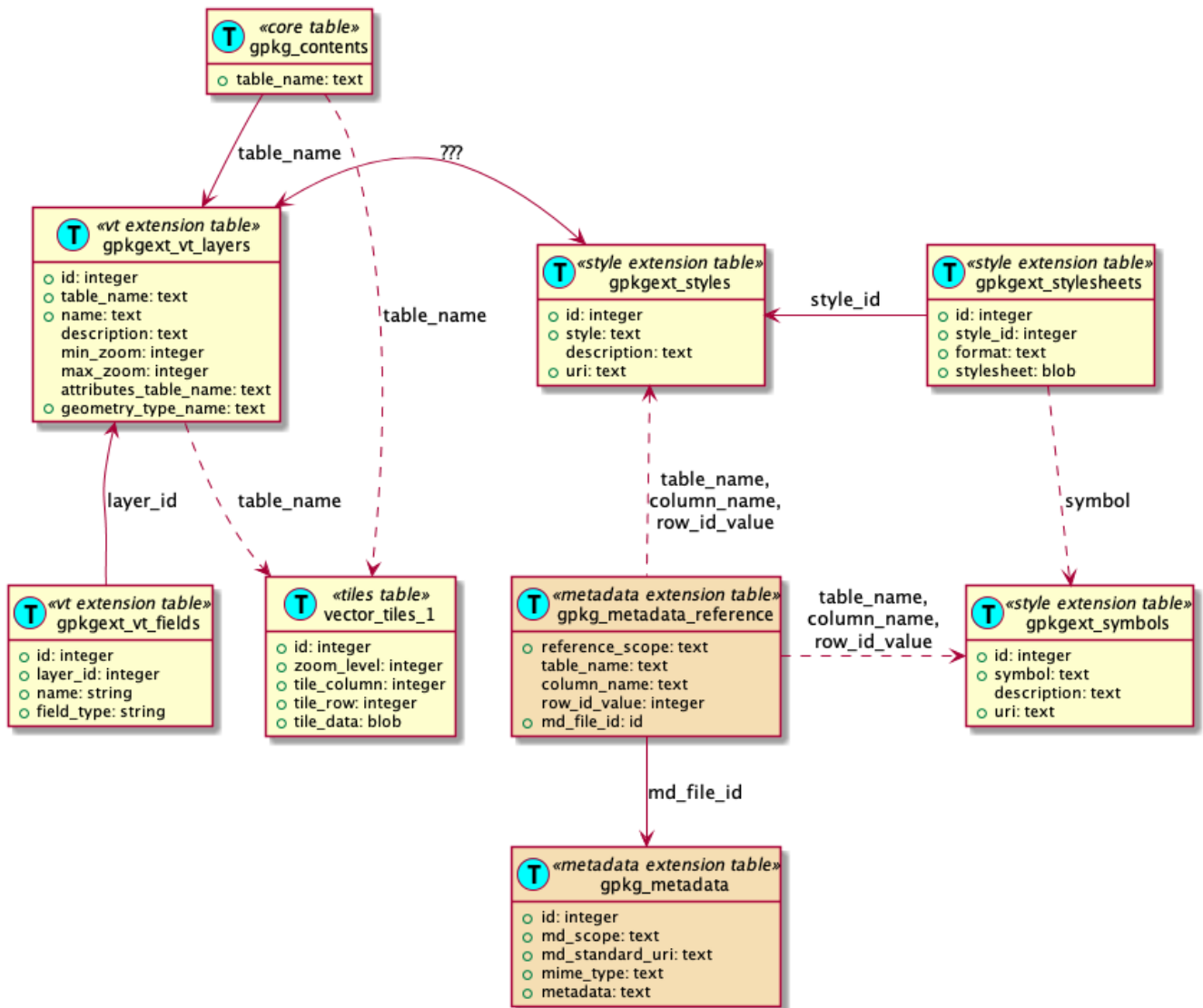


Figure 5. Vector Tiles with Styles, Symbols, and Style Metadata

8.4. Semantic Annotations

Semantic annotations provide a way to represent the meaning of a feature that is to be portrayed. Such annotations are resolvable via a URI and can be placed on any business object (layer, feature, tile, style, etc.). Semantic annotations are implemented in a GeoPackage through the [Semantic Annotations Extension](#). As illustrated in [Figure 6](#), semantic annotations may be placed on virtually any row in the GeoPackage. (Some tables already have URIs and when that is the case, the URIs are shared for consistency.)

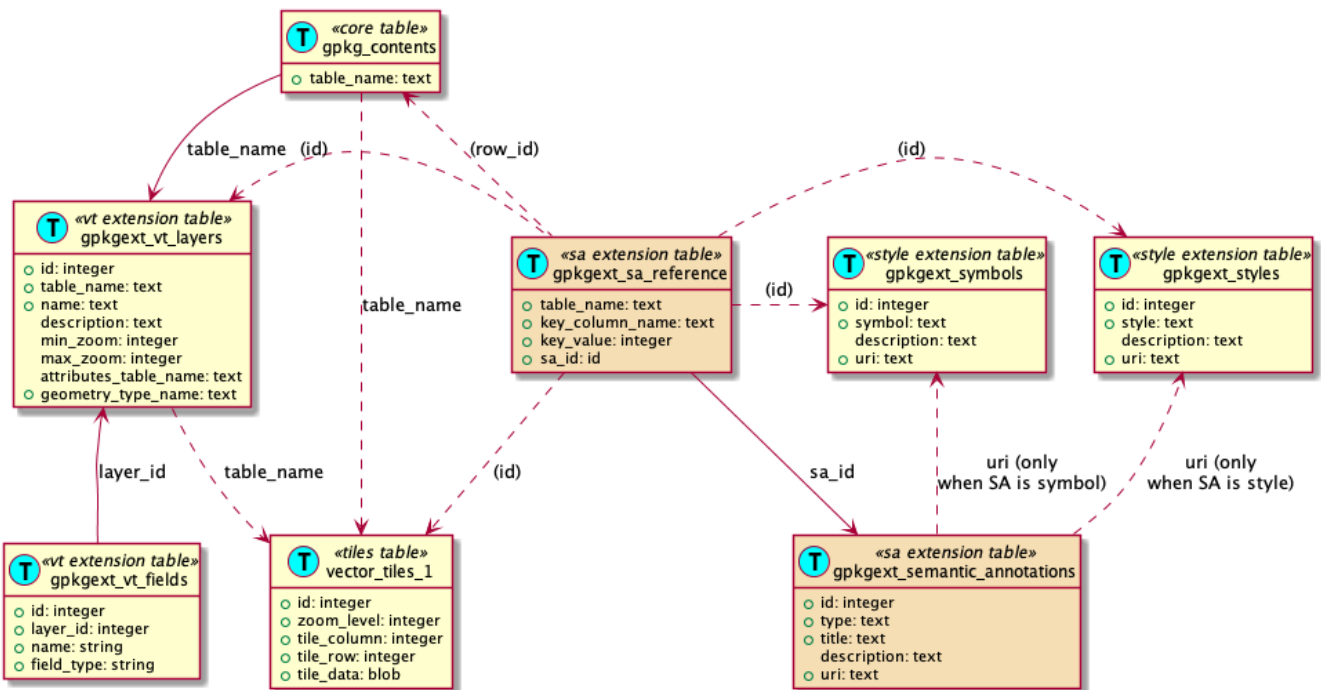


Figure 6. Semantic Annotations for GeoPackage business objects

To establish semantic annotations in a GeoPackage, the process is as follows:

1. Ensure required tables are present as per the [Semantic Annotations Extension](#):
 - `gpkgext_sa_reference`
 - `gpkgext_semantic_annotations`
2. Populate `gpkg_extensions` with references to all tables mentioned above.
3. Add a row to `gpkgext_semantic_annotations` for every semantic annotation. (See below for the specific *type* to use.)
4. Add a row to `gpkgext_sa_reference` for every row that must be annotated.

In [Portrayal Information](#), there is no explicit coupling between layers and styles. Coupling, in this context, refers to association of styles with layers. Coupling is completely the responsibility of the user and/or client application. In many scenarios, it is beneficial for the GeoPackage to explicitly declare the coupling. In VTP2, this was done through semantic annotations. There are two ways that this can be done.

styles

The URI in `gpkgext_semantic_annotations` (the same as in `gpkgext_styles`) allows conventional layers or vector tiles to be annotated as suitable for use with that style. See an example in [Table 4](#).

stylable layer sets

A new URI allows conventional layers, vector tiles layers, and styles to be annotated as belonging to the same stylable layer set. See an example in [Table 6](#).

Both approaches are described below through a tested example. Both worked examples start with the following common base information.

Table 1. *gpkg_contents*

rowid	table_name	data_type
0	basemap	tiles
1	tiles_daraa	vector-tiles
2	overlays	features

Table 2. *gpkgext_styles*

id	style	description	uri
3	night	Night style for OSMTDS	'http://geosolutions.com/styles/osmtids-night/'
4	topographic	Topographic style for OSMTDS	'http://geosolutions.com/styles/osmtids-topographic/'

Table 3. *gpkgext_vt_layers*

id	table_name	name
13	tiles_daraa	agricultureSrf
14	tiles_daraa	settlementSrf

8.4.1. Styles

In this example, the two styles ("night" and "topographic") are shared by the vector tiles and the overlays. First, two semantic annotations are created (one for each style).

Table 4. *gpkgext_semantic_annotations*

id	type	title	description	uri
23	style	night	Night style for OSMTDS	'http://geosolutions.com/styles/osmtids-night/'
24	style	topographic	Topographic style for OSMTDS	'http://geosolutions.com/styles/osmtids-topographic/'

Then, the layers (both conventional and vector tiles) and the styles are tagged with those annotations.

Table 5. *gpkgext_sa_reference*

table_name	key_column_name	key_value	sa_id
gpkg_contents	rowid	2	23
gpkgext_vt_layers	id	13	23

table_name	key_column_name	key_value	sa_id
gpkgext_vt_layers	id	14	23
gpkg_contents	rowid	2	24
gpkgext_vt_layers	id	13	24
gpkgext_vt_layers	id	14	24
gpkgext_styles	id	3	23
gpkgext_styles	id	4	24

NOTE

The fact that the styles are also stored in the GeoPackage is secondary. It is not mandatory to do so, but when the architecture calls for it, the styles and semantic annotations will have the same URI.

8.4.2. Stylable Layer Sets

In this example, the two styles are part of the same stylable layer set that works for both the vector tiles and the overlays. First, a semantic annotation is created for the stylable layer set.

Table 6. *gpkgext_semantic_annotations*

id	type	title	description	uri
33	StylableLayerSet	OSMTDS	stylable layer set for OpenStreetMap TDS	'http://opengis/stylableLayerSets/OSMTDS'

Then, the layers and styles are all tagged with this annotation.

Table 7. *gpkgext_sa_reference*

table_name	key_column_name	key_value	sa_id
gpkgext_vt_layers	id	13	33
gpkgext_vt_layers	id	14	33
gpkg_contents	rowid	2	33
gpkgext_styles	id	3	33
gpkgext_styles	id	4	33

8.5. Attributes

As described previously, attribute information is typically embedded in with the content contained in vector tiles. However, 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, which is 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, this section presents an alternate approach for managing attributes in a way that mitigates all three issues. Through the [Vector Tiles Attributes Extension](#), a GeoPackage may contain an [attributes table](#) [<https://www.geopackage.org/spec121/#attributes>] for each vector tiles layer.

To use this approach, the application should apply the process as illustrated by [Figure 7](#):

1. Ensure required tables are present:
 - an attributes table, preferably one with bounding box values `min_x`, `min_y`, `max_x`, and `max_y` (to be used in spatial indexing as described below)
2. Populate `gpkg_extensions` with all required references, as per:
 - the [RTree Spatial Indexes Extension](http://www.geopackage.org/guidance/extensions/rtree_spatial_indexes.html) [http://www.geopackage.org/guidance/extensions/rtree_spatial_indexes.html] (optional)
3. Add a row to `gpkg_contents` for each user-defined attributes table.
4. Populate the attributes table with the attributes from the vector tiles.
5. Remove the attributes from the vector tiles. (optional) ^[3]
6. Populate the `attributes_table_name` row of the `gpkgext_vt_layers` table with the name of the attributes table. ^[4]
7. Establish an R-Tree spatial index on the attributes table based on the bounding box attributes in the attributes table. (optional)

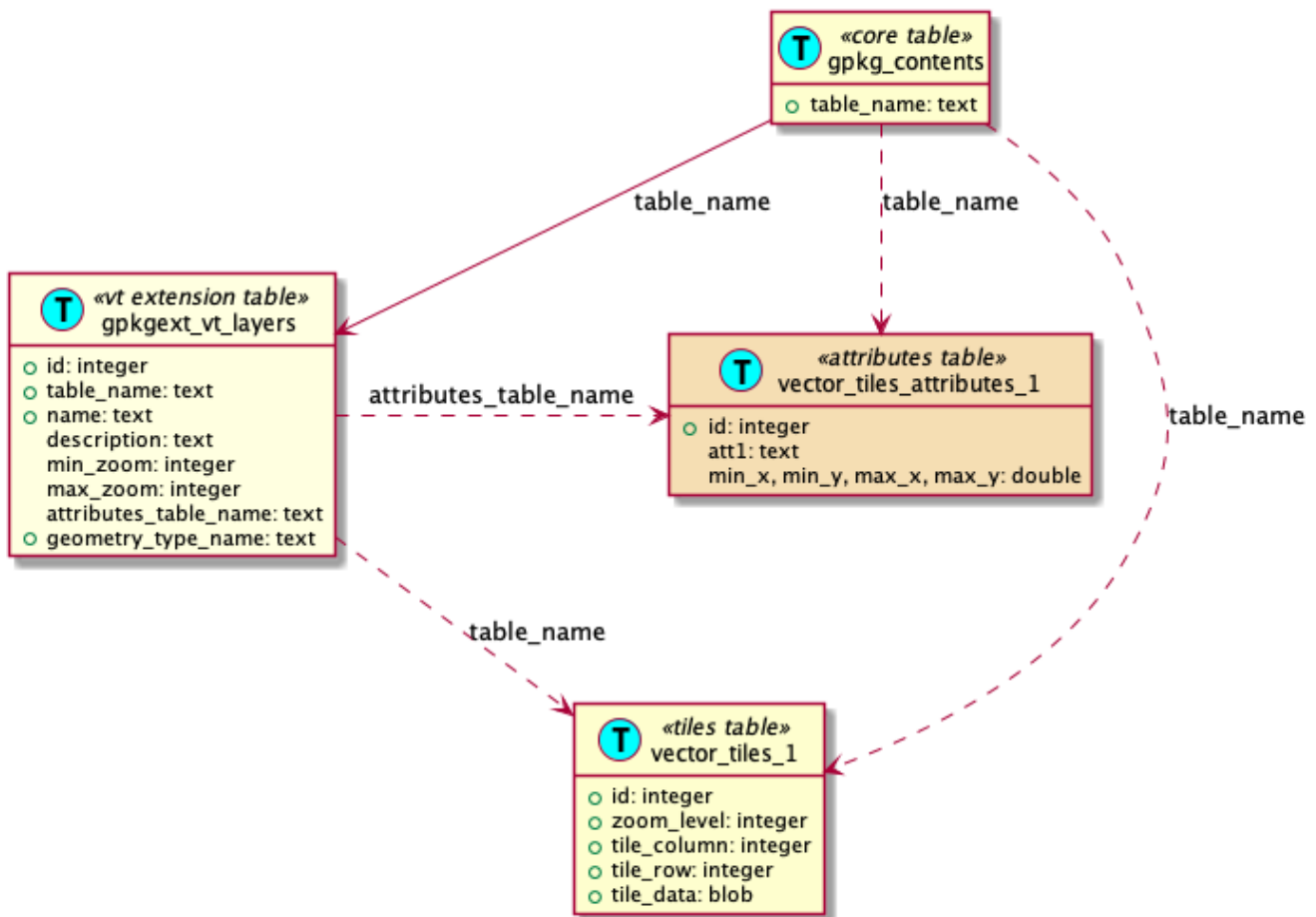


Figure 7. GeoPackage Vector Tiles with Relational Attributes

8.6. Attributes with Related Tables

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. The [GeoPackage Related Tables Extension](http://www.geopackage.org/guidance/extensions/related_tables.html) [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.

To use this approach, the process is as illustrated by [Figure 8](#):

1. Ensure required tables are present:
 - `gpkgext_relations` as per the Related Tables Extension
 - a [Related Tables Extension mapping table](http://www.geopackage.org/guidance/extensions/related_tables.html#user-defined-mapping-table) [http://www.geopackage.org/guidance/extensions/related_tables.html#user-defined-mapping-table] for each tile set - attributes table combination
2. Populate `gpkg_extensions` with all required references, as per:
 - the [Related Tables Extension](http://docs.opengeospatial.org/is/18-000/18-000.html#) [http://docs.opengeospatial.org/is/18-000/18-000.html#

gpkg_extensions_records]

- [this extension](#)

3. Add a row to `gpkgext_relations` for each tiles table / attributes table combination, naming the corresponding mapping table.
4. Add a row to the mapping table for each feature-tile combination.

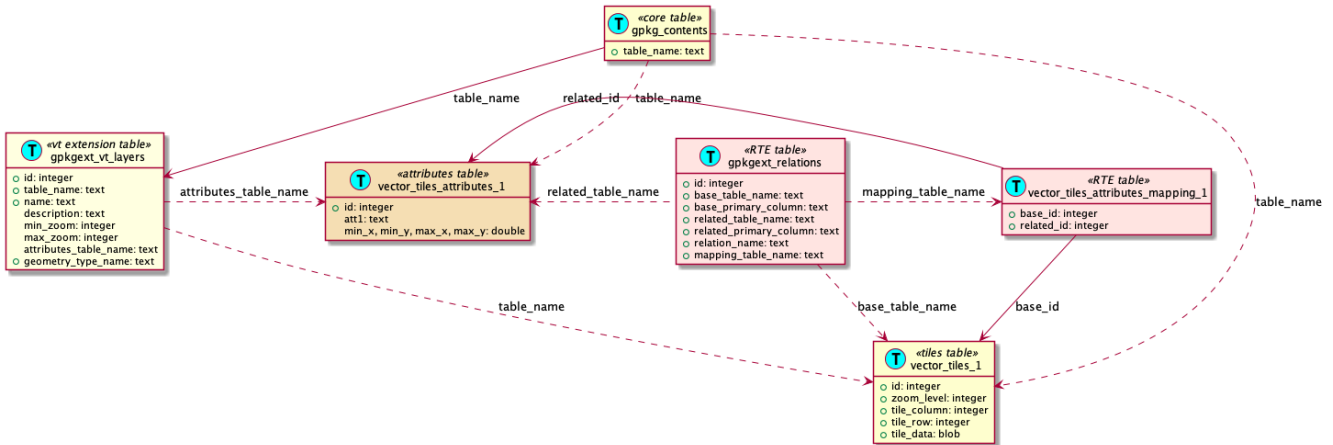


Figure 8. GeoPackage Vector Tiles Attribute Extension

8.7. Tile Matrix Sets

The adoption of the GeoPackage Encoding Standard predated the adoption of the [OGC Tile Matrix Set \(TMS\) Standard](http://docs.opengeospatial.org/is/17-083r2/17-083r2.html) [http://docs.opengeospatial.org/is/17-083r2/17-083r2.html]. The TMS Standard introduces tile matrix sets as first class business objects where in GeoPackage they were treated as attributes of tile pyramids. Allowing multiple tile pyramids to share common tile matrix sets improves consistency and eliminates the redundancy of copying tile matrix information for each tile pyramid. In addition, the TMS Standard introduces an option for variable-width tile matrix sets. The [GeoPackage Tile Matrix Set Extension](#) implements both concepts in GeoPackage.

To use this approach, the process is as illustrated by [Figure 9](#):

1. Add a row to `gpkgext_tile_matrix_set` for every tile matrix set used in the GeoPackage.
2. Add rows to `gpkgext_tile_matrix` for each tile matrix set, describing the individual zoom levels.
3. If variable-width tile matrixes are in use, add rows to `gpkgext_tile_matrix_variable_widths` for all cases where the coalescence coefficient is greater than 1.
4. Add a row to `gpkgext_tile_matrix_tables` for every tiles table, referencing the tile matrix set in use.

Chapter 9. Implementations

9.1. Services and Data Producers

This section describes the technologies and approaches taken for implementing the services and data producing components of the architecture.

9.1.1. Ecere D103 Features, Tiles and Styles API

Ecere improved the service implementation of the Features, Tiles and Styles APIs deployed during OGC Testbed-15, with new capabilities such as support for new tiling schemes, filtering and TileJSON tileset metadata. This instance of the GNOSIS Map Server was used to serve multiple datasets, including imagery, elevation data and OpenStreetMap vector data, by pilot participants in Technology Integration Experiments. The landing page of the server is shown in [Figure 10](#).

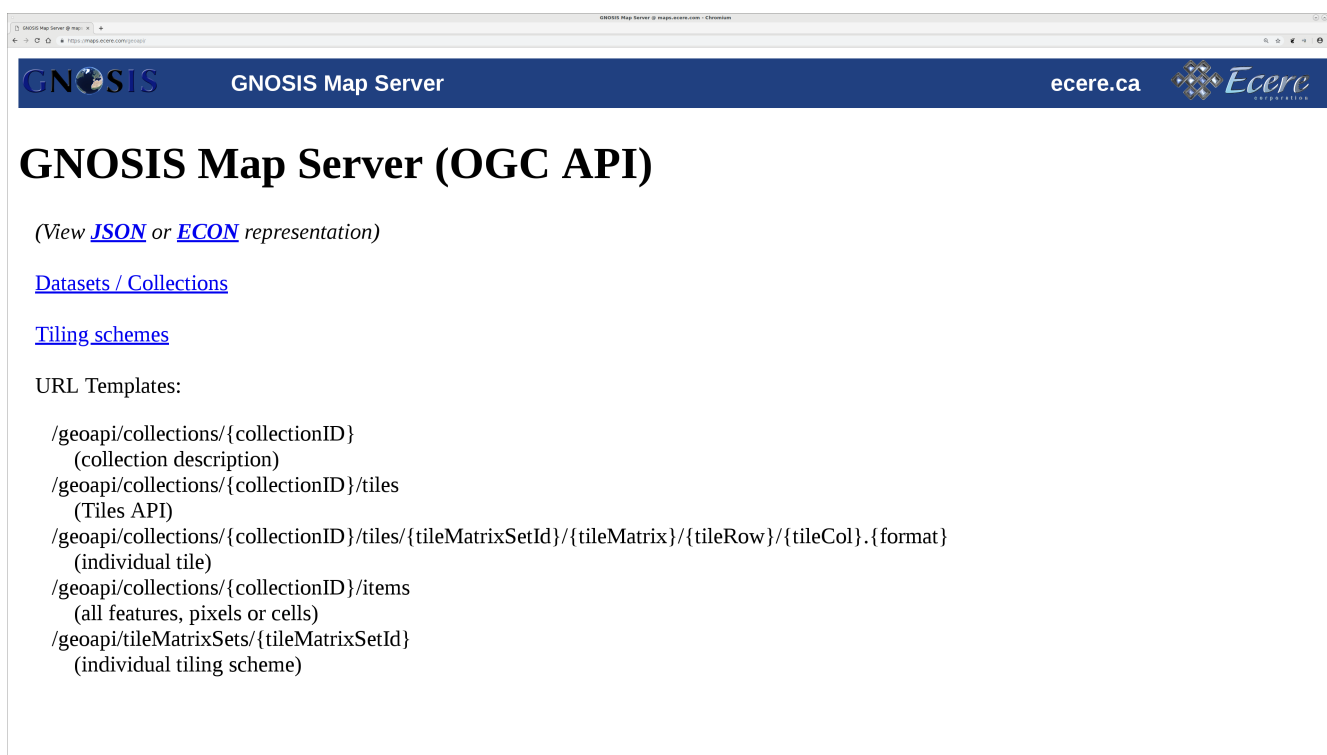


Figure 10. Landing page for GNOSIS Map Server

The server dynamically sources data from multiple tiled data stores, which can be either GNOSIS Data Stores or GeoPackages, but could eventually also be external OGC APIs for cascading servers, as well as chained rendering and processing. A screenshot showing the layers and styles served from a data source is presented in [Figure 11](#).



Figure 11. Vector and elevation layers (collections) and styles served from a GeoPackage

Requests are not constrained to the nature of the source data, as the server can re-project to another coordinate reference system, re-tile to a different tiling scheme, re-encode to another format and/or re-assemble vector features or images on-the-fly.

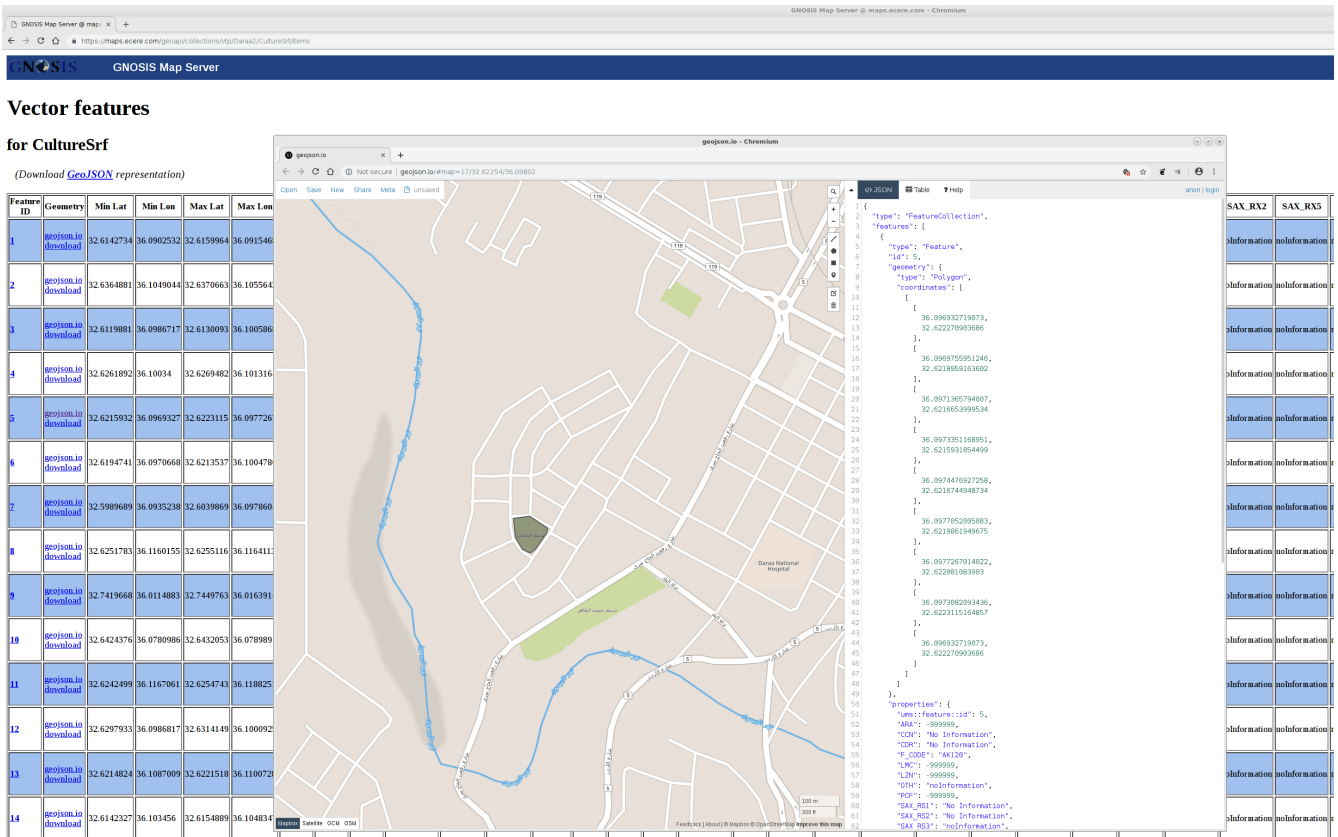


Figure 12. Cultural Surfaces from Daraa (vector features, available whole, tiled, or clipped to BBOX)

Tiles API

GPKG/Ecere/Daraa2-SingleLayer-GlobalGrid.gpkg/CultureSrf

(View [JSON](#) or [ECON](#) representation)

Tile URL template:
 /geopi/collections/GPKG/Ecere/Daraa2-SingleLayer-GlobalGrid.gpkg/CultureSrf/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}.{format}

Supported formats:
 gmt (GNOSIS Map Tiles)
 mvt (Mapbox Vector Tiles)
 json (GeoJSON)
 econ (GeoECON)
 gml (GML)
 mapml (MapML)

Supported tiling schemes:

Tiling Scheme	TMS definition	TileJSON Metadata
CDBGlobalGrid	TMS definition	TileJSON Metadata
GlobalCRS84Pixel	TMS definition	TileJSON Metadata
GlobalCRS84Scale	TMS definition	TileJSON Metadata
GNOSISGlobalGrid	TMS definition	TileJSON Metadata
GoogleCRS84Quad	TMS definition	TileJSON Metadata
WebMercatorQuad	TMS definition	TileJSON Metadata
WorldCRS84Quad	TMS definition	TileJSON Metadata
WorldMercatorWGS84Quad	TMS definition	TileJSON Metadata

Figure 13. The Tiles API for cultural surfaces served from a GeoPackage

The server supports encoding vector features in multiple encodings:

- GeoJSON
- Mapbox Vector Tiles
- GNOSIS Map Tiles (developed on Testbed 13)

- MapML (developed on Testbed 15)

A previous iteration of the service also supported the Geography Markup Language (GML), as well as GeoECON (similar to GeoJSON but based on [eC Object Notation](http://ec-lang.org/econ) [http://ec-lang.org/econ]), but these formats were not yet functional in the latest iteration at the conclusion of the pilot.

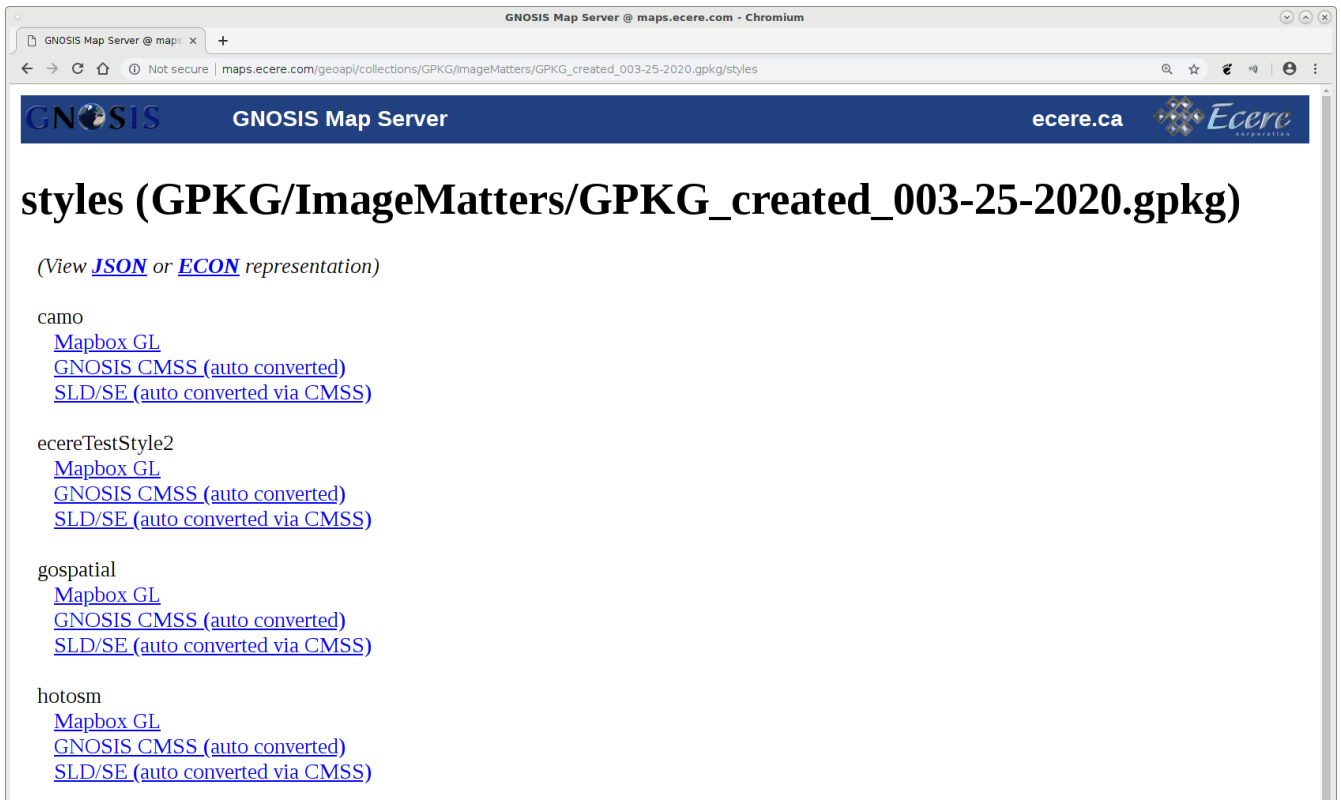


Figure 14. Styles being served directly from GeoPackage portrayal extensions

The Styles API provides a list of available styles suitable for portraying a particular dataset. The server performs (as best as it can manage) translation between supported styles encoding — GNOSIS Cascading Map Style Sheet (CMSS), SLD/SE, and MapboxGL styling — using the native CMSS as a bridge between SLD/SE and MapboxGL. CMSS is a geospatial symbology profile of ECCSS (eC Cascading Style Sheets, intended for additional uses such as styling user interfaces), and was originally developed as an encoding for the Symbology Conceptual Model developed in OGC Testbed-14. When requesting a style specifically for a single collection, the style will also be filtered to only include the rules pertaining to that particular collection. Support for uploading new styles or modifying existing styles was not yet supported at the conclusion of the pilot.

GNOSIS Map Server

ecere.ca

Vector features

for ne_10m_geography_regions_elevation_points

(Download [GeoJSON](#) representation)

Feature ID	Geometry	Min Lat	Min Lon	Max Lat	Max Lon	scalerank	featurecla	name	elevation	comment	name_alt	lat_y	long_x	nation1	nation2	note	region	subregion
431	geojson.io/download	36.5382844	74.5604378	36.5382844	74.5604378	7	mountain	Batura Mustagh I	7795			36	74				Asia	
457	geojson.io/download	30.5424778	79.9709704	30.5424778	79.9709704	5	mountain	Nanda Devi	7817			30	79				Asia	
463	geojson.io/download	28.6964616	83.4951355	28.6964616	83.4951355	3	mountain	Dhaulagiri	8172			28	83				Asia	
466	geojson.io/download	27.9804805	86.880596	27.9804805	86.880596	2	mountain	Mt. Everest	8848	Worlds highest point		27	86				Asia	
470	geojson.io/download	35.8824059	76.513209	35.8824059	76.513209	2	mountain	K2	8611			35	76				Asia	
477	geojson.io/download	36.2500097	71.8399843	36.2500097	71.8399843	9	mountain	Tirich M?r	7708			36	71	PK		From PeakBagger.com, with kind permission.	Asia	
486	geojson.io/download	35.2399779	74.5900045	35.2399779	74.5900045	9	mountain	Nanga Parbat	8125			35	74	PK		From PeakBagger.com, with kind permission.	Asia	
494	geojson.io/download	38.5900957	75.3098632	38.5900957	75.3098632	9	mountain	Kongkoer	7649			38	75	CN		From PeakBagger.com, with kind permission.	Asia	
496	geojson.io/download	27.7000821	88.1500225	27.7000821	88.1500225	9	mountain	Kanchenjunga	8586			27	88	IN	NP	From PeakBagger.com, with kind permission.	Asia	
611	geojson.io/download	29.6429978	95.0657068	29.6429978	95.0657068	7	mountain	Namcha Barwa	7782			29	95				Asia	
658	geojson.io/download	29.59578	101.8791196	29.59578	101.8791196	3	mountain	Gongga Shan	7556			29	101				Asia	
687	geojson.io/download	28.0499944	90.459936	28.0499944	90.459936	9	mountain	Gangkar Punsum	7570			28	90	BT	CN	From PeakBagger.com, with kind permission.	Asia	

Figure 15. Only highest elevation points returned from CMSS filtering expression

Filtering support was implemented for both the Tiles API and Features API, and is described in detail in the VTP2 Filtering Language Engineering Report. Support for comparison of attribute values, geometry intersection with a bounding box, as well as logical and arithmetic operations was added. The expressions syntax from the CMSS styling language, as used within selectors or the values of styling properties of a symbolizer, is what could be achieved during the pilot. Support for the CQL filtering language is a capability planned to be added eventually.

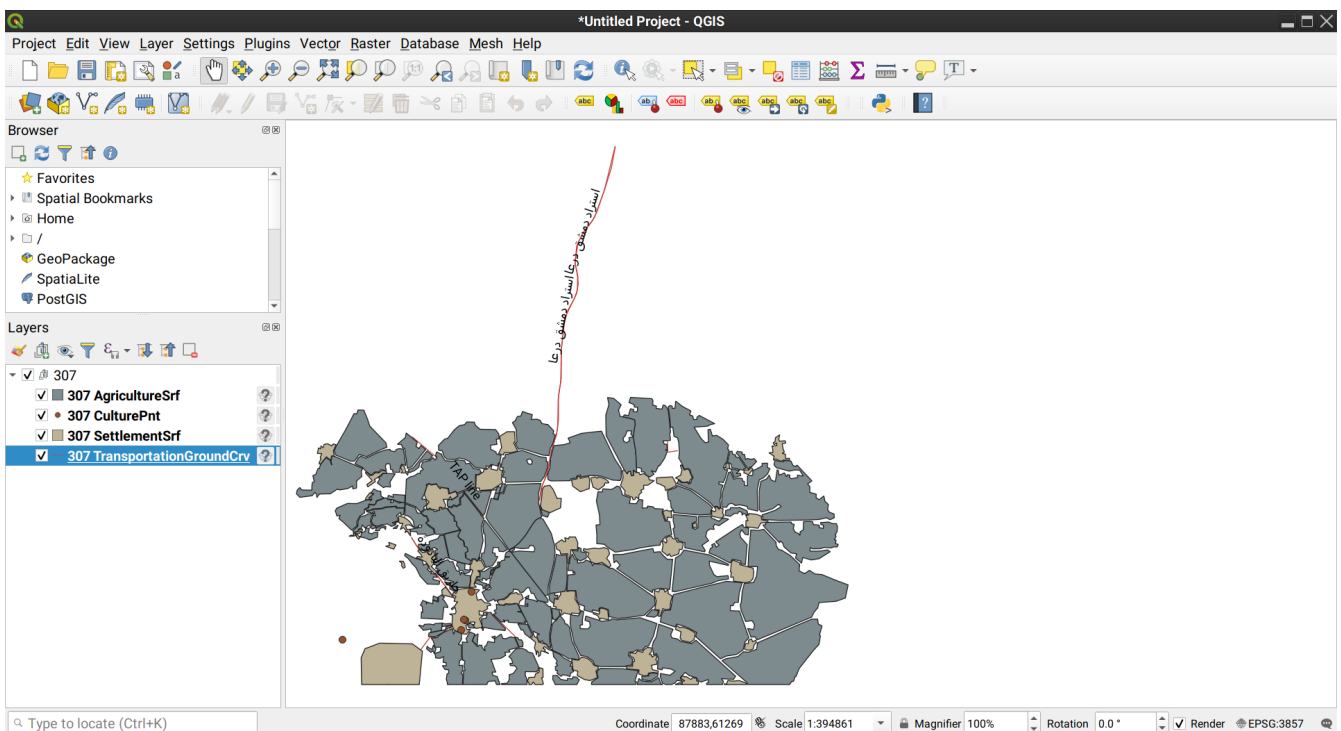


Figure 16. Multi-layer CMSS filter applied to a tile displayed in QGIS

With CMSS, it was possible to experiment with multi-layer filtering. The ability to select specific layers to include within multi-layer tiles makes it possible to greatly reduce the size of the tiles

produced. This can also be done in conjunction with a scale-based selector, making it possible to re-use the same tiles template for all zoom levels.

The new support for TileJSON tileset metadata conveys key information such as the layers found within a tileset, the fields for attribute information, the vector geometry type, the zoom levels as well as a simple URL template for retrieving the tiles themselves. This information is particularly useful for describing tiles consisting of multiple layers, as no other standard mechanism had yet been defined to do so with the Tiles API. Providing this metadata in a TileJSON format also enables compatibility with the rich set of tools available in the Mapbox ecosystem.

Ecere's implementation of the Tiles API already supported the GlobalCRS84Scale, GlobalCRS84Pixel, GoogleCRS84Quad, WorldCRS84Quad, and WebMercatorQuad tiling schemes.

(View [JSON](#) or [ECON](#) representation)

Identifier	CRS	Type	Widths	Level 0				Level 1					
				Scale	Rows	Columns	Width	Height	Scale	Rows	Columns	Width	Height
CDBGlobalGrid	EPSG:4326	geographic	variable	1:388,252	180	360	1024	1024	1:194,126	360	720	1024	1024
GlobalCRS84Pixel	EPSG:4326	geographic	fixed	1:795,139,220	1	1	256	256	1:397,569,610	1	2	256	256
GlobalCRS84Scale	EPSG:4326	geographic	fixed	1:500,000,000	1	2	256	256	1:250,000,000	2	3	256	256
GNOSISGlobalGrid	EPSG:4326	geographic	variable	1:139,770,566	2	4	256	256	1:69,885,283	4	8	256	256
GoogleCRS84Quad	EPSG:4326	geographic	fixed	1:559,082,264	1	1	256	256	1:279,541,132	1	2	256	256
WebMercatorQuad	EPSG:3857	projected	fixed	1:559,082,264	1	1	256	256	1:279,541,132	2	2	256	256
WorldCRS84Quad	EPSG:4326	geographic	fixed	1:279,541,132	1	2	256	256	1:139,770,566	2	4	256	256
WorldMercatorWGS84Quad	EPSG:3395	projected	fixed	1:559,082,264	1	1	256	256	1:279,541,132	2	2	256	256

Figure 17. Supported tiling schemes

During the pilot, support for three additional schemes was implemented:

- WorldMercatorWGS84Quad: a quad-tree based on EPSG:3395 (World Mercator, using a proper WGS84 ellipsoid),
- CDBGlobalGrid (an alias for CDB Global Grid): a multi-part quad-tree based based on EPSG:4326, and representing both the OGC CDB Levels of Detail and CDB Zones, also using variable width (defined in [OGC 15-113r5](https://portal.opengeospatial.org/files/15-113r5) [https://portal.opengeospatial.org/files/15-113r5] and [OGC 16-005r3](https://portal.opengeospatial.org/files/16-005r3) [https://portal.opengeospatial.org/files/16-005r3]). All tile matrices from level 0 and higher are defined to be 1024x1024 pixels. The CDBGlobalGrid also has negative zoom levels (down to -10), for which instead of dividing space differently, each tile maintains the same geographic bounds, but gets divided into fewer pixels (each dimension reduced by 2 at the next smaller level).
- GNOSISGlobalGrid (an alias for GNOSIS Global Grid): an almost-quad-tree based on EPSG:4326, with variable widths to approximate equal area tiles, as now defined by the OGC Two Dimensional Tile Matrix Set standard. See [Figure 18](#) and [Figure 19](#) for an illustration of support for this scheme. The standard introduces two constraints compared to Ecere's previous implementation of the GNOSIS Global Grid: all columns with the same number should align regardless of a row's variable width (with different columns sometimes representing the same tile), and rows should always be numbered from the top (north).

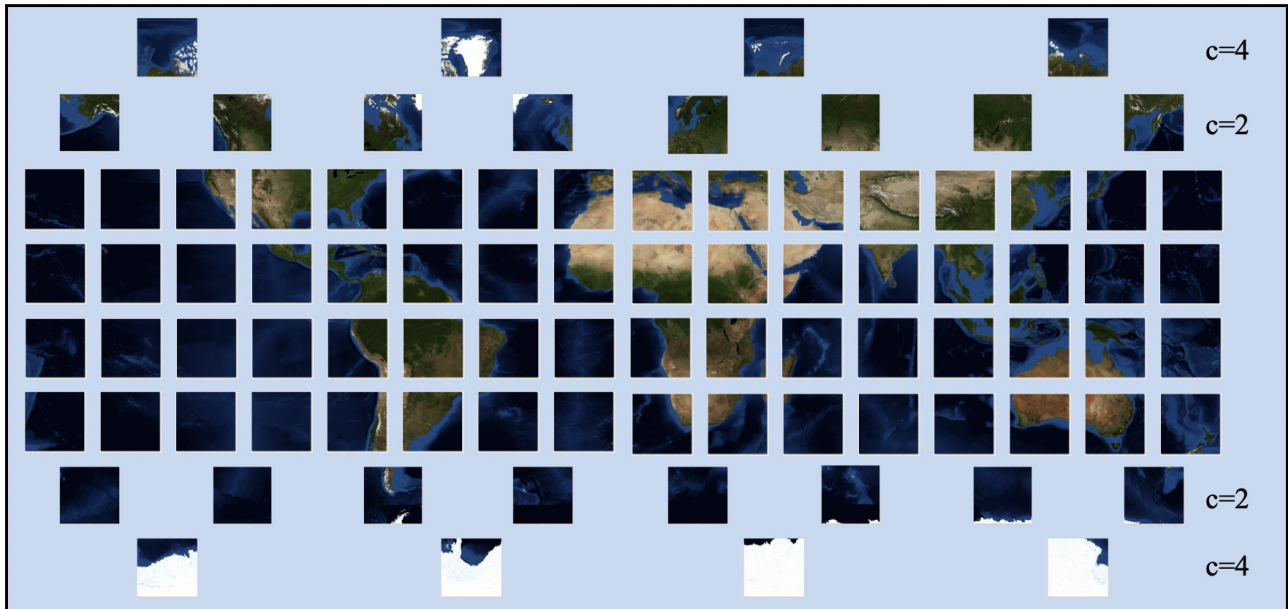


Figure 18. Level 2 of the GNOSIS Global Grid, shown with variable widths coalescence coefficients

GNOSIS Map Server

GNOSISGlobalGrid

(View [JSON](#) or [ECON](#) representation)

CRS: [EPSG:4326](#) (geographic)

Zoom Level	Scale	Rows Count	Columns (c=1)	Width (px)	Height (px)	Meters / pixel	Degrees / pixel	Top-Left Corner (lat, lon)	Delta Latitude	Delta Longitude (c=1)	Variable Width Rows Coalescing Coefficient (c)
0	1:139,770,566.007	2	4	256	256	39,135.75848201024	21.09375°	90, -180	90°	90°	
1	1:69,885,283.004	4	8	256	256	19,567.87924100512	10.546875°	90, -180	45°	45°	Row 0: 2 Row 3: 2
2	1:34,942,641.502	8	16	256	256	9783.93962050256	5.2734375°	90, -180	22.5°	22.5°	Row 0: 4 Row 1: 2 Row 6: 2 Row 7: 4
3	1:17,471,320.751	16	32	256	256	4891.96981025128	2.63671875°	90, -180	11.25°	11.25°	Row 0: 8 Row 1: 4 Rows 2..3: 2 Rows 12..13: 2 Row 14: 4 Row 15: 8
4	1:8,735,660.375	32	64	256	256	2445.98490512564	1.318359375°	90, -180	5.625°	5.625°	Row 0: 16 Row 1: 8 Rows 2..3: 4 Rows 4..7: 2 Rows 24..27: 2

Figure 19. GNOSIS Global Grid tile matrix set definition

9.1.2. Ecere D106 GeoPackage Producer

Ecere improved its GeoPackage producer with support for vector tiles extensions built in the first phase of the pilot and also enhanced during Testbed-15 (Open Portrayal Framework). This capability is integrated within GNOSIS Cartographer, allowing the export of imagery, gridded coverages and vector data to GeoPackages in a tiled manner.

The GeoPackage standard and extensions define how to store the geospatial data, styles and metadata (including coordinate reference systems and tiling schemes). During the second phase of the pilot, a new extension better aligning GeoPackages with the OGC Two Dimensional Tile Matrix Set standard was developed, while vector tiles and portrayal extensions defined in the first phase

and in Testbed 15 saw clarifications and improvements. GNOSIS Cartographer's GeoPackage producer was improved to reflect these additions and changes.

The new TMS extension enabled tiling schemes making use of variable width tile matrices, such as the GNOSIS Global Grid and CDB Global Grid. It also avoids duplicating the full description of the same tile matrix sets for each layer, as it is common for multiple layers in one GeoPackage to use the same tile matrix set, especially in the context of multi-layer vector tiles. The compatibility with the standard GeoPackage tile matrix set tables is provided using SQL views on the extension tables. All GeoPackages produced by Ecere, each following one of four different tiling schemes (WebMercatorQuad, WorldMercatorWGS84Quad, WorldCRS84Quad, GNOSIS Global Grid) leveraged this TMS extension.

```
sqlite> select * from gpkgext_styles;
id      style      description  uri
-----
1       newStyle1  newStyle1   osmtds/newStyle1
2       night     night      osmtds/night
3       overlay   overlay    osmtds/overlay
4       topographi topographic osmtds/topograph
sqlite> select * from gpkgext_semantic_annotations;
id      type      title      description  uri
-----
1       stylable_layer_set osmtds    osmtds      osmtds
sqlite> select * from gpkgext_sa_reference limit 5;
table_name  key_column_name  key_value  sa_id
-----
gpkgext_styles  id              1          1
gpkgext_styles  id              2          1
gpkgext_styles  id              3          1
gpkgext_styles  id              4          1
gpkgext_vt_lay  id              1          1
sqlite> select * from gpkgext_symbols limit 2;
id      uri      symbol      title      description
-----
1       square_b  square_b   square_b
2       Fuel_Stati Fuel_Stati Fuel_Stati
sqlite> select * from gpkgext_symbol_images limit 2;
id      content_id  symbol_id  width  height  offset_x  offset_y  pixel_ratio
-----
1       1          1         64     64     0         0         1
2       2          2         0      0      0         0         1
sqlite> select id, format, uri from gpkgext_symbol_content limit 2;
id      format      uri
-----
1       image/png  square_b
2       image/svg+ Fuel_Stati
sqlite> █
```

Figure 20. Portrayal extensions tables in GeoPackage produced by Ecere

Important progress was made on clarifying how to store symbology information within GeoPackages, how data layers and styles should be associated using flexible semantic annotations, and how images or SVG symbols should be stored and referenced by style sheets, including the possibility of using either individual or sprite sheets, as used in Mapbox GL styles. The improved specifications also facilitate using the exact same style sheets offline and online. The GeoPackages produced by Ecere included style sheets and symbols demonstrating the full extent of these capabilities, following development to reflect the updated specifications.

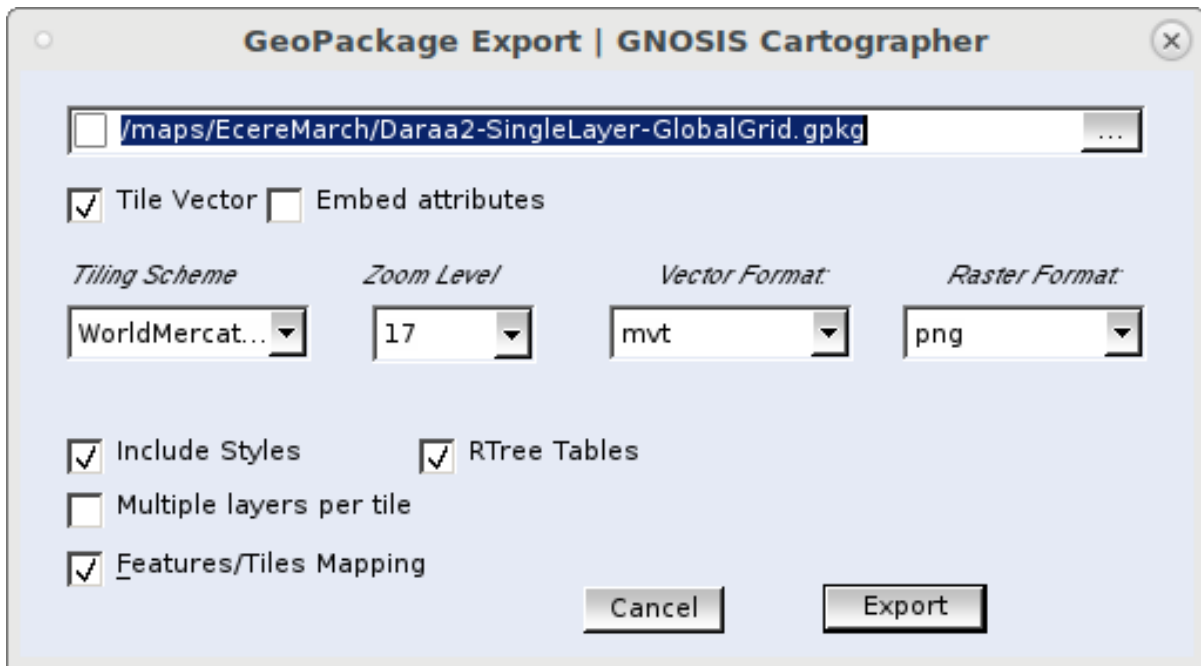


Figure 21. GeoPackage exporting options in GNOSIS Cartographer

When generating a GeoPackage, the user is presented with a dialog including a number of options. One is the choice of an encoding, for both vector and raster formats. The producer supports encoding vector tiles as Mapbox Vector Tiles, GeoJSON or GNOSIS Map Tiles. Only the Mapbox Vector Tiles encoding is available when choosing to encode multiple layers per tile. For the second phase of the pilot, Ecere focused on generating GeoPackages encoded as Mapbox Vector Tiles, in both *one layer per tile* and *multiple layer per tile* flavors. The GNOSIS visualization tools are optimized to work with individual layers per tiles, each in their own SQLite blob, which facilitates directly accessing a specific layer of a tile without having to decode an entire tile comprising multiple layers.

Another important option is whether to store attributes embedded within the tiles themselves, or in an attributes table (see the Attributes Table extension). Embedding the attributes within the tiles is not an option with GNOSIS Map Tiles, which only defines how to store geometry and associated numeric IDs referencing attributes in a database table. When using an attributes table, two additional options are available which provide mechanisms to relate the tiles with the attributes table. One such mechanism is a Features/Tiles mapping table, based on the GeoPackage Related Tables extension. The other is an R-tree spatial index which can store the extent of each feature and serves the dual purpose of identifying a limited set of tiles covered by a certain feature, while also allowing to efficiently focus a query on a specific area of interest. Both mechanisms can be used together, and were included in all GeoPackages including attributes tables generated by Ecere. The R-tree spatial index created by the producer uses 32-bit integer decimal degrees multiplied by a factor of 10^7 to store the extents, maintaining the same precision as OpenStreetMap. Most GeoPackages generated during this second phase used the attributes table approach, but the embedded attributes option was also tested.

```

sqlite> 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;
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 الزبيبي
sqlite> select base_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;
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 22. Performing queries enabled by attributes tables, R-Trees and Features/Tiles mapping tables

The use of an attributes table instead of embedded attributes is highly recommended, as on one hand it greatly reduces the size of the GeoPackage, while on the other it also enables the use of SQL queries, which together with compact vector tiles and a spatial index (also recommended) result in a very efficient general purpose geospatial vector data store, able to handle a large amount of both geometry and attributes information. For example, it should be capable of easily scaling to the entire OpenStreetMap Planet dataset in a single GeoPackage, wherever a single large database file is practical. Practical examples of attributes and spatial queries with the GeoPackages produced by Ecere are discussed in the VTP2 Filtering Language Engineering Report.

An important concept which would also greatly help covering large sparse extents for whole planet GeoPackage experiments would be a mechanism to identify completely *full* tiles, as previously discussed in Testbed 13 - Vector Tiles and the first phase of the pilot. An additional boolean flag column to the tiles table, or the use of a view may be required for this purpose. A value of *full = true* for a given tile would mean that no additional detail can be found at any more detailed zoom levels, but the content of this tile should be used instead. This would be particularly useful for the interior of vast land or ocean polygons, as it would allow to simply omit millions of tile entries. Potentially only the special case of entirely filled polygon tiles could be considered.

The GeoPackages produced were also tested in the Ecere client applications, as well as a backing data store for the GNOSIS Map Server Tiles API (requiring the use of attributes tables and R-trees for full functionality).

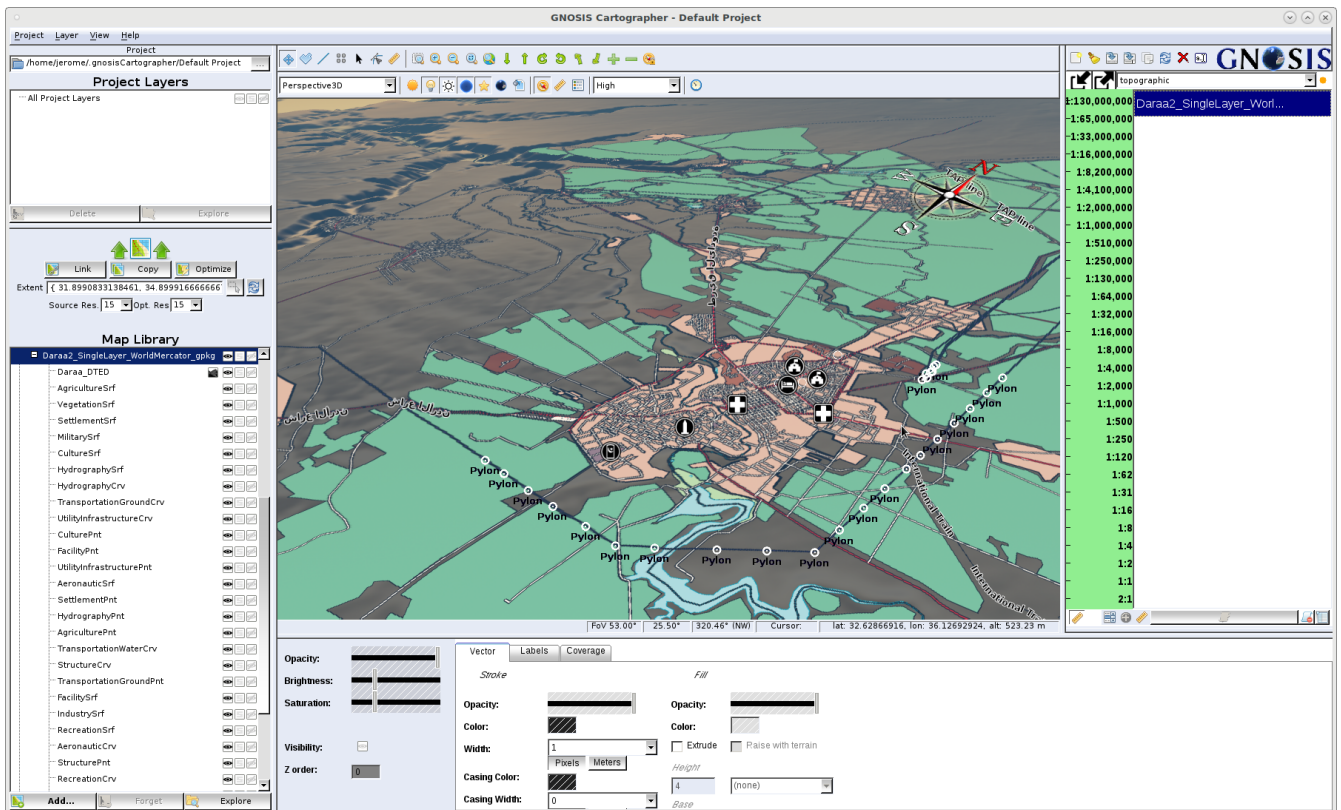


Figure 23. Visualizing the World Mercator / Single Layer GeoPackage produced by Ecere

9.1.3. GeoSolutions D100 Features API

GeoSolutions sought to include Vector Tiles resources below the collections exposed in their Feature Server which supports tiles and styles [8]. GeoSolutions extended the Feature Server implemented in VTP1 which had been built on top of GeoServer.

The GeoSolutions Feature Server supports the OGC API - Features standard. During VTP2 the following new capabilities and improvements were implemented:

- Upgraded the implementation to match the Features API 1.0 release
- Ran the CITE tests and fixed the issues reported (i.e. issues 86 [https://github.com/opengeospatial/ets-ogcapi-features10/issues/86] and 89 [https://github.com/opengeospatial/ets-ogcapi-features10/issues/89]).
- Implemented the draft [OGC API - Features CRS extension](https://github.com/opengeospatial/ogcapi-features/blob/master/extensions/crs) [https://github.com/opengeospatial/ogcapi-features/blob/master/extensions/crs], advertising the supported CRSs and allowing to reproject results to coordinate reference systems other than [CRS84](http://www.opengis.net/def/crs/OGC/1.3/CRS84) [http://www.opengis.net/def/crs/OGC/1.3/CRS84], as well as reporting the returned axis order as a response header.
- 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) [https://github.com/opengeospatial/ogcapi-features/tree/master/extensions/cql] draft, along with queryables support (more details available in the VPT2 Filtering ER).
- Implemented the draft OGC API - Tiles building blocks as part of the OGC Features API, and added filtering support in the same way as the OGC API - Features extensions.
- Added support for TileJSON metadata, allowing Mapbox clients seamless interaction with the server.

The basic Features API now includes the filtering extension as a default, including both collection

filtering via the `cql-text` language, and `queryables`. The intention is to move, step by step, the OGC API - Features module towards feature parity with classic GeoServer WFS functionality. In this spirit, the draft OGC API- Features CRS extensions was also implemented.

The OGC API - Tiles building block has been added as an optional plugin instead. When included, the following changes appear:

- The API shows new resources for `TileMatrixSet` and tiles for each collection
- The landing page points at the `TileMatrixSet` resource
- The collections expose a link to the "tiles" and add support for encoding both Mapbox and GeoJSON vector tiles
- The `tiles` resource provides access to a `TileJSON` description of tile contents.

Multi-layers support has not been added to this implementation, in order to preserve server stability. This is especially for existing installations, having already a set of layers, upgrading to a newer version of GeoServer and testing the OGC API in the process. Implementing multi-tiles support, as currently specified, would have required implementing a `features/tiles` endpoint, returning all available collections in the tiles by default.

By experience providing support, both on the public support lists, and commercially, GeoServer installations can have a large number of layers configured: the common case includes hundreds or thousands of them, with a small number of cases ranging in the tens of thousands, and a single case of more than a hundred thousand layers observed in practice. With those numbers in mind, including all the collections in tiles is not safe, and perhaps not useful. As a result, the implementation has been postponed and [an issue raised](https://github.com/opengeospatial/OGC-API-Tiles/issues/17) [https://github.com/opengeospatial/OGC-API-Tiles/issues/17] on the OGC API - Tiles repository. In the meantime, multi-layer tiles can be retrieved from the [Tiles endpoint](#), where their contents can be controlled via the layer groups mechanism, both in terms of layers, as well as contents based on the requested zoom level.

[Figure 24](#) shows the landing page of the GeoSolutions Feature Server deployed in VTP2.

GeoServer Features 1.0 Service

This is the landing page of the Features 1.0 service, providing links to the service API and its contents. This document is also available as [application/x-yaml](#), [application/json](#), [application/cbor](#).

API definition

The [API document](#) provides a machine processable description of this service API conformant to OpenAPI 3. This API document is also available as [application/vnd.oai.openapi+json;version=3.0](#), [application/x-yaml](#), [application/cbor](#), [text/html](#).

Collections

The [collection page](#) provides a list of all the collections available in this service. This collection page is also available as [application/x-yaml](#), [application/json](#), [application/cbor](#).

Tile matrix sets

Tiles are cached on [tile matrix sets](#), defining tile layouts and zoom levels. This page is also available as [application/x-yaml](#), [application/json](#), [application/cbor](#).

Figure 24. Landing page of the GeoSolutions Feature Server in VTP2

Figure 25 shows the API as presented using Swagger UI.

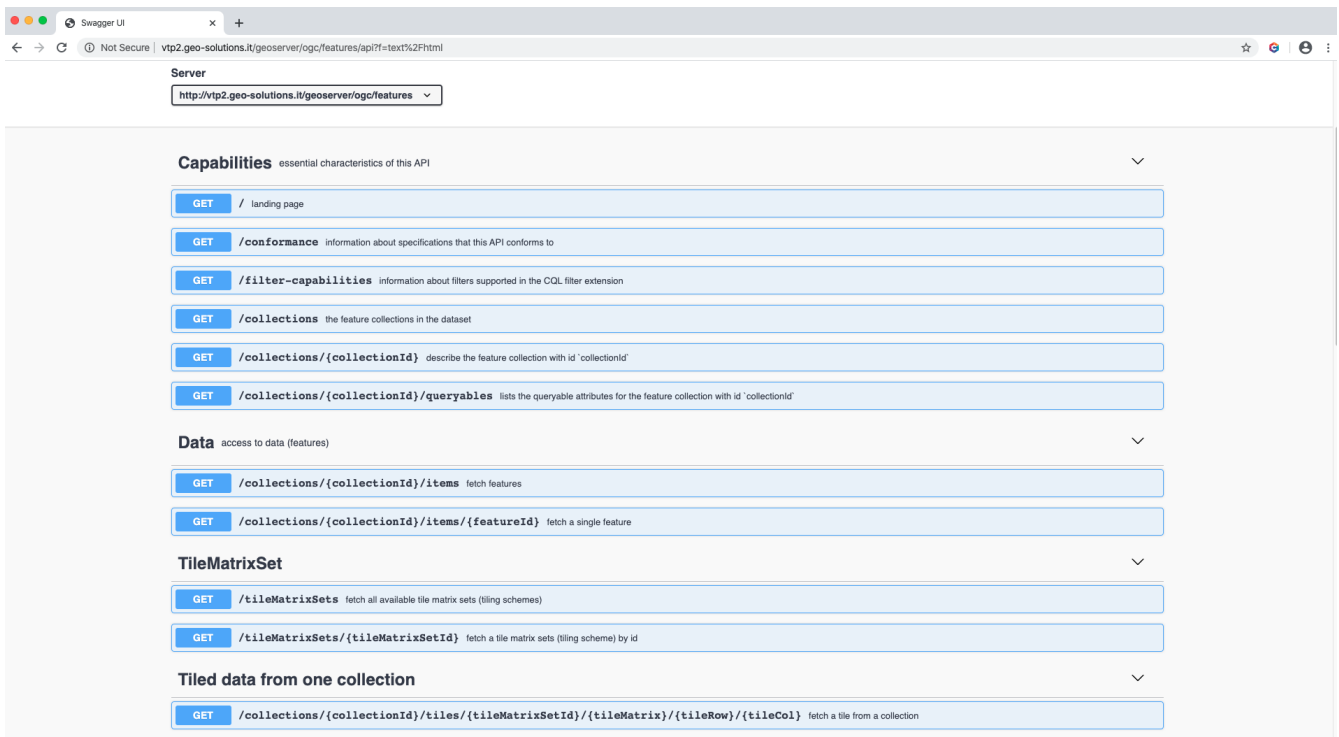


Figure 25. API of the GeoSolutions Feature Server as presented using Swagger UI

Figure 26 shows links to the tiles description for the `vtp:CultureSrf` collection.

vtp:CultureSrf

Formats available and url templates to access tiles:

- application/vnd.mapbox-vector-tile: <https://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp%3ACultureSrf/tiles/{tileMatrixSetId}/{tileMatrix}/{tileR>
- application/json;type=geojson: <https://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp%3ACultureSrf/tiles/{tileMatrixSetId}/{tileMatrix}/{tileR>
- application/json;type=topojson: <https://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp%3ACultureSrf/tiles/{tileMatrixSetId}/{tileMatrix}/{tileR>

Tiles metadata available as:

- application/x-yaml: <https://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp:CultureSrf/tiles/{tileMatrixSetId}/metadata?f=application%2Fyaml>
- application/json: <https://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp:CultureSrf/tiles/{tileMatrixSetId}/metadata?f=application%2Fjson>
- application/cbor: <https://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp:CultureSrf/tiles/{tileMatrixSetId}/metadata?f=application%2Fcbor>
- text/html: <https://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp:CultureSrf/tiles/{tileMatrixSetId}/metadata?f=text%2Fhtml>

Tile matrix sets in which the above URLs are available, and grid limits:

- [WebMercatorQuad](#)

tileMatrixSetId	minRow	maxRow	minCol	maxCol

Figure 26. Tiles description, including link to the TileJSON metadata

In order to help clients dumping vector tiles in a GeoPackage, and to best integrate with the Mapbox ecosystem, the `tiles` resources also point to TileJSON descriptions of the collection. Here is an example, for a single collection:

```
``http://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp:CultureSrf/tiles/WebMercatorQuad/metadata?f=application%2Fjson``
```

```
{
  "name": "vtp:CultureSrf",
  "scheme": "xyz",
  "tiles": [
    "http://vtp2.geo-solutions.it/geoserver/ogc/features/collections/vtp%3ACultureSrf/tiles/WebMercatorQuad/{z}/{y}/{x}?f=application%2Fvnd.mapbox-vector-tile"
  ],
  "center": [
    36.123046875000355,
    32.62083957569541,
    11
  ],
  "bounds": [
    36.078098556,
    32.598968891,
    36.118825116,
    32.6432052850001
  ],
  "vector_layers": [
    {
      "id": "CultureSrf",
      "fields": {
        "LZN": "number",
        "OTH": "string",
        "ADI": "integer",
        "SAX_RS2": "string",

```

```
"SAX_RS3": "string",
"SAX_RS4": "string",
"SAX_RS5": "string",
"F_CODE": "string",
"ZVH": "number",
"FFN": "integer",
"ADR": "string",
"ZI005_FNA": "string",
"ZI001_SRT": "string",
"SAX_RS1": "string",
"CDR": "string",
"ZI005_NFN": "string",
"ZI037_REL": "integer",
"ZSAX_RX3": "string",
"SDQ": "integer",
"ZSAX_RX4": "string",
"CAA": "integer",
"SAX_RX1": "string",
"ZI001_VSN": "string",
"SAX_RX2": "string",
"ZI020_GE4": "string",
"ZSAX_RX0": "string",
"SSR2": "integer",
"SSR3": "integer",
"SAX_RX7": "string",
"HSS": "integer",
"SAX_RX8": "string",
"SAX_RX9": "string",
"CAM": "integer",
"SAX_RX5": "string",
"SAX_RX6": "string",
"ARA": "number",
"VOI": "string",
"PCF": "integer",
"WID": "number",
"SAX_RS6": "string",
"HGT": "number",
"ZI001_SDP": "string",
"SAX_RS8": "string",
"SAX_RS9": "string",
"TOS": "integer",
"ZI001_VSD": "string",
"ZSAX_RS0": "string",
"ZI001_VSC": "string",
"ZI001_SDV": "string",
"SAX_RY0": "string",
"MCC2": "integer",
"SAX_RY1": "string",
"MCC3": "integer",
"SAX_RY2": "string",
"FFN2": "integer",
```



```

    "FFN3": "integer",
    "SRL": "integer",
    "ZI001_SPS": "integer",
    "VLM": "number",
    "TTY": "integer",
    "UFI": "string",
    "AWP": "integer",
    "ZI024_YWQ": "integer",
    "ZI026_CTUC": "integer",
    "CSO": "integer",
    "ZI006_MEM": "string",
    "A00": "number",
    "SSC": "integer",
    "FCSUBTYPE": "integer",
    "LMC": "integer",
    "CCG": "integer",
    "ZI026_CTUL": "integer",
    "BEN": "string",
    "MCC": "integer",
    "HEI": "number",
    "ZI026_CTUU": "integer",
    "CCN": "string",
    "SSR": "integer",
    "ZI037_RFA": "integer",
    "ZI004_RCG": "string"
  },
  "geometry_type": "polygon"
}
]
}

```

The work described above is, at the time of writing, available as two GeoServer community modules, the improved [OGC API - Features](https://github.com/geoserver/geoserver/tree/master/src/community/ogcapi/ogcapi-features) [https://github.com/geoserver/geoserver/tree/master/src/community/ogcapi/ogcapi-features] implementation, along with the tiled features plugin, [Tiled Features](https://github.com/geoserver/geoserver/tree/master/src/community/ogcapi/ogcapi-tiled-features) [https://github.com/geoserver/geoserver/tree/master/src/community/ogcapi/ogcapi-tiled-features] adding the tiles building blocks on the Features API implementation.

Binaries of the modules can be downloaded as part of the [GeoServer nightly builds](https://build.geoserver.org/geoserver/master/) [https://build.geoserver.org/geoserver/master/].

Finally, during VTP2 some experiments have been carried out with [FlatGeoBuf](https://docs.geoserver.org/latest/en/user/community/flatgeobuf/index.html) [https://docs.geoserver.org/latest/en/user/community/flatgeobuf/index.html], "a performant binary encoding for geographic data based on flatbuffers". The format is provided as an output from the `items` resource, and can be filtered just like the GeoJSON and GML ones. The superior encoding performance, ability to stream out the response without advance preparation, and the small network footprint, make it an interesting candidate for future experimentation. The [FlatGeoBuf module](https://docs.geoserver.org/latest/en/user/community/flatgeobuf/index.html) [https://docs.geoserver.org/latest/en/user/community/flatgeobuf/index.html] can also be found among the GeoServer nightly builds.

9.1.4. GeoSolutions D102 Tiles API

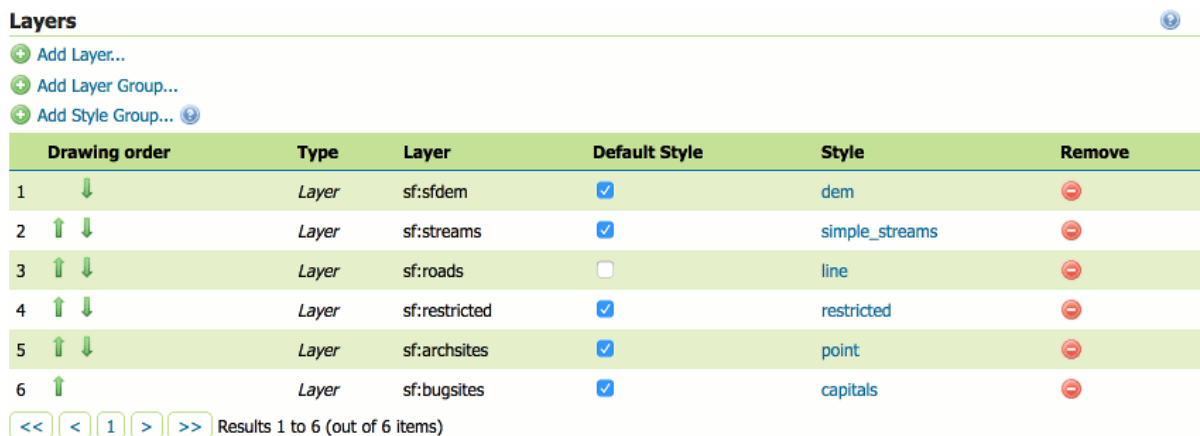
The GeoSolutions Tiles Server implemented for VTP1 (see VTP Extension ER) already supports Vector Tiles resources and supports styling.

The existing Tiles API publishes as collections different types of layers, including:

- Vector layers.
- Raster layers.
- **Layer groups** [<https://docs.geoserver.org/stable/en/user/data/webadmin/layergroups.html>], that is, server-side configured layer trees, normally stacked to build a base-map in WMS and WMTS.
- Cascaded WMS and WMTS layers.

The stand-alone vector layers, as well as pure vector layer groups, are the only type of collection exposing "data tiles", while the others serve "map" tiles only, that is, tiles rendered as PNGs.

As part of the Testbed 15 Style [9] work, most layers expose the list of styles associated to them in the GeoServer configuration. As shown in Figure 27, the association is made by linking to the relevant resources of the Styles API, for description and download.



The screenshot shows the 'Layers' page in GeoServer. At the top, there are three buttons: 'Add Layer...', 'Add Layer Group...', and 'Add Style Group...'. Below these is a table with the following columns: 'Drawing order', 'Type', 'Layer', 'Default Style', 'Style', and 'Remove'. The table contains six rows of layer data. At the bottom of the table, there are navigation buttons: '<<', '<', '1', '>', '>>' and the text 'Results 1 to 6 (out of 6 items)'.

Drawing order	Type	Layer	Default Style	Style	Remove
1	Layer	sf:sfdem	<input checked="" type="checkbox"/>	dem	
2	Layer	sf:streams	<input checked="" type="checkbox"/>	simple_streams	
3	Layer	sf:roads	<input type="checkbox"/>	line	
4	Layer	sf:restricted	<input checked="" type="checkbox"/>	restricted	
5	Layer	sf:archsites	<input checked="" type="checkbox"/>	point	
6	Layer	sf:bugsites	<input checked="" type="checkbox"/>	capitals	

Figure 27. GeoServer layer group definition as a stack of layers and associated style

Layer groups normally refer to a list of layers and styles as part of their definition, as such they do not have a "style" of their own and the collection resource does not point to any. However, a layer group in GeoServer can also be defined as a "style group", that is, a group whose definition is provided as a multi-layer SLD or MapBox GL styles file. In this case the style defining the group is made available as part of the collection resource.

Vector tile encodings supported by the server included GeoJSON and MVT formats, as illustrated in Figure 28. The tiles server was configured to offer filtering capability, with support for SQL and an ability to generate Vector Tiles on-the-fly with filters. The server was configured to serve tile caches if no filters are requested.

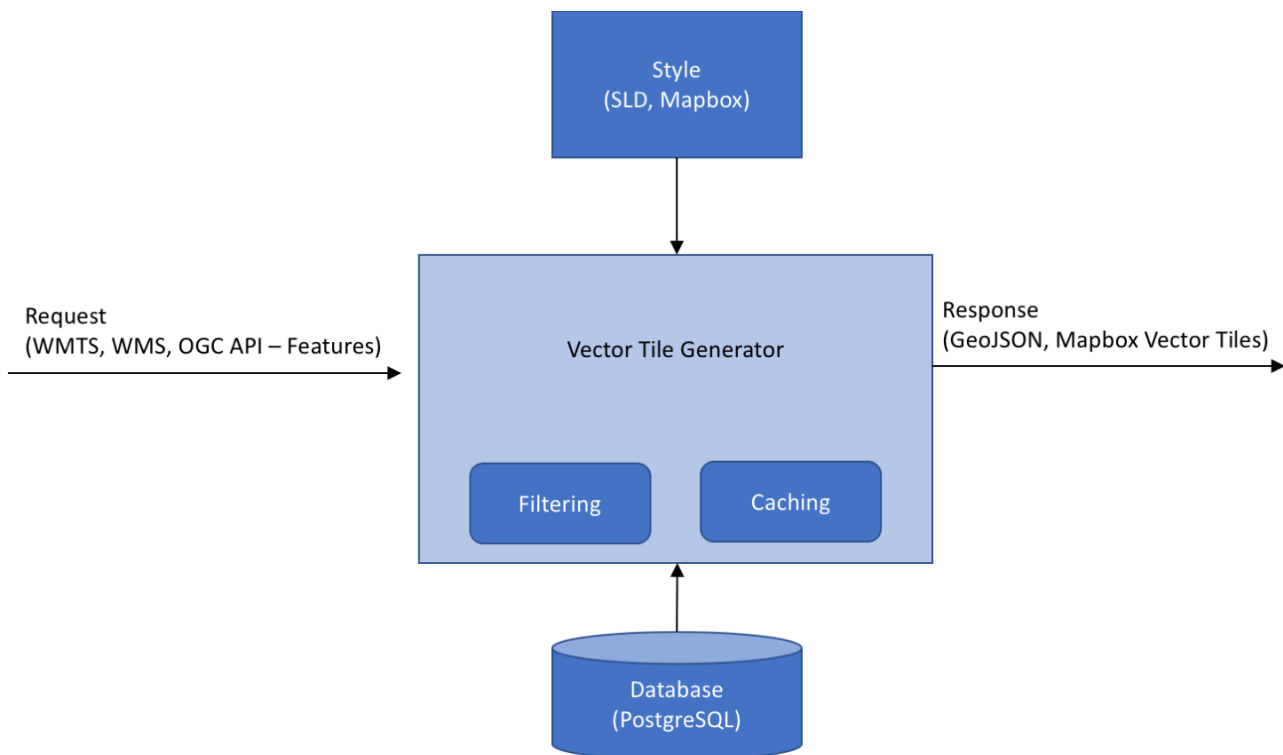


Figure 28. GeoSolutions Tile Server generation of Vector Tiles in VTP2

The GeoSolutions Tiles Server was configured to support a variety of tile matrix sets, including, among others:

- [WorldCRS84Quad](https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/WorldCRS84Quad?f=text%2Fhtml) [https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/WorldCRS84Quad?f=text%2Fhtml]
- [EuropeanETRS89_LAEAQuad](https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/EuropeanETRS89_LAEAQuad?f=text%2Fhtml) [https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/EuropeanETRS89_LAEAQuad?f=text%2Fhtml]
- [UPSArcticWGS84Quad](https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/UPSArcticWGS84Quad?f=text%2Fhtml) [https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/UPSArcticWGS84Quad?f=text%2Fhtml]
- [GoogleCRS84Quad](https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/GoogleCRS84Quad?f=text%2Fhtml) [https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/GoogleCRS84Quad?f=text%2Fhtml]
- [WebMercatorQuad](https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/WebMercatorQuad?f=text%2Fhtml) [https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/WebMercatorQuad?f=text%2Fhtml]
- [EPG:3395](https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/EPG%3A3395?f=text%2Fhtml) [https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/EPG%3A3395?f=text%2Fhtml]
- [UPSAntarcticWGS84](https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/UPSAntarcticWGS84?f=text%2Fhtml) [https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/UPSAntarcticWGS84?f=text%2Fhtml]
- [UTM##WGS84Quad](https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/UTM%23%23WGS84Quad?f=text%2Fhtml) [https://vtp2.geo-solutions.it/geoserver/ogc/tiles/tileMatrixSets/UTM%23%23WGS84Quad?f=text%2Fhtml]

The API resource of the GeoSolutions Tile Server is shown on [Figure 29](#).

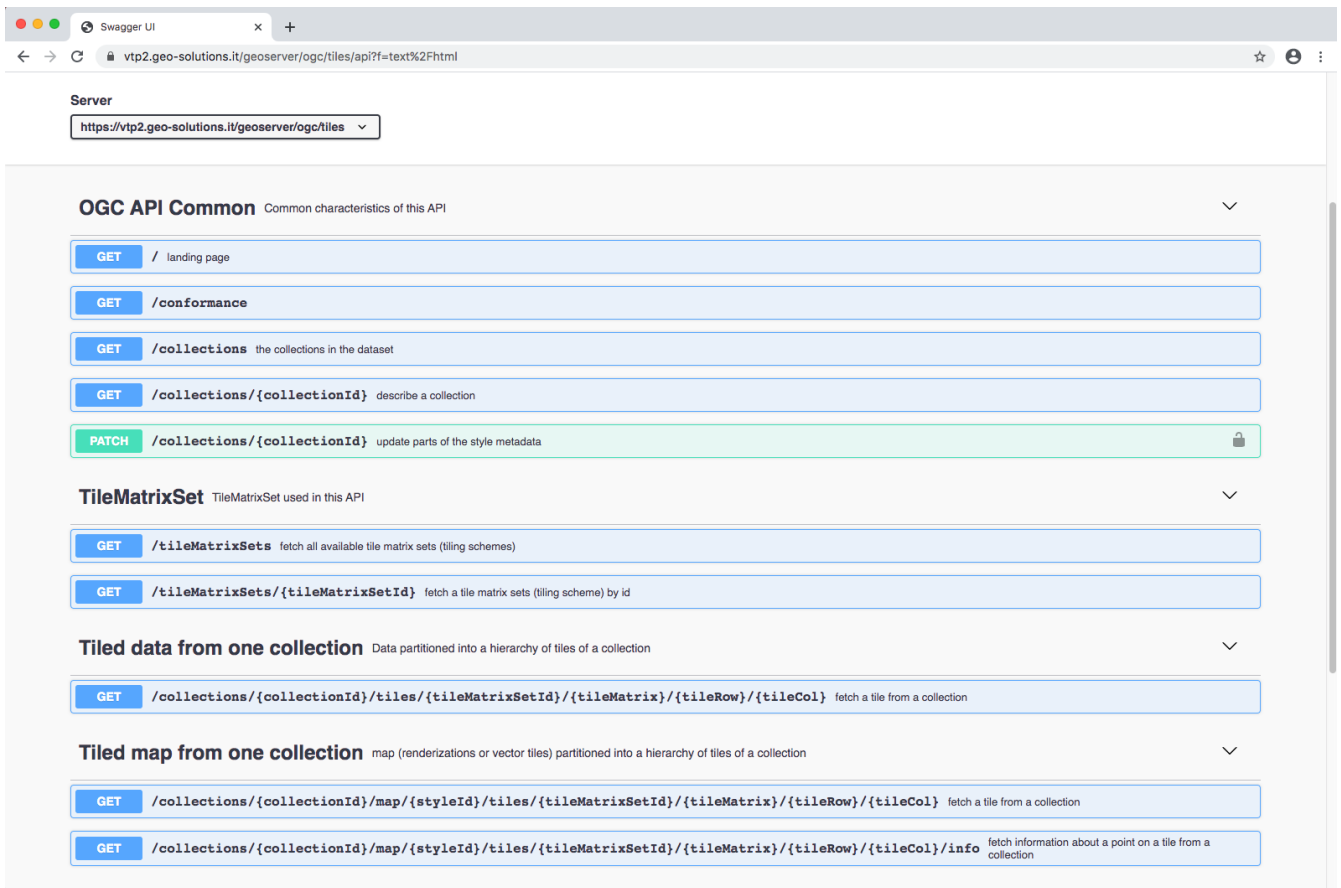


Figure 29. The landing page of the GeoSolutions Tile Server in VTP2

In order to help clients dumping vector tiles in a GeoPackage, and to best integrate with the Mapbox ecosystem, the `tiles` resources also point to TileJSON descriptions of the collection. Here is an excerpt from a multi-layer collection TileJSON:

```
``https://vtp2.geo-solutions.it/geoserver/ogc/tiles/collections/vtp:daraa_vtp/tiles/%7BtileMatrixSetId%7D/metadata?f=application%2Fjson``
```

```
{
  "name": "vtp:daraa_vtp",
  "scheme": "xyz",
  "tiles": [
    "http://vtp2.geo-solutions.it/geoserver/ogc/tiles/collections/vtp%3Adaraa_vtp/tiles/WebMercatorQuad/{z}/{y}/{x}?f=application%2Fvnd.mapbox-vector-tile"
  ],
  "center": [
    36.56250000000041,
    34.27502568554792,
    6
  ],
  "bounds": [
    35.8995094299316,
    32.4131851196289,
```

```

36.5781326293945,
33.1460647583008
],
"vector_layers": [
  {
    "id": "AgricultureSrf",
    "fields": {
      "OTH": "string",
      "PVH": "number",
      "TSCL": "number",
      "ZI005_FNA": "string",
      "CDR": "string",
      "...": "..."
    },
    "geometry_type": "polygon"
  },
  {
    "id": "VegetationSrf",
    "fields": {
      "LZN": "number",
      "OTH": "string",
      "PVH": "number",
      "TRE": "integer",
      "...": "..."
    },
    "geometry_type": "polygon"
  },
  {
    "id": "MilitarySrf",
    "fields": {
      "OTH": "string",
      "WD3": "number",
      "FRT": "integer",
      "FRT3": "integer",
      "FRT2": "integer",
      "ZI005_FNA": "string",
      "...": "..."
    },
    "geometry_type": "polygon"
  },
  {
    "..."
  }
]
}

```

The Tiles API is, at the time of writing, available as a [community module](https://github.com/geoserver/geoserver/tree/master/src/community/ogcapi/ogcapi-tiles) [https://github.com/geoserver/geoserver/tree/master/src/community/ogcapi/ogcapi-tiles], that anyone can inspect and download as part of the [GeoServer nightly builds](https://build.geoserver.org/geoserver/master/) [https://build.geoserver.org/geoserver/master/].

In the future the module is expected to be included as part of releases, as an extension. Eventually the module will take its place in the core GeoServer download, alongside the existing WMTS

functionality.

9.1.5. interactive instruments D101 Features, Tiles and Styles API

9.1.5.1. Overview

At the beginning of the OGC Vector Tiles Pilot 2, the open-source tool [ldproxy](https://github.com/interactive-instruments/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 schema of NGA was used. The data was loaded into a PostgreSQL database that served as the data backend for the ldproxy deployment.

For VTP2, interactive instruments extended ldproxy with:

- Consistent server-side filtering for features in the Features and Tiles resources of the API, based on the [proposed Common Query Language \(CQL\) extension](https://github.com/opengeospatial/ogcapi-features/tree/master/extensions/cql) [https://github.com/opengeospatial/ogcapi-features/tree/master/extensions/cql];
- Additional well-known tiling schemes - WorldCRS84Quad based on the geographic coordinates, and WorldMercatorWGS84Quad, based on the standard Mercator projection;
- Additional metadata for tile sets and features.

9.1.5.2. The starting point

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, as shown in [Figure 30](#) and [Figure 31](#).

Daraa

This is a test dataset for the Open Portrayal Framework thread in the OGC Testbed-15 as well as for the OGC Vector Tiles Pilot Phase 2. The data is OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.

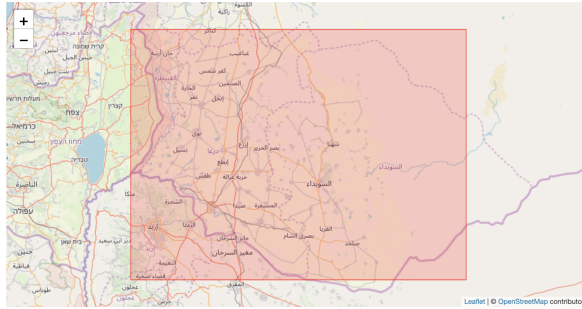
[Access the data](#)

[Access the data as vector tiles](#)

[Styles to render data](#)

API description [Formal definition of the API in OpenAPI 3.0](#)
[Documentation of the API](#)

Spatial Extent



Temporal Extent -

Expert information

Additional Links [OGC API conformance classes implemented by this server](#)
[List of tile matrix sets implemented by this API](#)
[Resources for rendering the data in maps](#)

Figure 30. interactive instruments - the landing page of the API

Daraa

This is a test dataset for the Open Portrayal Framework thread in the OGC Testbed-15 as well as for the OGC Vector Tiles Pilot Phase 2. The data is OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.

Data

- [Aeronautic \(Curves\) — more information](#)
- [Aeronautic \(Surfaces\) — more information](#)
- [Agricultural \(Points\) — more information](#)
- [Agricultural \(Surfaces\) — more information](#)
- [Cultural \(Points\) — more information](#)
- [Cultural \(Surfaces\) — more information](#)
- [Facility \(Surfaces\) — more information](#)
- [Facility \(Points\) — more information](#)
- [Hydrography \(Curves\) — more information](#)
- [Hydrography \(Points\) — more information](#)
- [Hydrography \(Surfaces\) — more information](#)
- [Industry \(Surfaces\) — more information](#)
- [Information \(Points\) — more information](#)
- [Military \(Surfaces\) — more information](#)
- [Other \(Curves\) — more information](#)
- [Other \(Points\) — more information](#)
- [Other \(Surfaces\) — more information](#)
- [Physiography \(Curves\) — more information](#)
- [Recreation \(Curves\) — more information](#)
- [Recreation \(Points\) — more information](#)
- [Recreation \(Surfaces\) — more information](#)
- [Settlement \(Points\) — more information](#)
- [Settlement \(Surfaces\) — more information](#)
- [Structure \(Curves\) — more information](#)
- [Structure \(Points\) — more information](#)
- [Structure \(Surfaces\) — more information](#)
- [Transportation - Ground \(Curves\) — more information](#)
- [Transportation - Ground \(Points\) — more information](#)
- [Transportation - Ground \(Surfaces\) — more information](#)
- [Transportation - Water \(Curves\) — more information](#)
- [Utility Infrastructure \(Curves\) — more information](#)
- [Utility Infrastructure \(Points\) — more information](#)
- [Vegetation \(Surfaces\) — more information](#)

Expert information

Supported CRS <http://www.opengis.net/def/crs/OGC/1.3/CRS84>
<http://www.opengis.net/def/crs/EPSSG/0/3395>
<http://www.opengis.net/def/crs/EPSSG/0/3857>
<http://www.opengis.net/def/crs/EPSSG/0/4326>

Figure 31. interactive instruments - the collections (feature types) in the dataset

The API documentation page of the interactive instruments ldproxy instance is shown on [Figure 32](#).

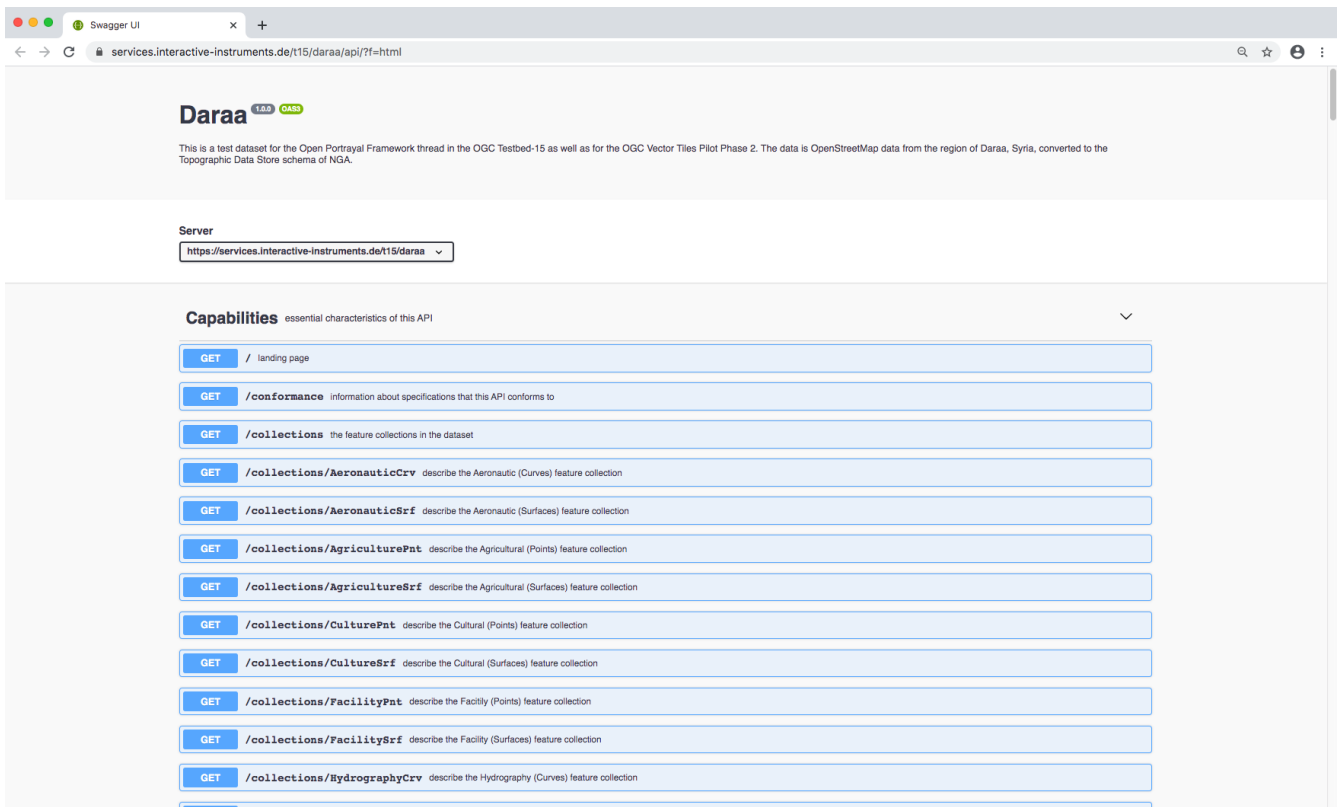


Figure 32. The API documentation page of the interactive instruments *ldprox* instance in VTP2

For filtering, knowledge about the available properties for use in filter predicates is important. These were published as a separate resource for each collection, as shown in Figure 33 for the Transportation (Ground) features with a line string geometry.

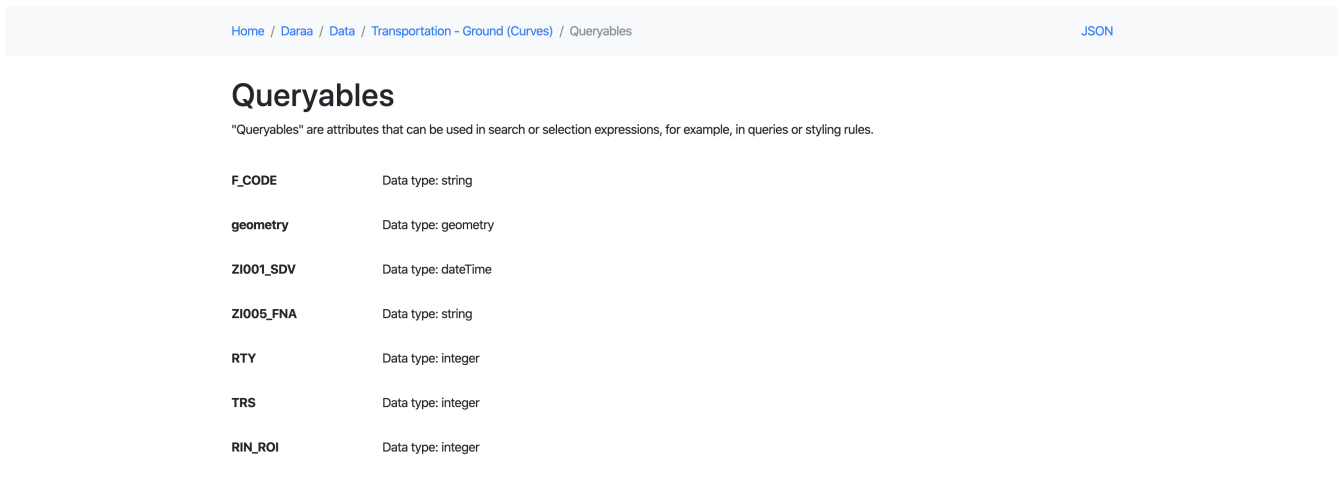


Figure 33. interactive instruments - queryable properties for the Transportation (Ground) features with line string geometry

To illustrate the data, here is a screenshot of a road feature Figure 34:

No Information

id	34
Feature Type	Road
Last Change	09/13/2014, 18:57:11
Name	No Information
Roadway Type	Limited Access Motorway
Route Designation Type	National
Unique Entity Identifier	86c0f5fe-4abd-4e0e-970a-feaf7e0ab314
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

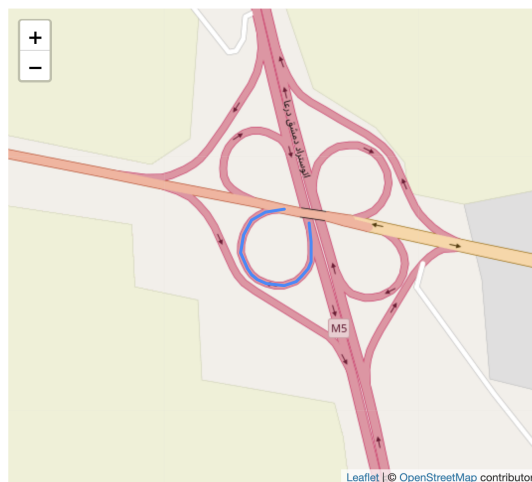


Figure 34. interactive instruments - a road feature

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

```
{
  "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, consistent with the OGC API Tiles draft specification at the time of the pilot. That is, the Mapbox Vector Tiles tile set for every feature collection includes a single layer for the features of the collection and the multi-collection Mapbox Vector Tiles tile set includes multiple layers, one layer for each collection with features for that tile.

The screenshot below, in [Figure 35](#), shows the vector tiles of the Transportation (Ground) features in an OpenLayers map.

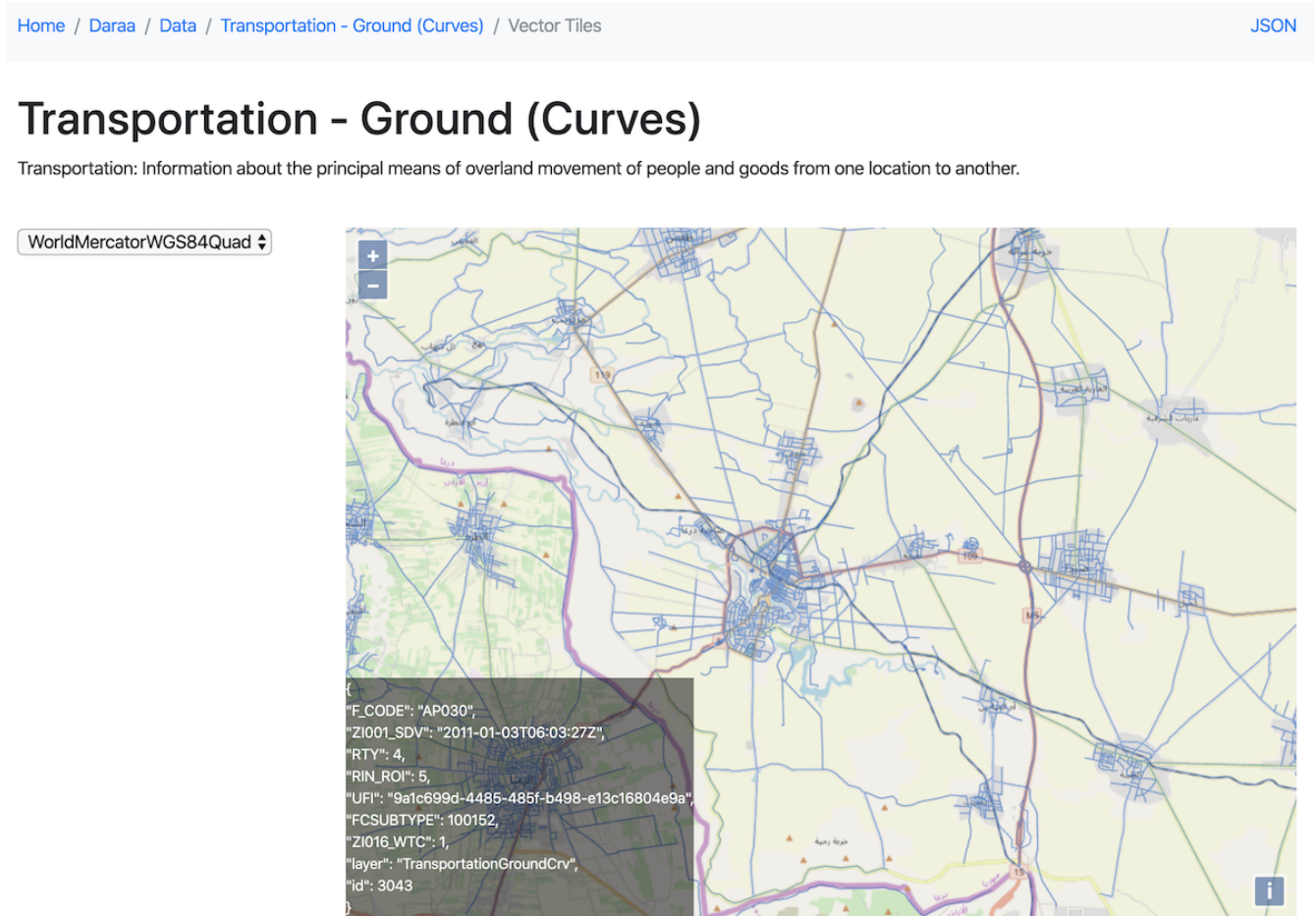


Figure 35. interactive instruments - Mapbox Vector Tiles for the Transportation (Ground) features in OpenLayers

Two sample styles for the data were provided for rendering the feature data using Mapbox Styles as the style encoding: a "topographic" style and a "night" style. The screenshots in [Figure 36](#) and [Figure 37](#) illustrate the two styles using an OpenLayers map and the multi-layer vector tiles.

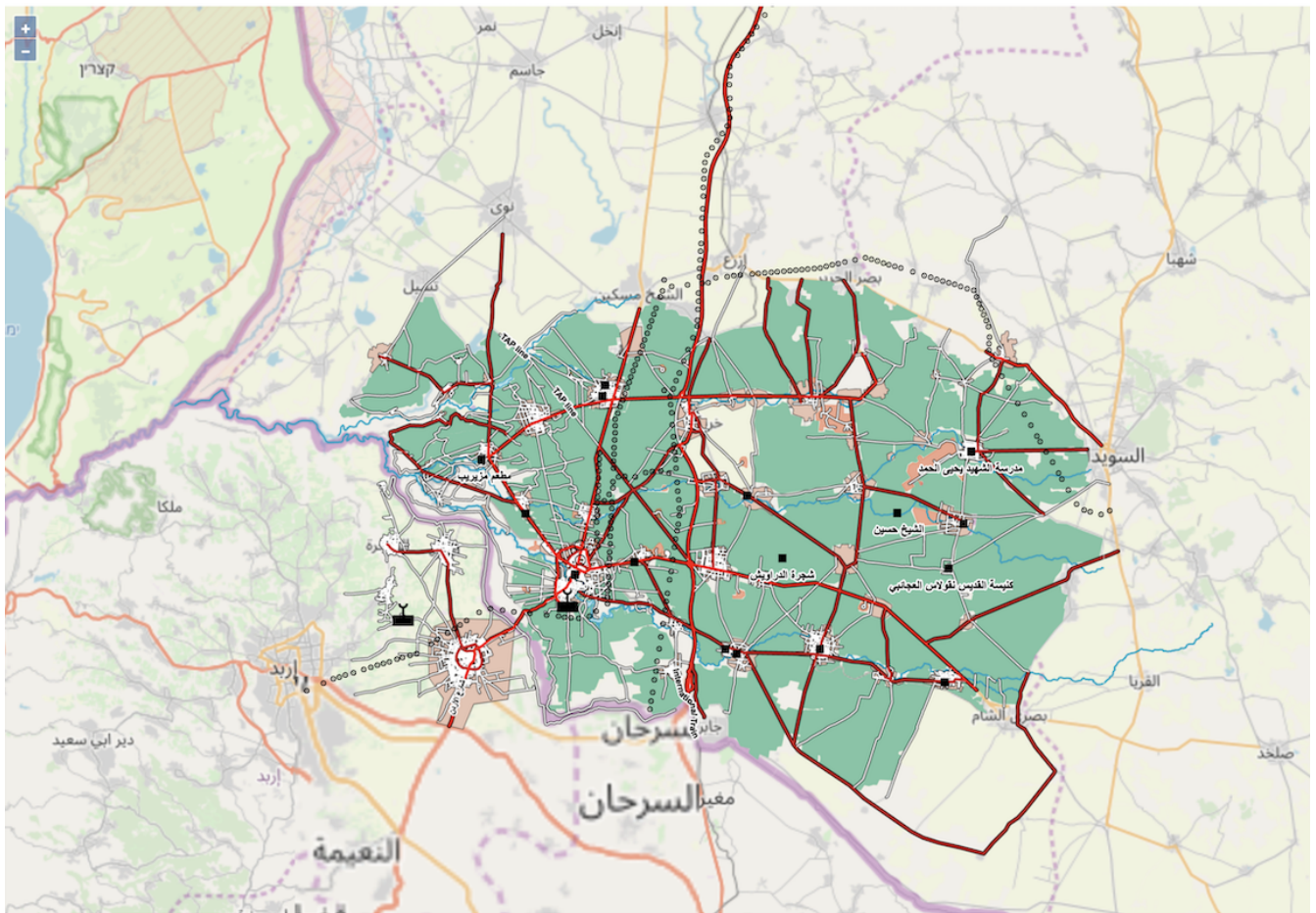


Figure 36. interactive instruments - Multi-layer vector tiles rendered using the topographic style

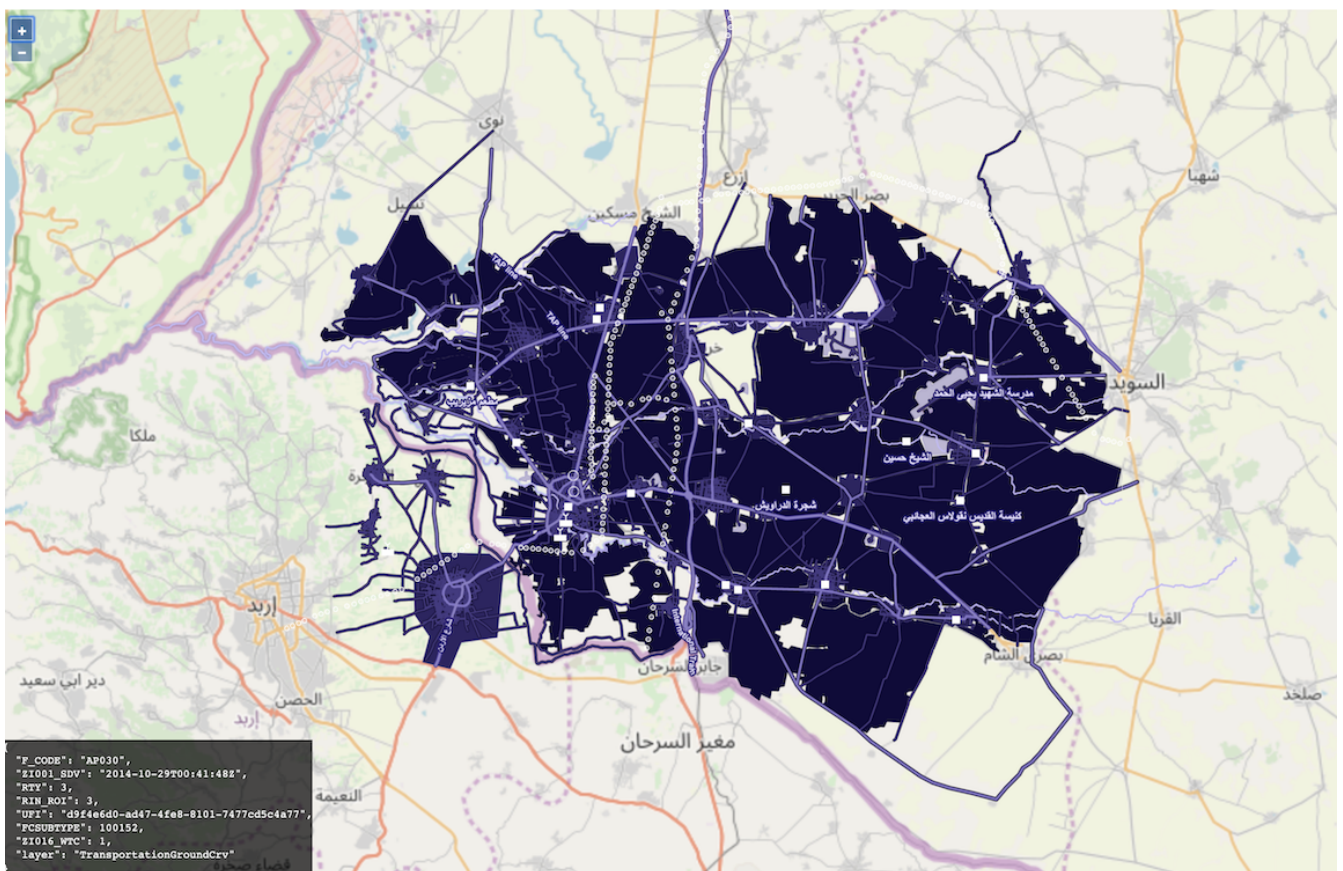


Figure 37. interactive instruments - Multi-layer vector tiles rendered using the night style

9.1.6. Support for filtering

The filtering support implemented in the pilot is documented in detail in the [OGC Vector Tiles Pilot 2: Vector Tiles Filtering Language Engineering Report](http://www.opengis.net/doc/PER/vtp2-D002#ii_implementation) [http://www.opengis.net/doc/PER/vtp2-D002#ii_implementation].

9.1.7. Support for additional tiling schemes

Support for additional tiling schemes beside the standard "WebMercatorQuad" scheme was implemented in ldproxy. The following tiling schemes defined in the OGC Two Dimensional Tile Matrix Set standard were deployed for all tile sets:

- Google Maps Compatible for the World (WebMercatorQuad), based on the Web Mercator projection
- CRS84 for the World (WorldCRS84Quad), based on the geographic coordinates
- WGS84 for the World (WorldMercatorWGS84Quad), based on the standard Mercator projection

Additional screenshots are presented in Appendix B, in the subsection titled [interactive instruments](#).

9.1.7.1. Schema information for features

For each feature collection, a JSON schema describing the schema of the features in the collection was implemented during the pilot as published as a [schema](#) sub-resource of the collection.

interactive instruments - information about the TransportationGroundCrv collection

```
{
  "title" : "Transportation - Ground (Curves)",
  "description" : "Transportation: Information about the principal means of overland
movement of people and goods from one location to another.",
  "links" : [ {
    "rel" : "self",
    "type" : "application/json",
    "title" : "This document",
    "href" : "https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv?f=json"
  }, {
    "rel" : "alternate",
    "type" : "text/html",
    "title" : "This document as HTML",
    "href" : "https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv?f=html"
  }, {
    "rel" : "items",
    "type" : "application/geo+json",
    "title" : "Access the features in the collection 'Transportation - Ground
(Curves)'",
    "href" : "https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv/items?f=json"
  }
]
```

```

}, {
  "rel" : "items",
  "type" : "text/html",
  "title" : "Access the features in the collection 'Transportation - Ground
(Curves)'",
  "href" : "https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv/items?f=html"
}, {
  "rel" : "queryables",
  "title" : "Queryable attributes",
  "href" : "https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv/queryables"
}, {
  "rel" : "describedby",
  "type" : "application/schema+json",
  "title" : "Schema of the features",
  "href" : "https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv/schema"
}, {
  "rel" : "tiles",
  "title" : "Access the data as vector tiles",
  "href" : "https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv/tiles"
} ],
"id" : "TransportationGroundCrv",
"extent" : {
  "spatial" : {
    "bbox" : [ [ 35.9028738, 32.4168138, 36.5747694, 33.1424348 ] ],
    "crs" : "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
  },
  "temporal" : {
    "interval" : [ [ null, null ] ],
    "trs" : "http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"
  }
},
"crs" : [ "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
"http://www.opengis.net/def/crs/EPSG/0/3395",
"http://www.opengis.net/def/crs/EPSG/0/3857",
"http://www.opengis.net/def/crs/EPSG/0/4326" ],
"storageCrs" : "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
"styles" : [ {
  "title" : "Topographic night style",
  "links" : [ {
    "rel" : "stylesheet",
    "type" : "application/vnd.mapbox.style+json",
    "title" : "Stylesheet in style encoding 'Mapbox Style'",
    "href" : "https://services.interactive-
instruments.de/t15/daraa/styles/night?f=mbs"
  } ],
  "rel" : "describedBy",
  "title" : "Style metadata",

```

```

    "href" : "https://services.interactive-
instruments.de/t15/daraa/styles/night/metadata"
  }, {
    "rel" : "map",
    "title" : "Map showing the Style",
    "href" : "https://services.interactive-
instruments.de/t15/daraa/styles/night/map"
  } ],
  "id" : "night"
}, {
  "title" : "Topographic style",
  "links" : [ {
    "rel" : "stylesheet",
    "type" : "application/vnd.mapbox.style+json",
    "title" : "Stylesheet in style encoding 'Mapbox Style'",
    "href" : "https://services.interactive-
instruments.de/t15/daraa/styles/topographic?f=mbs"
  }, {
    "rel" : "describedBy",
    "title" : "Style metadata",
    "href" : "https://services.interactive-
instruments.de/t15/daraa/styles/topographic/metadata"
  }, {
    "rel" : "map",
    "title" : "Map showing the Style",
    "href" : "https://services.interactive-
instruments.de/t15/daraa/styles/topographic/map"
  } ],
  "id" : "topographic"
} ],
"defaultStyle" : "topographic"
}

```

The links include a link with `rel=describedby` and `type=application/schema+json`. The JSON schema is shown in the code block below:

interactive instruments - JSON schema of the TransportationGroundCrv features

```

{
  "$schema" : "http://json-schema.org/draft-07/schema#",
  "$id" : "https://services.interactive-
instruments.de/t15/daraa/collections/TransportationGroundCrv/schema?f=json",
  "type" : "object",
  "title" : "Transportation - Ground (Curves)",
  "description" : "Transportation: Information about the principal means of overland
movement of people and goods from one location to another.",
  "required" : [ "type", "geometry", "properties" ],
  "properties" : {
    "type" : {
      "type" : "string",
      "enum" : [ "Feature" ]
    }
  }
}

```

```

},
"id" : {
  "oneOf" : [ {
    "type" : "string"
  }, {
    "type" : "integer"
  } ]
},
"links" : {
  "type" : "array",
  "items" : {
    "$ref" : "https://api.swaggerhub.com/domains/cportele/ogcapi-features-1/1.0.0#/components/schemas/link"
  }
},
"geometry" : {
  "oneOf" : [ {
    "$ref" : "https://geojson.org/schema/Geometry.json"
  }, {
    "type" : "null"
  } ]
},
"properties" : {
  "oneOf" : [ {
    "type" : "object",
    "properties" : {
      "id" : {
        "type" : "string"
      },
      "F_CODE" : {
        "type" : "string"
      },
      "ZI001_SDV" : {
        "type" : "string",
        "format" : "date-time"
      },
      "UFI" : {
        "type" : "string"
      },
      "ZI005_FNA" : {
        "type" : "string"
      },
      "RTY" : {
        "type" : "integer"
      },
      "FCSUBTYPE" : {
        "type" : "integer"
      },
      "TRS" : {
        "type" : "integer"
      }
    }
  }
}

```



```

    "RIN_ROI" : {
      "type" : "integer"
    },
    "ZI006_MEM" : {
      "type" : "string"
    },
    "ZI001_SDP" : {
      "type" : "string"
    },
    "ZI016_WTC" : {
      "type" : "integer"
    },
    "RIN_RTN" : {
      "type" : "string"
    },
    "RLE" : {
      "type" : "integer"
    },
    "LOC" : {
      "type" : "integer"
    }
  }, {
    "type" : "null"
  } ]
}
}
}

```

9.1.8. Image Matters D107 GeoPackage

For VTP2, Image Matters built on the capabilities implemented in Testbed-15 by extending their GeoPackage Provisioner (producer). As shown in [Figure 38](#), the GeoPackage Provisioner is designed to populate a GeoPackage with the content obtained from existing web services. The Provisioner inspects a web service, parses the relevant documents, locates the relevant content based on the API specification (such as OGC API - Tiles), harvests that information, and writes it to the GeoPackage. The Provisioner uses open-source libraries, sponsored by NGA, to handle the GeoPackage operations.

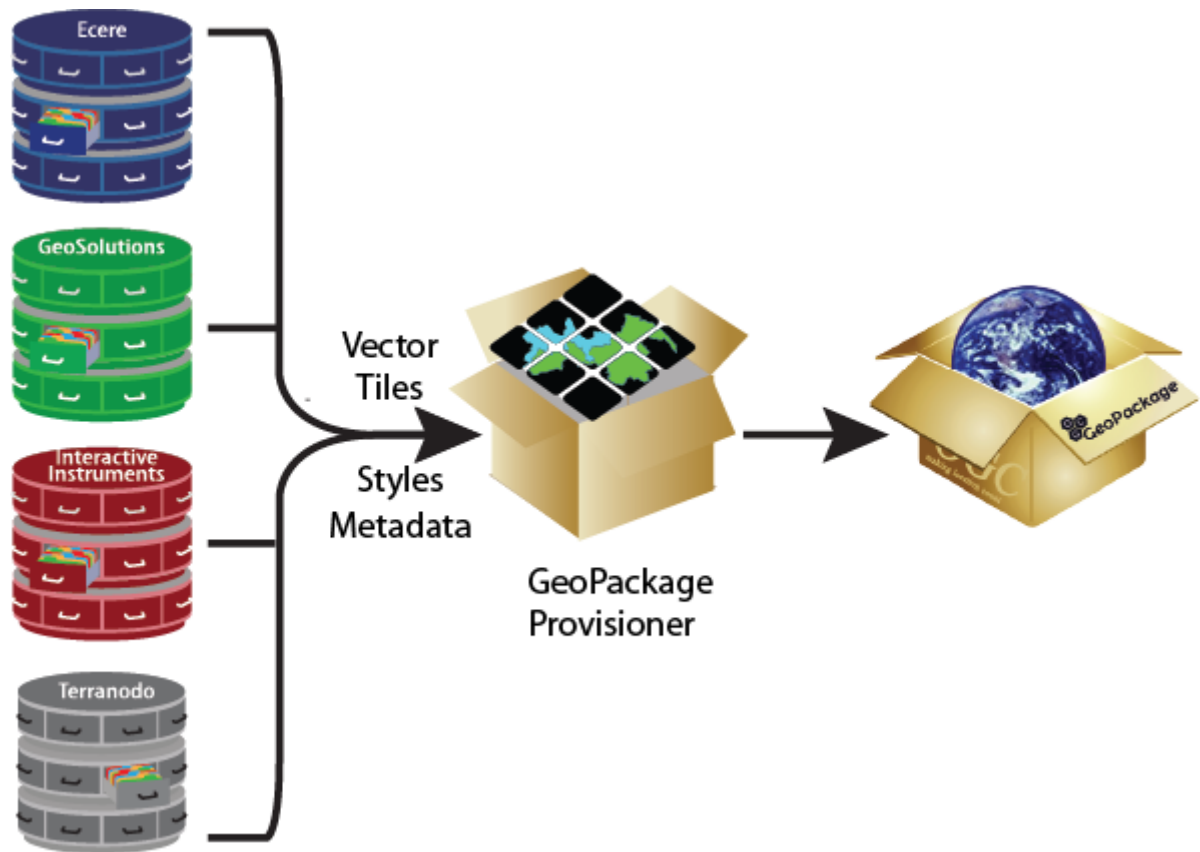


Figure 38. Image Matters GeoPackage Provisioner

For VTP2, Image Matters updated the Provisioner to support the recent changes to the Tiles API. The GeoPackage libraries were extended to support new GeoPackage capabilities. The Provisioner is now able to access vector tiles, styles, and style metadata through this API by applying the following workflow:

1. A provider hosts descriptions of well-known tile matrix sets based on the URL template <https://services.interactive-instruments.de/t15/daraa/tileMatrixSets/{TileMatrixSet}?f=json>. The layer information includes a description of the available tile matrix sets. This information goes into `gpkg_tile_matrix_set` and `gpkg_tile_matrix` and is used in the creation of a user-defined tiles table. Note that other providers are possible.
2. [Apache SIS](http://sis.apache.org/) [http://sis.apache.org/] provides resolution and descriptions of Coordinate Reference Systems (CRSs).
3. A Capabilities document provides links to a multi-tiles layer (conveniently called "tiles") and the styles (if available). If available, tile and description information goes into `gpkg_contents` along with the name of the user-defined tiles table and the requested extents of the layer.
4. The layer information includes descriptions of the tile matrix sets it supports. Those tile matrix sets may include tile matrix set limits, the boundaries around which tiles are available. Making use of this information prevents the client from making unnecessary requests for non-existent tiles.
5. The layer information includes an "item" link that provides a template for accessing tiles based on tile matrix set, tile matrix, row, and column. The tiles are available in one or more formats (e.g., Mapbox and GeoJSON).
6. Based on the parameters provided to the client (extents, tile matrix set, zoom levels, and tile format), the Provisioner calculates and retrieves appropriate vector tiles and inserts them into

the GeoPackage.

7. The Capabilities document may contain a link to styles information. If styles are available, they are harvested into the `gpkgext_styles` and `gpkgext_stylesheets` tables. If metadata is available for those styles, that information is harvested into `gpkg_metadata` and `gpkg_metadata_records`.

The following endpoints were used:

- Ecere: <http://maps.ecere.com/geoapi/collections/vtp/Daraa2>
- GeoSolutions: https://vtp2.geo-solutions.it/geoserver/ogc/tiles/collections/vtp:daraa_vtp
- Interactive Instruments: <https://services.interactive-instruments.de/t15/daraa>
- Terranodo: <http://ogc-vtp.gospatial.org/ogc-api-tiles>

This design proved to be a good match for the emerging OGC API - Tiles. It was straight-forward to identify the relevant information, harvest the content, and write the content to the GeoPackage. Details on how to populate the GeoPackage with vector tiles information, including styles and symbols, are provided in [Section 8](#).

9.1.9. Terranodo D100 Features API

Terranodo used Tegola, a vector tile server written in the [Go programming language](https://golang.org) [https://golang.org]. Tegola supports the EPSG:4326 coordinate reference system. Terranodo created a service with [OpenStreetMap \(OSM\)](http://openstreetmap.org) [http://openstreetmap.org] data from [geofabrik](https://www.geofabrik.de/data/download.html) [https://www.geofabrik.de/data/download.html] Protobuf exports, and served the exports as a set of static resources from the Amazon Web Services (AWS) Simple Storage Service (S3). AWS S3 is one of the services offered by the Amazon Cloud. The OSM data was converted into Postgres tables, from where Tegola was configured to serve data from. The basic metadata for the provisioned data indicated the bounds, min/max zoom, scheme, and grids. For each layer, the metadata described the geometry type and valid zoom levels, among others.

The server was implemented as a tile server/service and not having any concept of styles. Terranodo added an auto-generated style for the built-in viewer (see [Figure 39](#)), but there is no capability to provide actual styles for a particular map or set of layers. This is left entirely up to the client. Styles are handled separately from the server implementation. They can be defined inline inside a client application or retrieved over HTTP. Inside these styles, filters are defined that can specify a subset of features in a layer to be used for cartographic purposes. These styles also define a collection of sprites (referred to as a spriteset) to use for the style as well as fonts. A sprite is an image representing a symbol.

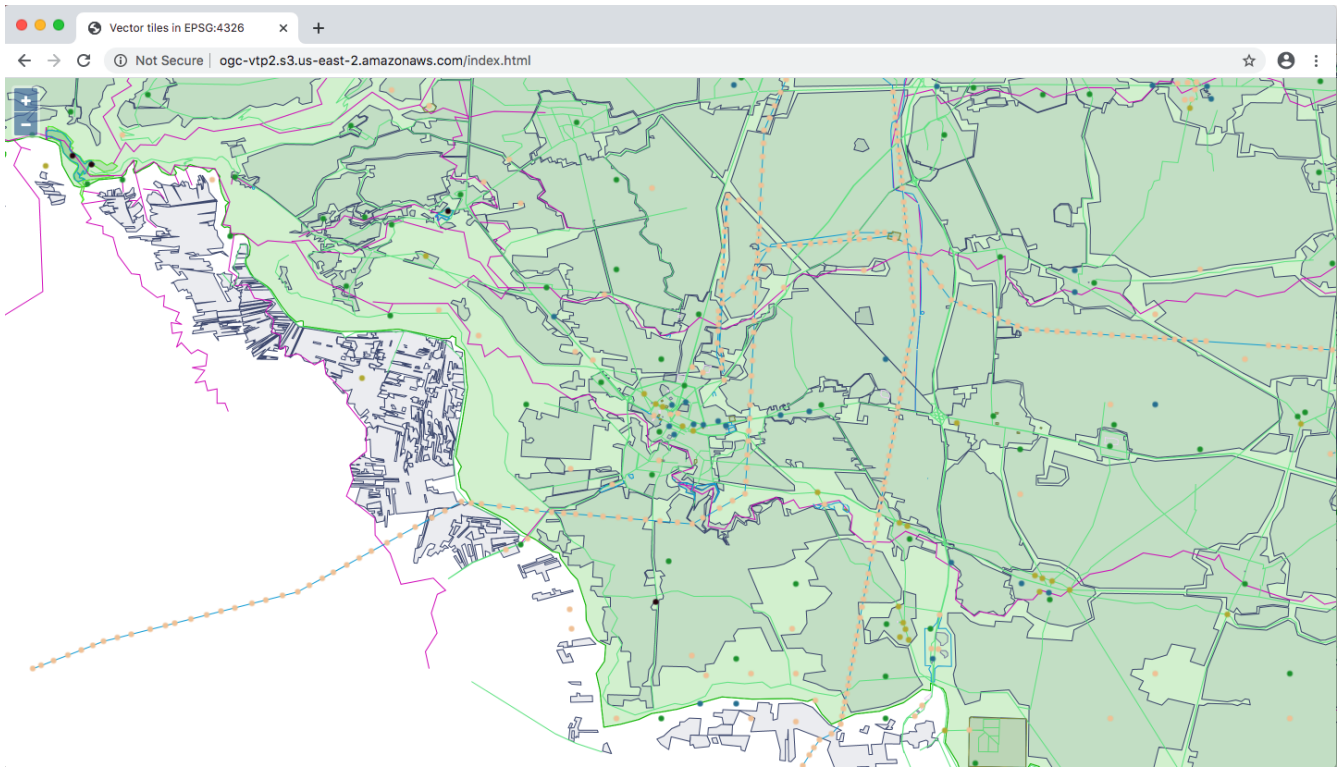


Figure 39. Client application built into the Terranodo Tegola vector tile server.

9.2. Client applications

This section describes the technologies and approaches taken for implementing the client application components of the VTP2 architecture.

9.2.1. Ecere D105 OGC API Client and D106 GeoPackage visualization

Ecere implemented cross-platform map client applications, shown in [Figure 40](#) and [Figure 41](#). These applications supported the Features, Tiles and Styles APIs, as well as offline tiled data stores. Capabilities for visualizing tile-based GeoPackages, including support for tiled vector and gridded coverage extensions were a subject of deliverable D106. The applications also supported the native GNOSIS Data Store, a compact offline tiled data store able to store coverage, imagery and vector data, balancing the number of files versus the size of each file. These capabilities were a continuation of Ecere's work in the first phase of the pilot, as well as OGC Testbed-13 (Vector Tiles), Testbed-14 (Symbology Conceptual Model) and Testbed-15 (Open Portrayal Framework).

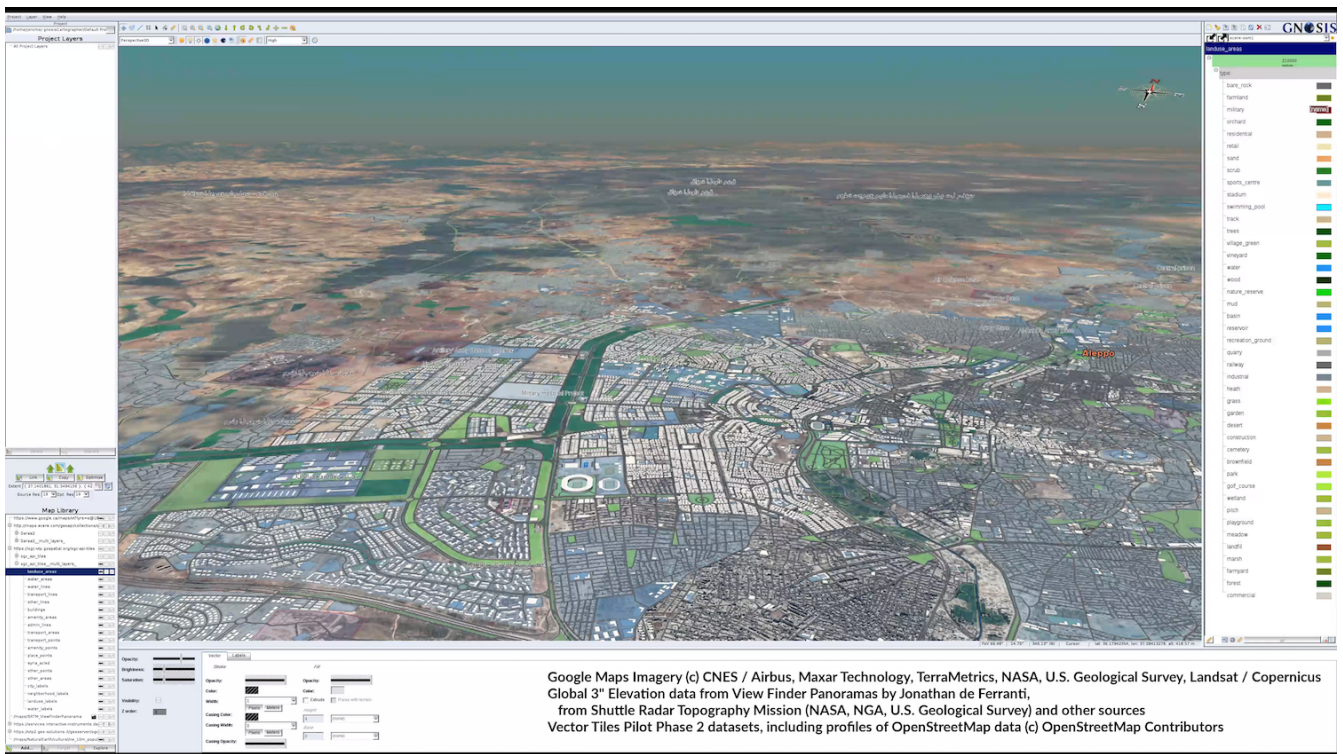


Figure 40. OpenStreetMap, Imagery and Elevation data from Syria displayed in GNOSIS Cartographer GIS tool

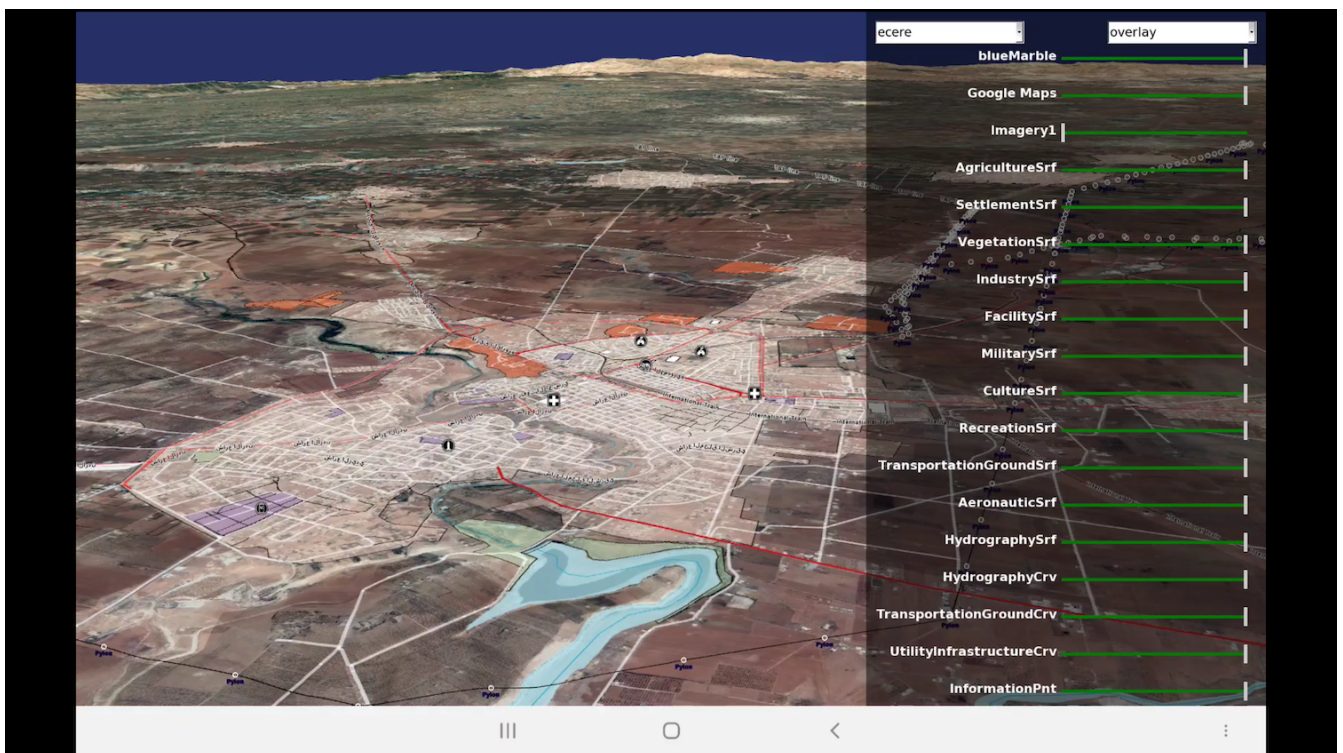


Figure 41. Screenshot of Ecere's mobile Android client built using cross-platform GNOSIS SDK

The design of the clients ensures consistent functionality online and offline, permitting to support operations faced with Degraded, Denied, Intermittent or Limited connectivity. GNOSIS Cartographer and the simple mobile Android client are built with Ecere's [GNOSIS Software Development Kit](http://ecere.ca/gnosis) [http://ecere.ca/gnosis], a geospatial visualization framework, and leveraging the Open-Source [Ecere SDK](http://ecere.org) [http://ecere.org], which also provides a cross-platform user interface toolkit. The toolkits and applications are written in the [eC programming language](http://ec-lang.org) [http://ec-lang.org], and offer bindings to additional programming languages such as C, C++ and Python.

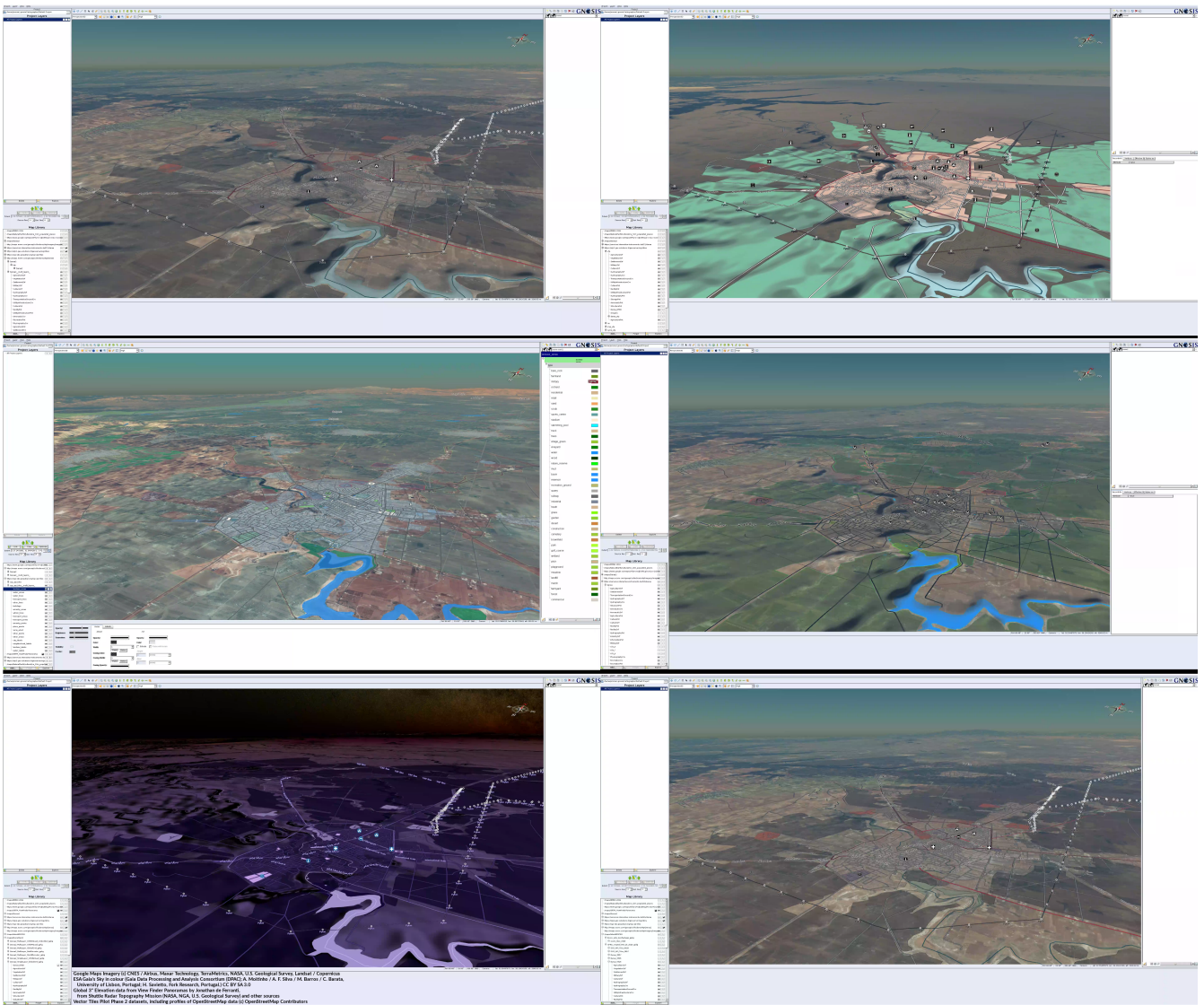


Figure 42. Tiled vector data from all Tiles APIs and GeoPackage producers shown in GNOSIS Cartographer: Ecere Tiles API (top-left), GeoSolutions (top-right), Terranodo (middle-left), interactive instruments (middle-right), Ecere GeoPackage (bottom-left) and Image Matters (bottom-right)

Adjustments were made to the client code to reflect the latest development of the Tiles API, including the use of TileJSON to better support multi-layer tiles. This enabled performing Technology Integration Experiments with all the different service providers of the pilot, including Ecere, interactive instruments, GeoSolutions and Terranodo. A screenshot showing tiled vector data retrieved from different servers and rendered on the Ecere client is presented in [Figure 42](#).

A large number of flavors and revisions of GeoPackages produced by both Ecere and Image Matters were tested throughout the pilot. Support for updated and new extensions for vector tiles, attributes, tiles/features mapping tables, spatial indexes, tile matrix set definitions, styles, semantic annotations and symbols was implemented in the client. For the Ecere GeoPackages, at least 9 different combinations of single or multiple layers per tiles, embedded attributes or attributes table, and 4 different TileMatrixSets). For Image Matters GeoPackages, each of them featured data retrieved from the Tiles API provided by all the different participants of the pilot, in multiple supported TileMatrixSets, which were all individually verified. Various performance improvements were also made to the rendering engine. Additional screenshots are presented in [Appendix B](#).

Support for two new tiling schemes was also implemented: WorldMercatorWGS84Quad and GNOSISGlobalGrid (as defined by the TileMatrixSet standard, using [Figure 18](#)). See the Ecere Tiles

API implementation for more details about those tiling schemes.

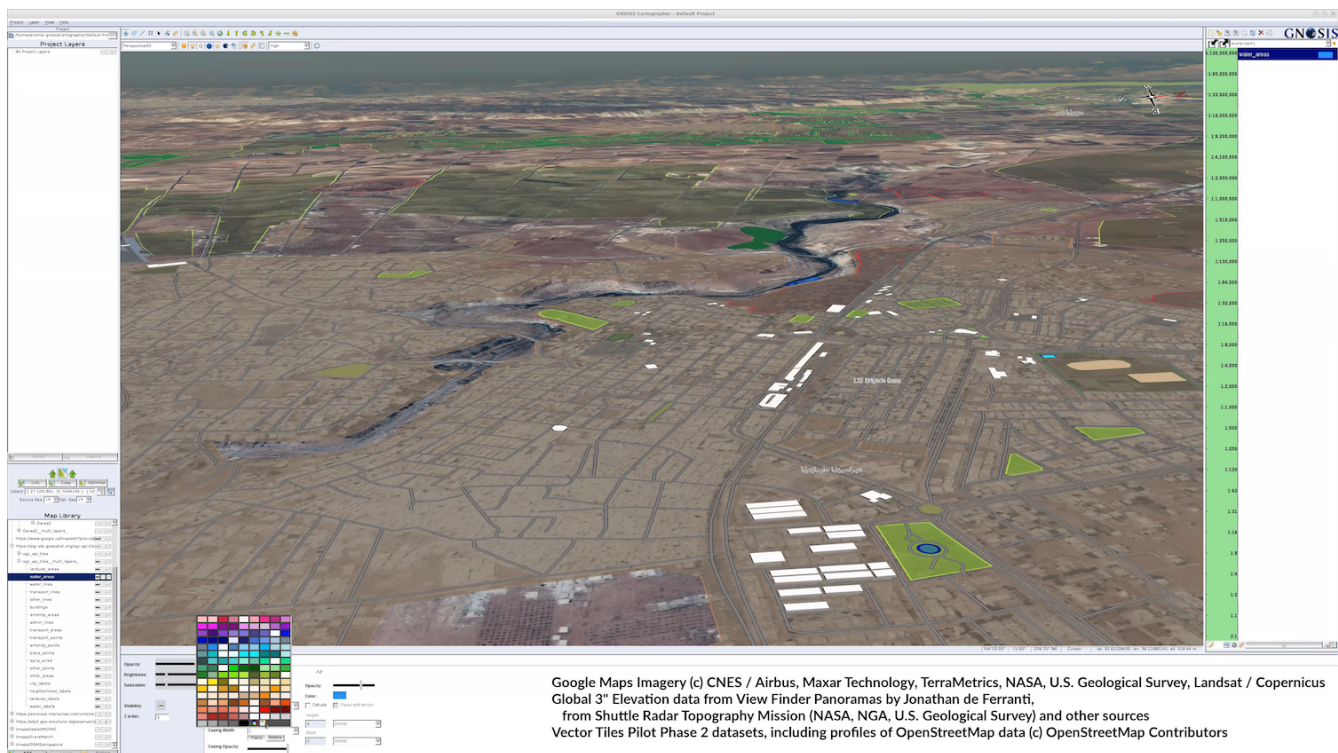


Figure 43. Tiled vector data from all Tiles APIs and GeoPackage producers shown in GNOSIS Cartographer: Ecere Tiles API (top-left), GeoSolutions (top-right), Terranodo (middle-left), interactive instruments (middle-right), Ecere GeoPackage (bottom-left) and Image Matters (bottom-right)

GNOSIS Cartographer's Visual Style Editor allows selecting symbology from an existing style, modifying or defining new symbology, and updating styles, whether they are stored within a GNOSIS Data Store or a GeoPackage, or are published through a Styles API. This functionality was improved and tested more in depth during the initiative. Support for functions was added to the GNOSIS Cascading Map Style Sheets (CMSS), including text manipulation and geometry / bounding box intersection, enabling new client-side styling and filtering capabilities.

The client automatically filtered features based on the style in use, as originally intended in the client design, but the filtering was applied on the client-side, as there was not enough time available to implement filtering requests to the service. The automatic filtering considers a style's selectors, as well as visualization properties such as opacity and visibility. This automatic generation of a filter could also be done by the server if referencing a style rather than supplying a filter. 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.

The ability to define 3D extrusion of polygons, where the heights can be defined as attributes-based expressions, was also used to render 3D OpenStreetMap buildings from both the Terranodo Tiles API (Figure 68) as well as an offline GNOSIS Data Store (Figure 69).

Testing was also performed with GeoJSON tiles sourced from the GeoSolutions Tiles API stored within a GeoPackage provided by Image Matters. As these tiles each contained multiple layers within a single GeoJSON files, that format was not yet properly supported by the GNOSIS visualization tools, which normally expect a separate GeoJSON per layer.

9.2.2. GeoSolutions D104 Client

GeoSolutions implemented a browser-based web map client application, shown in [Figure 44](#). The client application was implemented using MapStore a JavaScript framework based on [React](#) [<https://reactjs.org/>]. [MapStore](#) [<https://mapstore.geo-solutions.it/>] is an open source highly modular geospatial web framework for creating, managing and securely sharing maps and mashups. React is a JavaScript library for building user interfaces.

- [live demo](http://demo.vtp2.geo-solutions.it/mapstore/index.html#/) [<http://demo.vtp2.geo-solutions.it/mapstore/index.html#/>]
- [repository](https://github.com/geosolutions-it/ogc-vector-tiles-vtp/tree/master/vtp2) [<https://github.com/geosolutions-it/ogc-vector-tiles-vtp/tree/master/vtp2>]
- [offline client \(stand alone app for Windows\)](http://demo.vtp2.geo-solutions.it/mapstore-electron-client/windows.zip) [<http://demo.vtp2.geo-solutions.it/mapstore-electron-client/windows.zip>]

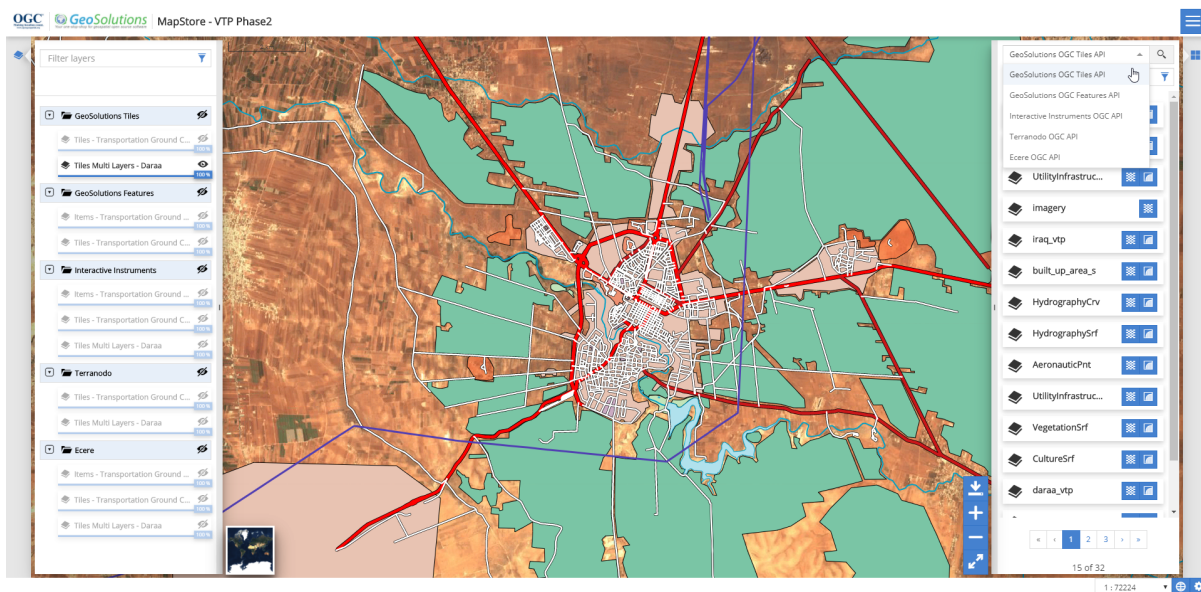


Figure 44. GeoSolutions Client in VTP2

A screenshot showing multi-layer collections retrieved from different servers and rendered on the GeoSolutions client is presented in [Figure 45](#). Additional screenshots are presented in [Appendix B](#).

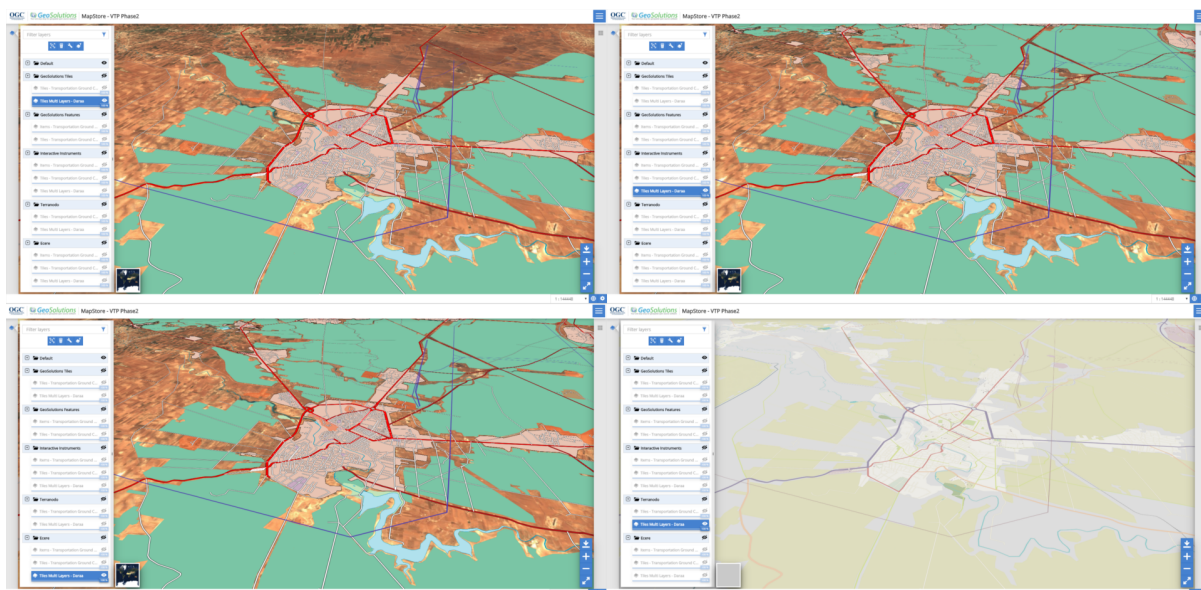


Figure 45. GeoSolutions MapStore Client shows multi-layer collection from different servers: GeoSolutions (top left), Interactive Instruments (top right), Ecere (bottom left) and Terranodo (bottom right)

The client implemented the following functionalities to be able to interact with all participants services:

- connected to servers supporting OGC API - Tiles [Figure 44](#)
- added single and multi-layer collections to the map [Figure 45](#)
- selected and applied styles to a vector tiles layer [Figure 46](#)
- built and applied a CQL filter to a single collection layer [Figure 47](#)
- generated and downloaded tile set metadata [Figure 48](#)
- visualized the downloaded tile set metadata on an offline client [Figure 49](#)
- selected and applied different tile matrix sets to a vector tile layer [Figure 57](#)

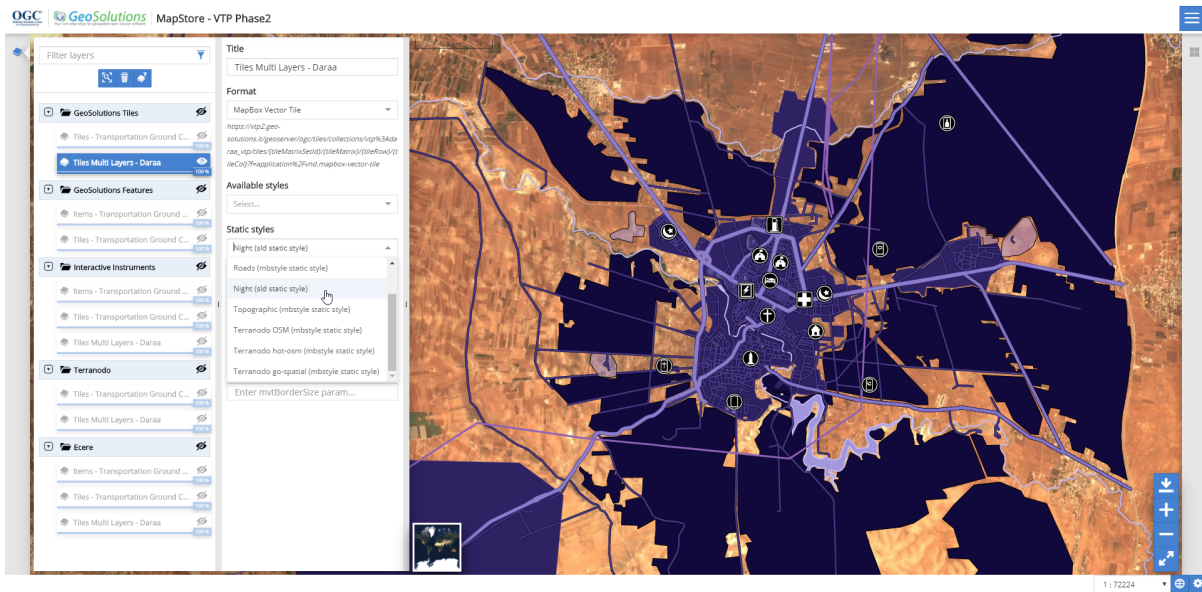


Figure 46. GeoSolutions MapStore Client selection of styles from GeoSolutions

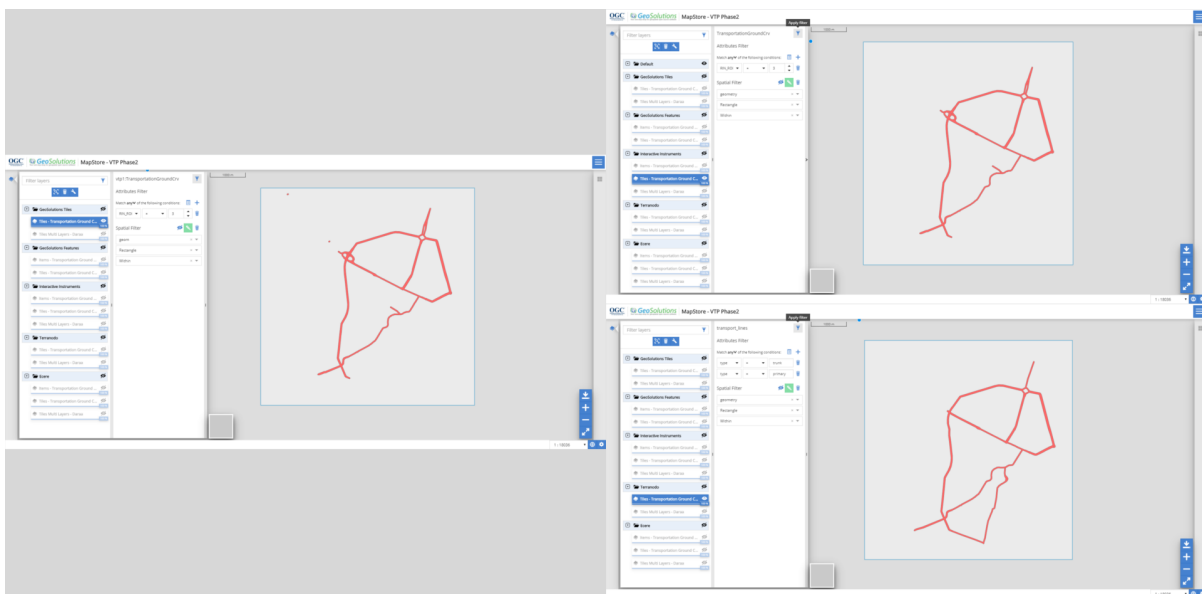


Figure 47. GeoSolutions MapStore Client with attribute and spatial filters applied to Transportation Ground Curve layer from different services: GeoSolutions (left), Interactive Instruments (top right) and Terranodo (bottom right)

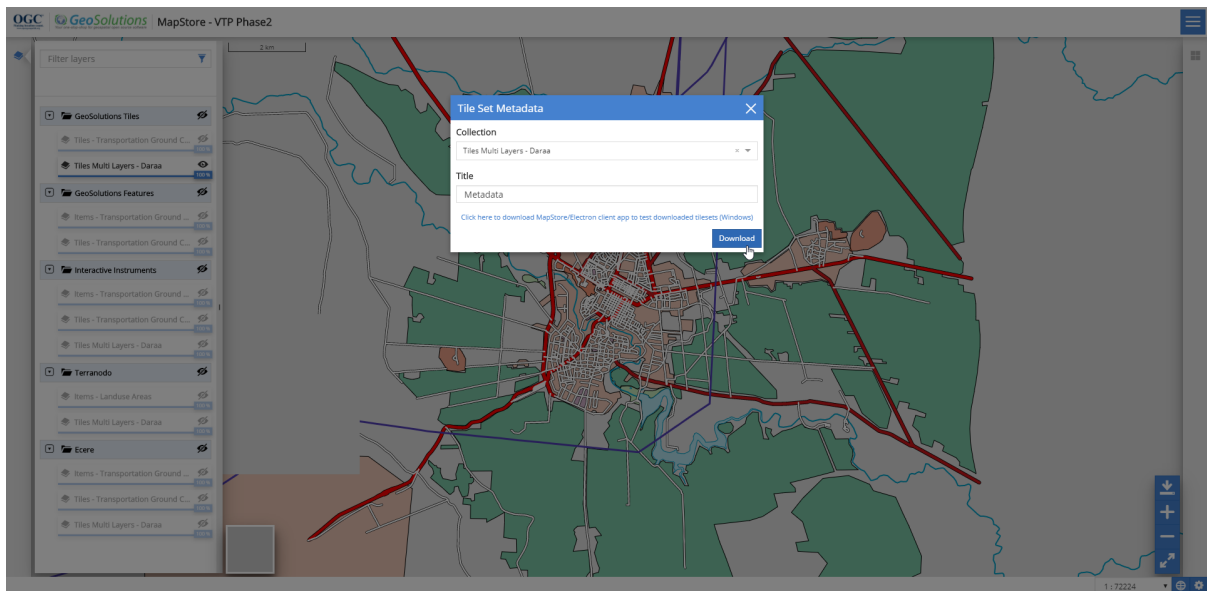


Figure 48. GeoSolutions MapStore Client displays the tile set metadata download form

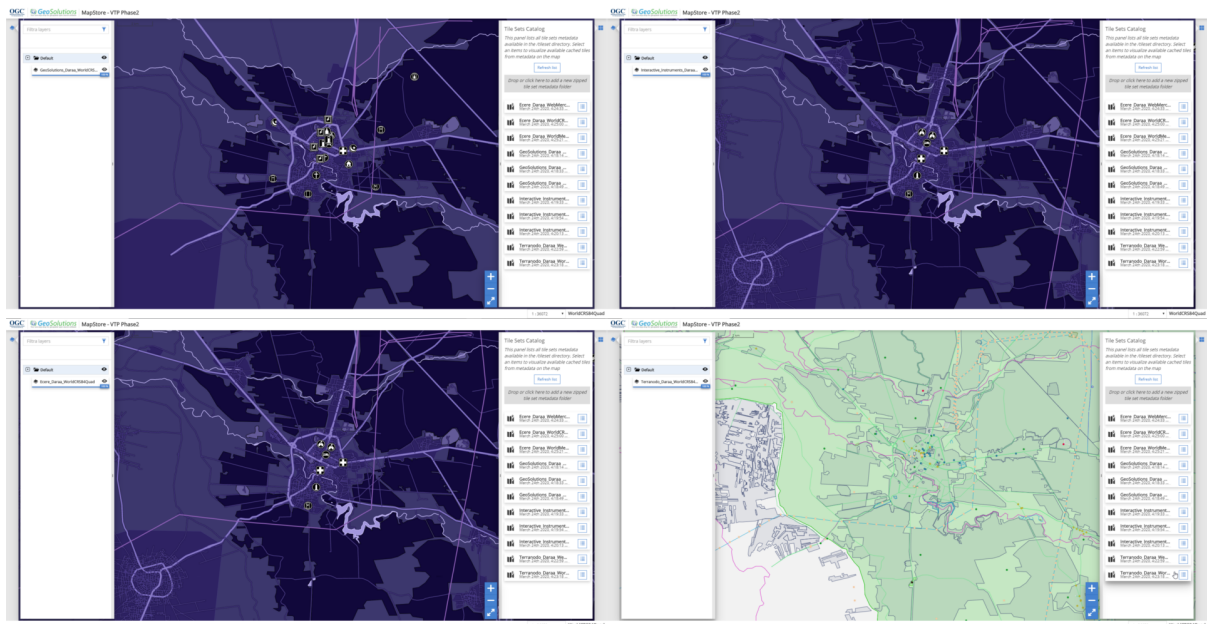


Figure 49. GeoSolutions MapStore Client in an Electron app displays downloaded tile set metadata offline from different servers: GeoSolutions (top left), Interactive Instruments (top right), Ecere (bottom left) and Terranodo (bottom right)

GeoSolutions carried out additional experimentation as listed below:

- tested support for Mapbox GL in MapStore [Figure 58](#)
- tested flatgeobuf format for Feature items [Figure 59](#)
- tested integration of MapStore in an Electron app (offline client) [Figure 49](#)

9.2.2.1. Some notes

The client was able to use styles from a Styles API implementation. If such a service is not available, the client was able to use styles available in a static folder. [Figure 46](#) shows selection of styles through the user interface of the GeoSolutions Client.

GeoSolutions also added the capability to choose between two JavaScript map libraries: OpenLayers

or Mapbox GL. Images [Figure 44](#) and [Figure 46](#) show the map rendered with OpenLayers while [Figure 58](#) shows a screenshot of the client application with vector tiles rendered using Mapbox GL.

9.2.3. Skymantics D104 Client

Skymantics sought to use MAIDEN, an indoor navigation platform that uses computer vision and augmented reality. MAIDEN uses the Unity engine and Vuforia SDK for augmented reality applications in a mobile platform. MAIDEN has been used in other OGC activities, such as the OGC 3D IoT Pilot, to import [IndoorGML](#) [<http://docs.opengeospatial.org/is/14-005r5/14-005r5.html>] as well as [3D Tiles](#) [<http://docs.opengeospatial.org/cs/18-053r2/18-053r2.html>] (in glTF format) for displaying air quality and room occupancy data. As the foundation for the Skymantics Vector Tiles Mobile Client, MAIDEN incorporates Mapbox SDK to render vector tiles on a map.

The initial proposed architecture for the Skymantics D104 Client is shown in [Figure 50](#).

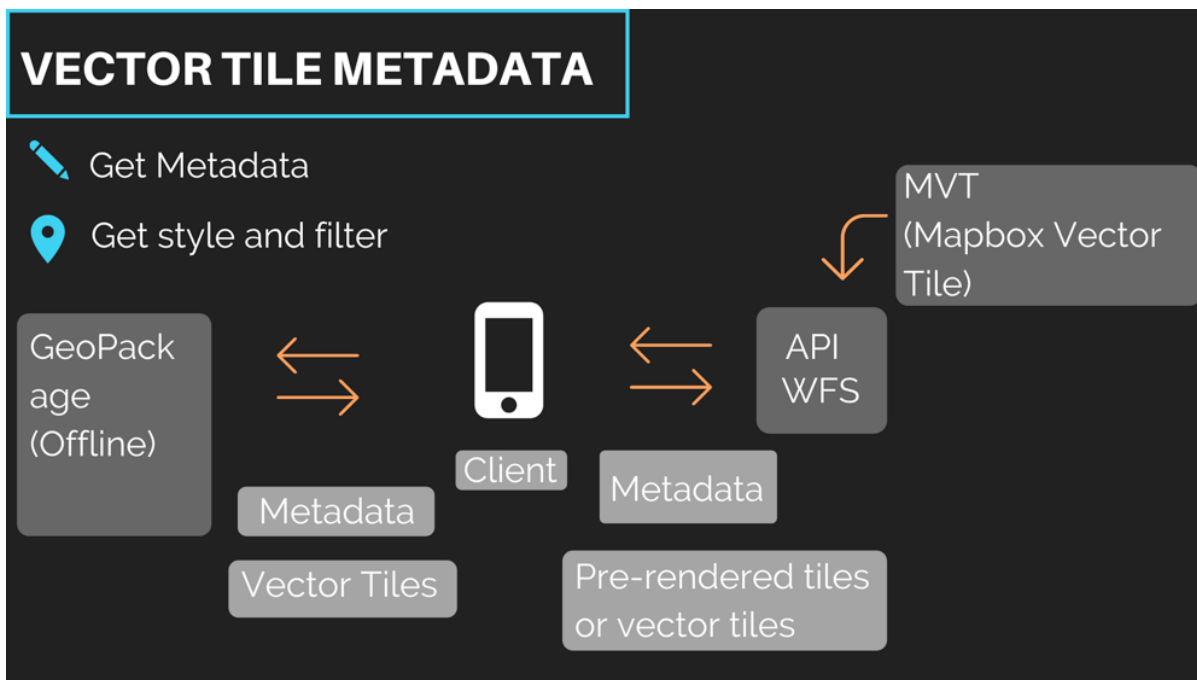


Figure 50. Skymantics D104 Initiative Architecture

9.2.3.1. Conceptual Idea

Skymantics selected Unity 3D to develop a map client with augmented reality capabilities. This software was combined with the Mapbox SDK and Mapbox Studio to provide a framework for map utilization. This decision was made for three main reasons:

- Mapbox Vector Tiles (MVTs) was the encoding chosen for tiles served by OGC APIs during this Pilot.
- Mapbox Documentation reviewed during the discovery phase of this Pilot stated that custom vector tile data could indeed be used, and custom API URLs were also able to be utilised with Mapbox SDK.
- Utilizing commercial off-the-shelf tools together with an extensively-used specification such as MVT would benefit interoperability and facilitate further implementations.

9.2.3.2. Components

9.2.3.2.1. Unity 3D

[Unity](https://unity.com) [https://unity.com] (also known as Unity 3D) is a cross-platform game engine developed by Unity Technologies. The engine can be used to create three-dimensional, two-dimensional, virtual reality, and augmented reality games, as well as simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering and construction.

Unity gives users the ability to create games and experiences in both 2D and 3D. The engine offers a primary scripting API in C#, in the form of plugins, and games themselves, as well as drag and drop functionality.

Unity allows importation of sprites and an advanced 2D world renderer. For 3D games, Unity allows specification of texture compression, mipmaps, and resolution settings for each platform that the game engine supports, and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects.

As of 2018, Unity has been used to create many of the mobile games on the market, as well as content for augmented reality and virtual reality applications.

One of the benefits of using Unity is that a 'slippy' map can be created and modified for custom use. Within the context of Unity, a 'slippy' map is an interactive 2D/3D map working with a variety of online tile providers (OpenStreetMap VirtualEarth/Bing Maps) and offline sources (DBMap, MBTiles).

9.2.3.2.2. Mapbox

Mapbox is an American provider of custom online maps for websites and applications such as Foursquare, Lonely Planet, Facebook, the Financial Times, The Weather Channel and Snapchat. Mapbox is the creator of, or a significant contributor to, some open source mapping libraries and applications, including the Mapbox GL-JS JavaScript library, the MBTiles specification, the TileMill cartography IDE, the Leaflet JavaScript library, and the CartoCSS map styling language and parser.

The data are taken from open data sources, such as OpenStreetMap and NASA, and from purchased proprietary data sources, such as DigitalGlobe. The technology is based on Node.js, Mapnik, GDAL, and Leaflet. Mapbox uses anonymised data from telemetry pings, such as Strava and RunKeeper, to identify likely missing data in OpenStreetMap with automatic methods, then manually applies the fixes or reports the issue to OSM contributors.

[Mapbox SDK for Unity](https://www.mapbox.com/unity/) [https://www.mapbox.com/unity/] allows developers to generate maps and location data optimized for Unity. Written from the ground up in C#, the Maps SDK for Unity unlocks global data to generate custom 3D worlds, power location lookup, and incorporate traffic-based directions in Unity. This SDK connects to Mapbox Studio through the Unity Inspector window in the Unity interface. Mapbox Studio is an online map design studio developed by Mapbox. Mapbox Studio is available for free to all registered users on mapbox.com. Its primary feature is a graphical style editor for authoring styles for Mapbox-hosted vector maps. It also includes a dataset editor that allows developers to import and edit GeoJSON data.

Mapbox has defined a specification for vector map tiles called 'vector-tile-spec' which uses Google protocol buffers for space-efficient data serialisation. Web Mercator is the projection of reference, but vector tiles may be used to represent data with any projection and tile extent scheme. It is also tied to the Mapnik rendering engine, using a serialized version of the internal data that Mapnik uses.

As illustrated in [Figure 50](#), the architecture supports MVT display interfacing with implementations of the OGC API - Features to retrieve metadata, selection of styles through the Styles API, and querying of filtered data based on metadata.

9.2.3.3. Challenges Encountered During Development

Skymanatics' initial approach was to use the Mapbox SDK to directly access the OGC Tiles API and Styles API to retrieve the tiles and render them within Unity. However, it was discovered that the SDK was preconfigured to interface with a valid Mapbox Studio API and a user ID key. This constraint appears to have been introduced after the Mapbox SDK 1.4 release. Furthermore, Mapbox restricts the styling options by only allowing its Mapbox Studio application to style tilesets. The implication of this restriction is that styles would not be applied directly in Unity, eliminating the possibility of dynamically styling maps in the mobile application as well.

Since Mapbox SDK does not connect to third-party APIs out-of-the-box, Skymanatics explored the open source nature of the SDK to modify the code to make it work with OGC APIs. However, it was discovered that the code structure was deeply nested, and modifying the function calls for the APIs would have required rewriting over 50% of their codebase.

A C# library called '[vector-tile-cs](https://github.com/mapbox/vector-tile-cs)' [https://github.com/mapbox/vector-tile-cs] is an open source code made by the same developers as Mapbox, so it was also tested in Mapbox SDK. The script included a location to enter the URL of an API to connect to, however, it was commented out. When the script is executed with an OGC API endpoint, the response noted that it required a valid Mapbox API.

Since Skymanatics was unable to access the OGC API directly through the Mapbox SDK, a custom C# script was developed to access the API and download the desired tilesets and store them on a local drive. It was assumed that once the Tiles were available within the Unity environment, that the Mapbox SDK would be able to render the tiles. However, this proved to not be the case.

Skymanatics explored the use of other rendering tools that had the potential for integration with Unity. One Unity plugin, [geojsontomesh](https://github.com/peted70/geojsontomesh) [https://github.com/peted70/geojsontomesh], was designed for use with the Microsoft HoloLens Platform, and enables JSON objects to be rendered as a Unity game object with a mesh overlay. Using this plugin, Tile features were able to be retrieved and displayed within Unity. However, it was found that this plugin was incompatible with the current Unity JSON plugins used for Skymanatics' client, and was not able to render the map images. Due to the time constraints of the pilot, this plugin was not investigated further.

Mapbox Studio allows clients to upload JSON encodings of Mapbox styles. Skymanatics connected to the Styles API to access the JSON styles encodings and loaded them into Mapbox Studio. However, during these TIEs with other participants, errors were encountered when accessing each participant's Style encoding. It was assumed that there were inconsistencies in the way each JSON style was created, which may be incompatible with the Mapbox style format. Manual creation of Mapbox styles within Mapbox Studio produced successful styling of the Tiles for rendering in

9.2.3.4. Final Implementation

After significant exploration in developmental approaches in which several lessons learned were recorded, a final implementation produced a successful demonstration of a mobile mapping application with augmented reality capabilities.

A C# script was developed to connect to the OGC APIs of the Pilot participants and download the tilesets and styles into a local file system as JSON files. Tilesets and styles were successfully accessed and retrieved from the APIs of all participants.

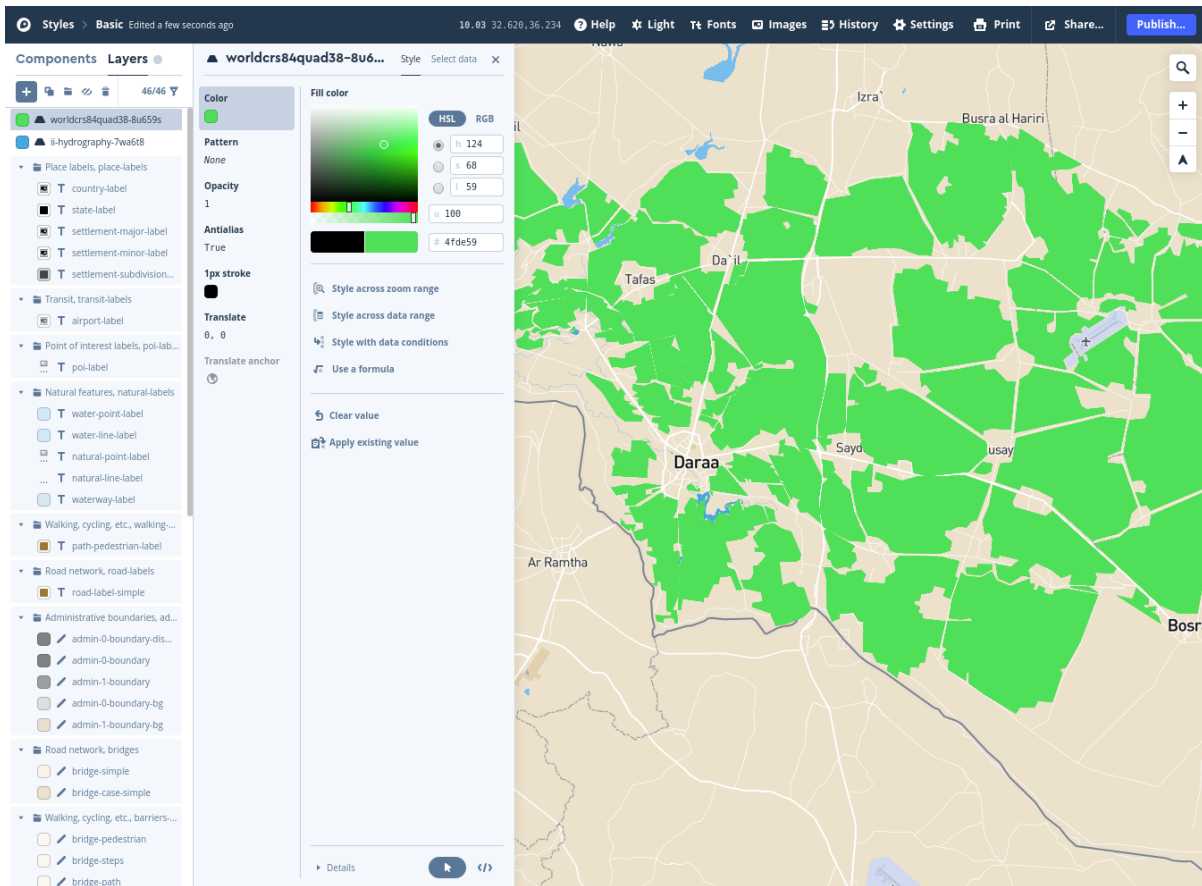


Figure 51. Skymantics D104 Daraa Agriculture Tileset Styled and Rendered in MapBox Studio

Next, these tilesets were uploaded to Mapbox Studio in accordance with Mapbox user guidelines for later retrieval using the Mapbox SDK. The tilesets were then styled using the Mapbox Studio built-in style editor to produce various styles such as night mode, etc. Once tilesets and loaded and styled (as seen in Figure 51), they were published and made available to Unity by using the user ID key and Mapbox API endpoint.

As seen in Figure 52 and Figure 53, the same Daraa map was loaded into Unity with the dark background (A.K.A. Night Mode) and light background (A.K.A. Day Mode) styles, respectively. The endpoint URLs of the combinations of tilesets and styles were tagged with a label in a Mapbox SDK.

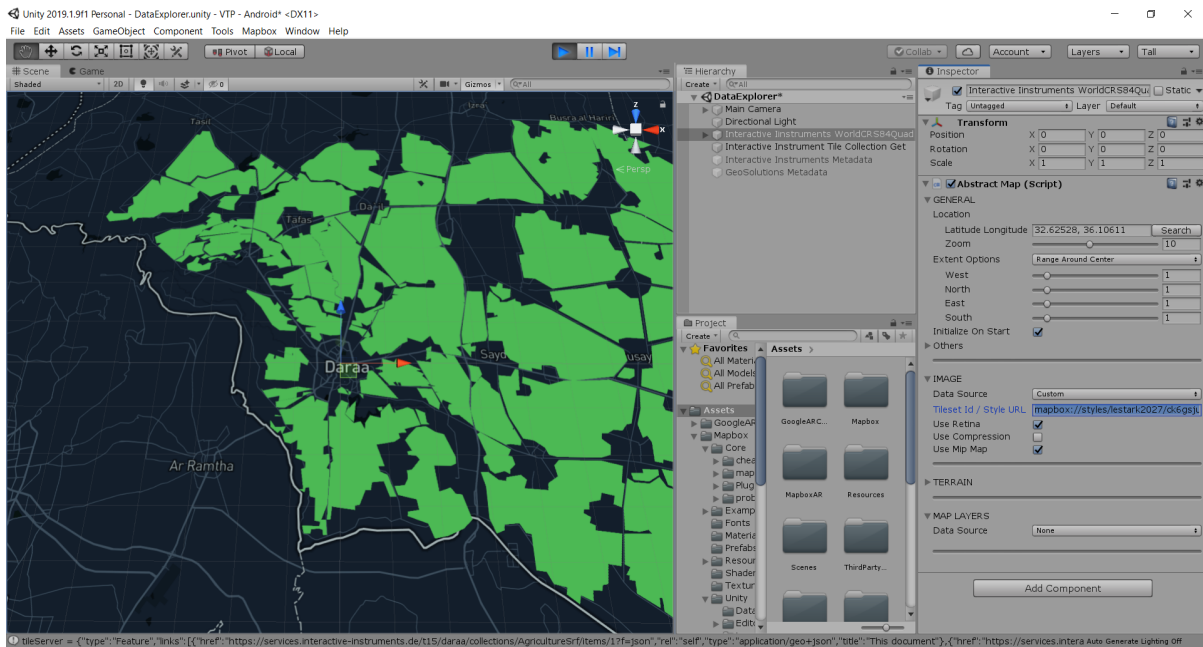


Figure 52. Skymantics D104 Daraa Tileset Rendering in Dark Style

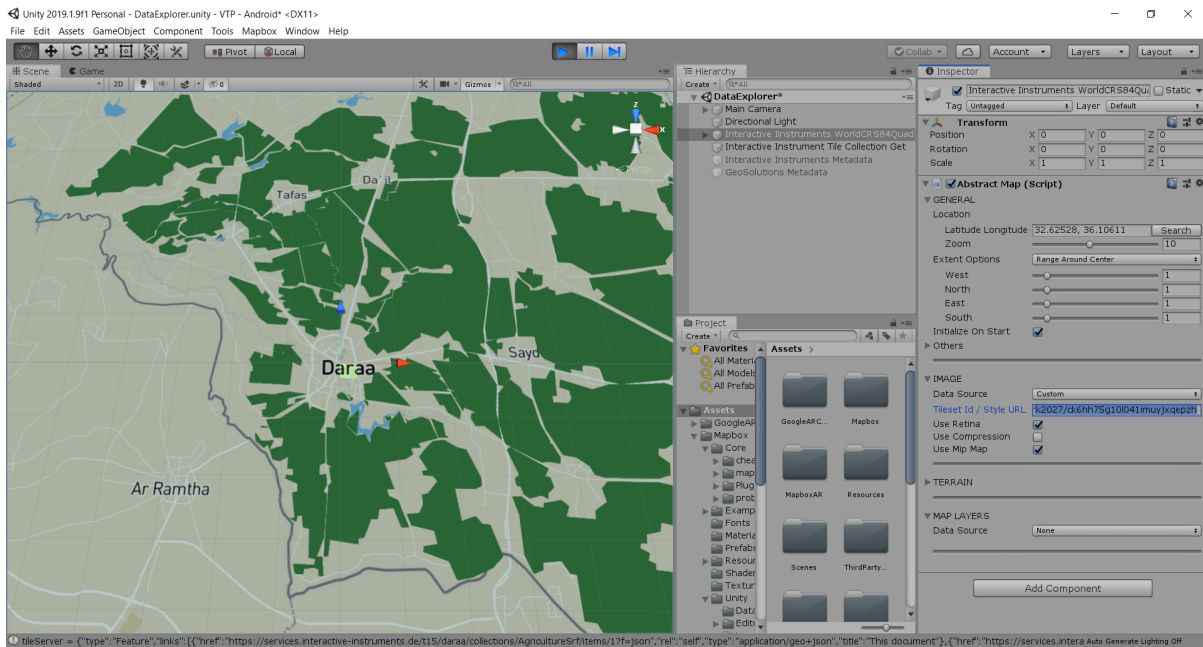


Figure 53. Skymantics D104 Daraa Tileset Rendering in Light Style

Once map rendering in Unity was successful, a mobile application was compiled. The mobile app can visualize the map in augmented reality by overlaying the map on top of camera vision; these AR capabilities were enabled through the Mapbox SDK and the [Google AR Core](https://developers.google.com/ar/discover) [https://developers.google.com/ar/discover] product. Upon initialization, the app used the device camera to detect a flat plane or surface and then rendered the map on the flat surface. As seen in [Figure 54](#), the map can be manipulated by dragging to move the map, and use of pinch and zoom gestures to focus in certain areas.



Figure 54. Map of Daraa Being Displayed in Skymantics' Augmented Reality Mobile Application

9.2.3.5. Future Work

The work carried out during this Pilot paved the way for future developments in mapping applications combined with augmented reality capabilities.

The largest challenge in this part of the project was the use of Mapbox SDK and Mapbox Studio with Unity, due to constraints on the manipulation of maps in Unity and consequently in the final mobile application. The challenge is described in detail in the Section titled '[Challenges Encountered During Development](#)'.

Future work should consider a number of different alternative approaches. One alternative is to render the maps directly in Unity through a custom-built Slippy map. Augmented Reality capabilities would be introduced by implementing Google AR Core directly into Unity. Tilesets and styles would be downloaded from OGC APIs using the scripts developed by Skymantics during this Pilot, and custom Unity C# scripts would enable dynamic filtering and styling capabilities within the mobile application.

A more accessible approach would be to continue using the Mapbox SDK for Unity, accepting its constraints of not allowing dynamic map loading, filtering and styling, but expanding the mapping capabilities of Unity. To accomplish this, future work could anticipate in advance a large number of combinations of layers, styles, and filters that an end user would be requiring, and make them easily available in the Augmented Reality mobile application. To do this, one Mapbox style per tileset/filter/style combination would be created in Mapbox Studio, and made available in Unity by means of individual 'scenes' that the end user would select in the mobile app. Each Unity 'scene' would access a specific Mapbox style URL previously created.

This approach would benefit from the following extra developments:

- Adopting more strict encodings of styles published by Styles API implementations for those styles to match Mapbox Studio specifications, enable direct upload of Styles, and avoid users the need of manually creating styles.

- Further exploring alternatives to programmatically uploading, filtering and styling tilesets in Mapbox Studio in order to trigger these commands from a Unity mobile application, giving users more control over the maps being rendered.

Chapter 10. Results

This section presents the results of implementing the various components. This section also discusses some of the issues and challenges encountered during the implementation, deployment and testing of the components.

10.1. Considerations

10.1.1. Portrayal information

This subsection presents a summary of the issues considered by the initiative participants in relation to portrayal information.

10.1.1.1. Styles

The Pilot participants considered a number of approaches for handling styles within the context of the architecture shown in [Section 7](#). The participants noted that while connected to a telecommunications network, an end-user could use a client application to request stylesheets from an implementation of the OGC API – Styles proof of concept that was developed in Testbed-15 [10]. Once the end-user retrieves the styles, the application could store those styles in a GeoPackage for use in DDIL environments.

The location for placing styles in the architecture was therefore considered. Participants noted that symbols are currently stored in a variety of places such as in a resources folder that could then be referenced by a GeoPackage. Typically, with vector tiles, an application that handles vector tiles would reference icons, styles, and stripes through URLs. The same goes for SLD. SLD documents can prescribe the rendering of symbols through an embedded Symbology Encoding element, but can also reference external symbols through URLs. There is no standardized location for symbols to be stored.

The VTP2 pilot designed an approach for storing styles, along with other business objects in a GeoPackage. One of the GeoPackage extensions designed in the Pilot enables Semantic Annotations which are annotations that are resolvable via a URI and can be placed on any business object (layer, feature, tile, style, etc.). Semantic annotations are implemented in GeoPackage through the Semantic Annotations Extension which is described in the section titled [Semantic Annotations](#). The metadata for styles is handled in the same way as metadata for other business objects; that is, the GeoPackage Metadata Extension is used to add metadata for any business object (layer, feature, tile, style, etc.) in the GeoPackage.

10.1.1.2. Links to styles

Another issue raised was how styles are accessed through the Styles API. Currently, the Styles API provides resource links to each stylesheet, the link relation type `stylesheet` and the media type of the style encoding in `type`. The `href` is then a "clickable", format-specific URI. Since the URL for a stylesheet of a specific format is unique, this approach makes access to stylesheets of different formats efficient because the URL for a format-specific stylesheet is unique. In order to have one link per style for all stylesheets, participants considered whether an alternative could involve use of a link relation type `style` without media type hints. This approach would require an alternate

mechanism to determine the available style encodings for a style. One option would be to implement HEAD support on the Style resource in combination with including the links in the HTTP headers. The `self/alternate` links could then be used to determine the available stylesheets. [Issue #2](https://github.com/opengeospatial/ogcapi-styles/issues/2) [https://github.com/opengeospatial/ogcapi-styles/issues/2] was created in the OGC API - Styles Github repository in order to advance this topic within the SWG.

10.1.1.3. Sprites

In addition to specifications of symbols, participants noted that sprites, glyphs and fonts are equally important. The handling of such artifacts both online and offline was therefore considered. In some cases, the same name is used for icons in a GeoPackage table and those of the symbol file. This enables an application to cross reference uses of the icon to the symbol file itself.

Extending this approach to sprites, participants noted that individual symbols that are entries in a GeoPackage could be given the same name as a JSON-encoded sprite configuration file, and then referenced from an SLD document. Then when styling with the Mapbox GL, an application could ignore the sprite URL and use the name of the actual ID into the symbols table. The JSON-encoded sprite configuration file could include information such as the name, height, width, pixel ratio, XY anchor, and storage location for each sprite.

Participants noted that Appendix B of the draft NSG Vector Tile Interoperability Specification has an example of a JSON encoding of a sprite configuration [11]. The specification describes how to access a sprite for a particular symbol.

10.1.2. Shared or Disjoint APIs

The participants made an observation that the OGC APIs that are being developed share some common patterns, which provides the APIs with a common foundation for paths and resources. The VTP2 participants referred to such an approach as a Shared API strategy. The collections path and support for bounding box (BBOX) querying are some of the patterns that provide a Shared API when implemented as described in the OGC API – Common draft specification.

In contrast, the participants also observed that an alternative to the Shared API strategy is the Disjoint API approach whereby each API has its own path scheme for accessing resources. Some resources may appear in two or more APIs such as features in Features API and Tiles API implementations. Further, a service that implements the Tiles API may operate independently or as a building block that is attached to another API. Therefore consideration of what types of constraints may be needed to facilitate interoperability between implementations of the Tiles API is necessary.

10.1.3. Static API

A Static API for serving tiles was another API strategy considered in the pilot. In such a strategy, tiles are provided through a web server simply as a system of files and folders where the hierarchical arrangement of the files documented in a JSON file that is also available through the same web server. This API strategy has the benefit of making it relatively easy to move large collections of tile sets from one storage system to another, as the location of the tiles can be inferred from the hierarchical arrangement of the folders. Moreover, should the JSON file be corrupted, it can be easily generated by reviewing the hierarchical arrangement of the folders.

A precondition for supporting a Static API is if the client application provides a request without any query parameters, a client application should get a reasonable response. Such a precondition was observed to be different from the approach taken by classic OGC web services (other than WMTS) which required that the SERVICE and REQUEST parameters be provided by a client application. The participants observed that with GeoPackage it is not the API that does the filtering, but instead it is the GeoPackage library. More specifically, the SQLite file itself does not have code for filtering. The SQLite library does the filtering by operating on the SQLite file.

Whereas a Static API offers the benefit of simplicity and ease of moving tiles from one storage to another, a Static API is also limited in its capabilities. For example, a Static API does not allow reprojection to other coordinate reference systems and would not allow support for returning the tiles in different media types.

10.1.4. Tile Cache vs. Tile Set

Initially, the participants considered a repository of tiles to be a cache and such a repository was initially referred to as a Tile Cache. However after some additional consideration, the participants noted that Tile Cache to some people means a location where a collection of tiles can be placed to service some client applications. For example, an end-user could create a cache and place it in a USB storage device, enabling a truck driver (in the VTP2 scenario) to bring up the tiles and display them on a dashboard mounted device while driving. Such a collection of tiles therefore has application in access and portrayal of information, not just in storage. The participants observed that it becomes confusing when the term Tile Cache is used because there are different aspects to the use of such collections of tiles such as storage, access and update. The participants therefore concluded that the term Tile Set was more appropriate for the context of the VTP2 initiative.

10.1.5. Out of bounds and empty tiles

In order to populate a GeoPackage in the VTP2 initiative, a client requests tiles and does not necessarily inspect the content. Since some tiles media types cannot be 0 bytes in size, there is a possibility that the client may not know whether the tile is empty (although Mapbox vector tiles can be of 0 bytes).

One potential solution would be that:

1. An HTTP 204 return code could be returned by the server, assuming the requested tile was within the TileMatrixLimits. This would allow 0-byte responses for media types which cannot be of 0 bytes. There are some arguments about whether that is proper or not for an HTTP GET request, but the HTTP specification does say that such a response is valid.
2. When a requested tile falls outside the TileMatrixLimit, but within the overall global TileMatrixSet definition, an HTTP 404 code should be returned (this should be in line with current draft specs).
3. However, if the tile falls outside of the global TileMatrixSet definition altogether, an HTTP 400 code would be more appropriate (this may not be reflected right now in the draft specs).
4. Also, VTP2 participants suggested two optional tile statuses, which could be included as a response header:
 - 'Empty all the way zoomed in' — for large areas where there is absolutely no data at any

zoom level even further down, but within the `TileMatrixLimit`. This would avoid having to request the levels further down and save tons of requests.

- 'Completely full tile' — for large polygons with detailed contours, e.g. lands and oceans as filled polygons. This would avoid having to request the levels further down and save tons of requests.

There are drawbacks to the potential solution described above:

- If the server is streaming the response, the server may start the response at a point when it is unknown whether the response will contain any features or not. In such a case, supporting the HTTP 204 approach would have a negative impact on the performance for fetching tiles that are not empty, at least in the situations where tiles are constructed dynamically.
- The additional status code would require special handling in clients.
- An HTTP 204 response to a GET request is valid HTTP. However, client developers may still find using HTTP 204 status codes for a GET request unusual if they are unfamiliar with its use in the context proposed above. There should be sufficient feedback from client implementors about whether they think that use of the HTTP 204 code is a good idea or not.
- An HTTP 204 response may be considered, by some, as the equivalent to a null/nil value without any content type. This would be different from an empty set in a content type. Related to this is also that the behavior would differ from the behavior of the Features resource in OGC API - Features (`/collections/{collectionId}/items`).

10.1.6. Collections

The concept of Collections as a resource was introduced by OGC API - Features - Part 1:Core and adopted by OGC API - Common. [Part 2](https://github.com/opengeospatial/oapi_common/blob/master/collections/OAPI_Common-Collections.adoc) [https://github.com/opengeospatial/oapi_common/blob/master/collections/OAPI_Common-Collections.adoc] of the draft OGC API - Common specification focuses on Collections. In the OGC API - Features standard, 'collection' is used as a synonym for 'feature collection'. A property called 'itemType' acts as an indicator about the type of the items in the collection. If an OGC API uses an `".../items/{id}"` path, then the `itemType` describes the resource type that a client application can access at that URL. In OGC API - Features it is "feature", whereas in the draft OGC API - Records specification it will be "record", etc. In some OGC APIs, `itemType` is not relevant and may be left out. The draft OGC API - Tiles specification illustrates the relationship between collections, tiles and maps as shown in [Figure 55](#).

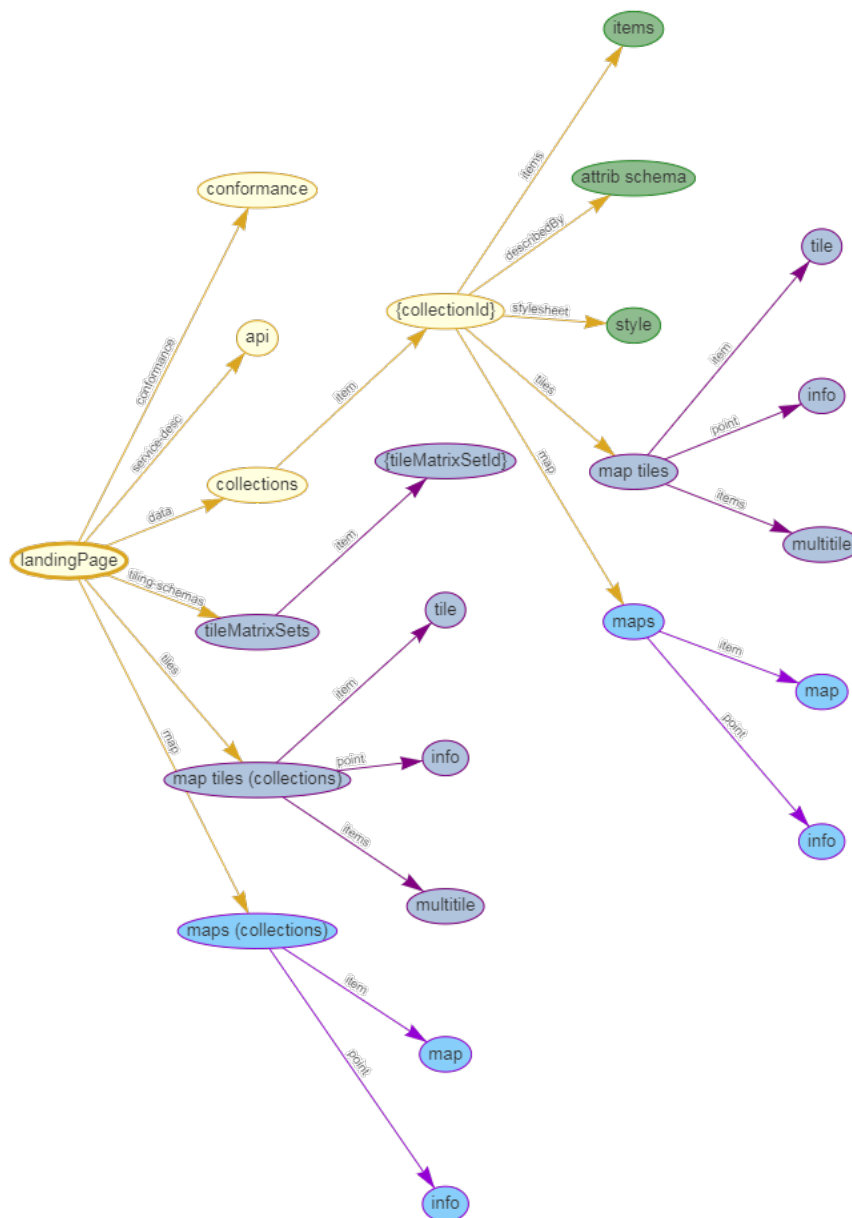


Figure 55. Relationships between Tiles, Maps and Collections in OGC API - Tiles

The VTP2 participants noted that, in some cases, the concept 'Collections' complicates matters because the content in the collections may be untyped. A collection might have a tiles link - or not. They could also have queryables and/or items. They might even have more collections. This forces a developer to create a very generic Collections class with a variety of methods which might or might not be applicable to the current situation. This approach may work but VTP2 participants noted that the approach is not very elegant because there is no telling what a client application could find in a particular document. An alternative approach would be to have clearly distinguished object types that report things such as feature collections, tile pyramids, and so forth.

10.2. Technology Integration Experiments

10.2.1. Data Exchange TIEs

An X is placed where there has been a successful exchange of vector tiles data between client and server/data producer.

An — is placed where a TIE is not required.

Table 8. Technology Integration Experiments (TIE)

	GeoSolutions D104 Client	Skymantics D104 Client	Ecere D105 Client	Image Matters Client
Ecere D100 Features API	X	X	X	X
Ecere D103 Tiles API	X	X	X	X
Ecere D101 Styles API	X		X	X
Ecere D106 GPKG	—	—	X	—
GeoSolutions D100 Features API	X	X	X	X
GeoSolutions D102 Tiles API	X	X	X	X
GeoSolutions D102 Styles API	X		X	X
interactive instruments D101 Features API	X	X	X	X
interactive instruments D101 Tiles API	X	X	X	X
interactive instruments D101 Styles API	X		X	X
Image Matters D107 GPKG	—	—	X	—
Terranodo D100 Features API	—	—	—	—
Terranodo D100 Tiles API	X		X	X
Terranodo D100 Styles API	X		X	X

NOTE Services on rows and Clients on columns

10.2.2. Styles Encoding TIEs

An X is placed where there has been a successful exchange of styles between client and server/data producer.

An — is placed where a TIE is not required.

Table 9. Technology Integration Experiments (TIE)

	GeoSolutions D104 Client	Skymantics D104 Client	Ecere D105 Client	Image Matters Client
Ecere D101 Styles API (MBStyles)	X		X	X
Ecere D101 Styles API (SLD)	X		X	X
Ecere D106 GPKG (MBStyles)	—		X	—
Ecere D106 GPKG (SLD)	—		X	—
GeoSolutions D102 Styles API (MBStyles)	X		X	X
GeoSolutions D102 Styles API (SLD)	X		X	X
interactive instruments D101 Styles API (MBStyles)	X		X	X
interactive instruments D101 Styles API (SLD)	X		X	X
Image Matters D107 GPKG (MBStyles)	—		X	—
Image Matters D107 GPKG (SLD)	—		X	—
Terranodo D100 Styles API (MBStyles)	X		X	X
Terranodo D100 Styles API (SLD)	—	—	—	—

NOTE | Services on rows and Clients on columns

10.2.3. Multi-Layer Tile TIEs

An X is placed where there has been a successful exchange of multi-layer vector tiles data between client and server/data producer.

An — is placed where a TIE is not required.

Table 10. Technology Integration Experiments (TIE)

	GeoSolutions D104 Client	Skymantics D104 Client	Ecere D105 Client	Image Matters Client
Ecere D103 Tiles API	X		X	X
Ecere D106 GPKG	—		X	—
GeoSolutions D102 Tiles API	X		X	X
interactive instruments D101 Tiles API	X		X	X
Image Matters D107 GPKG	—		X	—
Terranodo D100 Tiles API	X		X	X

NOTE Services on rows and Clients on columns

10.2.4. Multiple Projections TIEs

An X is placed where servers have provided vector tiles in multiple projections and clients have been able to display the vector tiles.

An — is placed where a TIE is not required.

Table 11. Technology Integration Experiments (TIE)

	GeoSolutions D104 Client	Skymantics D104 Client	Ecere D105 Client	Image Matters Client
Ecere D103 Tiles API - WorldMercatorWGS84Quad	X		X	X
Ecere D103 Tiles API - WebMercatorQuad	X		X	X

	GeoSolutions D104 Client	Skymantics D104 Client	Ecere D105 Client	Image Matters Client
Ecere D103 Tiles API - WorldCRS84Quad	X	X	X	X
Ecere D106 GPKG - WorldMercatorWGS84Quad	—	—	X	—
Ecere D106 GPKG - WebMercatorQuad	—	—	X	—
Ecere D106 GPKG - WorldCRS84Quad	—	—	X	—
Ecere D106 GPKG - GNOSISGlobalGrid	—	—	X	—
GeoSolutions D102 Tiles API - WorldMercatorWGS84Quad	X	X	X	X
GeoSolutions D102 Tiles API - WebMercatorQuad	X	X	X	X
GeoSolutions D102 Tiles API - WorldCRS84Quad	X	X	X	X
interactive instruments D101 Tiles API - WorldMercatorWGS84Quad	X	X	X	X
interactive instruments D101 Tiles API - WebMercatorQuad	X	X	X	X
interactive instruments D101 Tiles API - WorldCRS84Quad	X	X	X	X

	GeoSolutions D104 Client	Skymanatics D104 Client	Ecere D105 Client	Image Matters Client
Image Matters D107 GPKG - WorldMercatorWGS84Quad	—	—	X	—
Image Matters D107 GPKG - WebMercatorQuad	—	—	X	—
Image Matters D107 GPKG - WorldCRS84Quad	—	—	X	—
Terranodo D100 Tiles API - WorldMercatorWGS84Quad	—	—	—	—
Terranodo D100 Tiles API - WebMercatorQuad	X		X	X
Terranodo D100 Tiles API - WorldCRS84Quad	X		X	X

NOTE Services on rows and Clients on columns

10.2.5. TileJSON (Metadata) TIEs

An X is placed where there has been a successful exchange of Vector Tiles Metadata based on the VTP2 metadata model or TileJSON (via describedby rel link).

An — is placed where a TIE is not required.

Table 12. Technology Integration Experiments (TIE)

	GeoSolutions D104 Client	Skymanatics D104 Client	Ecere D105 Client	Image Matters Client	Terranodo Client (in kind)
Ecere D103 Tiles API	X		X	X	X
GeoSolutions D102 Tiles API	X	X	X	X	X

	GeoSolutions D104 Client	Skymanatics D104 Client	Ecere D105 Client	Image Matters Client	Terranodo Client (in kind)
interactive instruments D101 Tiles API	X	X	X	X	X
Terranodo D100 Tiles API	X	X	X	X	X

NOTE Services on rows and Clients on columns

10.2.6. Filtering TIEs

An X is placed where there has been a successful application of the VTP2 filtering approach.

An — is placed where a TIE is not required.

Table 13. Technology Integration Experiments (TIE)

	GeoSolutions D104 Client	Skymanatics D104 Client	Ecere D105 Client	Image Matters Client
Ecere D100 Features API				—
Ecere D103 Tiles API				—
Ecere D106 GPKG	—			—
GeoSolutions D100 Features API	X			—
GeoSolutions D102 Tiles API	X			—
interactive instruments D101 Features API	X			—
interactive instruments D101 Tiles API	X			—
Image Matters D107 GPKG	—			—
Terranodo D100 Features API	—	—	—	—
Terranodo D100 Tiles API	X			—

NOTE | Services on rows and Clients on columns

Chapter 11. Key Findings

Several of the considerations are discussed in [Section 10](#). The following subsections focus solely on key findings.

11.1. Bounding box inconsistency

Bounding boxes are expressed inconsistently in the various APIs used in this pilot. Some participants took issue with the varied approaches. Of particular concern is the coordinate order. Some bounding boxes are composed of an array of numeric values. Other bounding boxes have a defined pair of corners (e.g., lower left and upper right) with numeric values.

Some OGC standards declare a coordinate order and others allow the coordinate order to be declared through the specified Coordinate Reference System (CRS). It is even possible for the specification and the CRS to conflict. For example, consider the following JSON block:

```
"boundingBox" : {  
  "crs" : "http://www.opengis.net/def/crs/EPSG/0/4326",  
  "lowerCorner" : [  
    -90,  
    -180  
  ],  
  "upperCorner" : [  
    90,  
    180  
  ]  
}
```

The underlying specification requires (x, y, [z]) order but this is contradicted by the CRS which specifies (lat, lon) order. Some participants viewed the need to resolve the CRS as an undesired complication that is not necessary to calculate tile boundaries. Other participants viewed it necessary for applications to enforce consistency between the declared CRS and the coordinate values.

While ISO 6709 specifies (latitude, longitude) coordinate order for geographic point locations, modern schema-based encodings like JSON and XML make this approach unnecessary. A better approach may be for the schema to explicitly define the values (rather than using arrays) so that there is no potential confusion. However, this needs to be addressed in an OGC-wide way, perhaps as an update to [OGC 08-038r7, Revision to Axis Order Policy and Recommendations](#) [https://portal.opengeospatial.org/files/?artifact_id=76024].

11.2. Online and offline access to multi-layer vector tiles can be supported by OGC APIs and GeoPackage

A key finding of the pilot is that an architecture that enables consistent online and offline support of tiled feature data can be implemented using the OGC API - Features and GeoPackage standards,

as well as the draft OGC API - Tiles specification. This was shown by the successful implementation of the VTP2 pilot architecture.

11.3. The absence of a way to describe schemas creates difficulty for data transfer from an OGC API to a GeoPackage

With regard to the OGC API – Features standard, another finding is that the absence of a way to describe schemas makes it difficult to transfer data from a Features API endpoint to an offline datastore such as a GeoPackage. This is because a GeoPackage relies on a schema to structure tables. Whereas a queryables resource could address some of the requirement, descriptions of queryables are designed to be simple and less detailed than schema descriptions. Applications can, of course, still download a complete dataset and then parse every property in order to build a new schema. However, such a workaround would be computationally expensive and less efficient than reading a schema document that is already published by the API. There is therefore a need for a standard way for implementations of the Features API to describe the schemas of the feature datasets they publish.

The VTP2 participants noted that the OGC API - Features SWG has identified a potential future work item for an OGC API - Features extension that would offer schema support. The discussion on this need is recorded in an [issue](https://github.com/opengeospatial/ogcapi-features/issues/56) [https://github.com/opengeospatial/ogcapi-features/issues/56] on the SWG's GitHub repository.

This finding is supported by the successful TIEs that involved transferring data from the OGC API implementations to the GeoPackage implementations. Participants worked around this problem by using the draft OGC API – Tiles specification and accessing tile set metadata in the TileJSON format.

11.4. There is a need for generic collections that do not advertise much about their contents

Another finding related to the draft Tiles and Common API specifications, is that there is a need for "generic" collections that do not advertise much about their contents.

This ability would address the situation in which one wants to publish a collection that may, at anytime, consist of a variety of data objects. The collection, for instance, could represent a container (e.g. OWS Context) that is able to hold any type of resource. This could allow for an OGC API that is akin to the Web Object Service (WOS) that was demonstrated in OGC Testbed-12 [12].

11.5. There is a need for multi-layer tiles without having explicit collections

Another finding related to the draft Tiles and Common API specifications, is that there is a need for an ability to offer multi-layer tiles without having explicit collections. Such a capability would address the situation in which one wants to cascade a common XYZ tile service offering MVT files. This involves wrapping the XYZ tile service in a standalone Tiles API (as in, one that does not offer

feature or coverage resources on the side), without exposing the internal layering structure as separate collections. In such a case, all that is desired is to offer tiles and not the separate collections.

11.6. The need to test larger datasets

Throughout the Pilot, the dataset size was presented as a risk factor when making architecture design decisions. The dataset used in the pilot was appropriate for the use case and scenario applied, however, there was a risk that some of the design decisions made during the Pilot may not scale to larger datasets. To mitigate this risk and to further test the implementations developed during this Pilot, future initiatives should make use of larger datasets for testing. For example, OGC Testbed-16 makes use of a significantly larger dataset from Ordnance Survey.

11.7. Proposed GeoPackage extensions are ready for progression

The proposed GeoPackage Extensions that support Mapbox Vector Tiles and GeoJSON Vector Tiles are mature enough for further discussion and development by the GeoPackage SWG. The extensions were developed in both the VTP2 and Testbed-15 initiatives. Implementations of the extensions have been demonstrated to work, in both initiatives.

11.8. Consistency of schemas across some OGC resources could be improved

A participant attempted to use OGC resources for tile matrix set definitions. OGC provides tile matrix set definitions at the URL template <http://schemas.opengis.net/tms/1.0/json/examples/{TileMatrixSet}.json>. However, this approach had to be discarded when it was discovered that documents provided by this URL template do not follow a consistent schema.

Chapter 12. Recommendations

This section presents the recommendations and ideas for future work.

12.1. Key recommendations

12.1.1. Publish schemas through the Tiles and Features APIs

At the time of publication of this ER, there was no requirement nor specification within the OGC API suite for publishing schema information for a feature layer. This is a problem for clients of tiled feature data because the schema can only be ascertained by scanning all the features contained in all tiles and discovering their complete set of properties.

There are use cases where this is reasonable. However, this will not scale in the tiled feature data use case. The VTP2 participants agreed that there should be a common way to express this information and that servers serving tiled feature data should make use of this capability.

12.1.2. Development of a standard for Tile Set Metadata

As described in the OGC Vector Tiles Pilot 2: Tile Set Metadata Engineering Report [7] the VTP2 participants successfully designed and implemented a Tile Set Metadata Model. A relevant OGC SWG could therefore review and consider developing a standard based on the outputs of this initiative. As the tiling of 2D vector data affects several different domains, a new SWG may be needed for developing a standard for Tile Set Metadata. Alternatively, the charter of an existing SWG could be adapted by the TC to include development of a standard for Tile Set Metadata.

12.1.3. Introduce additional concepts in the OGC Symbology Conceptual Core Model

The OGC is currently developing the Symbology Conceptual Core Model (SCCM). Based on the requirements of VTP2, the participants of VTP2 recommend that the Portrayal DWG and the SLD SWG (which is responsible for the SCCM) should discuss and potentially include the following concepts in the SCCM, all of which are defined in the [Terms and definitions](#) clause of this ER:

- sprite
- stylable layer set
- stylesheet
- style encoding
- symbol
- symbol content

12.1.4. OGC API - Tiles should provide direction on multi-layer vector tiles schemas

VTP2 participants noted that developers want to deliver multi-layer Mapbox Vector Tiles (MVTs)

using the Tiles API. Further, developers would like to have a good way to advertise the structure of the layers found in the vector tiles, along with their attributes for cases where these are available and stable across tiles.

One possible approach would be to deliver each layer as a collection, each collection having a 'describedby' link relation that refers to a resource providing information about the link's context. The approach would also involve gathering the multi-layer vector tiles via the "tiles from more than one collection" requirement class. A client needing a schema would then scan over the collections about to be requested and gather the schemas. This is, however, not ideal in all scenarios. For example:

- One might not want to make the individual collections available.
- One might not want to expose the "tiles from more than one collection" extension.

Another possible approach the initiative participants would like to explore is serving multi-layer MVT files from a single collection in the Tiles API and then having a single 'describedby' link reporting the structure of the multi-layer vector tile, possibly as a TileJSON document. VTP2 participants recommend that the draft OGC API - Tiles specification be updated to provide some direction in this regard.

12.1.5. OGC API - Tiles should allow for multi-layer tiles that are without explicit collections

VTP2 participants noted that developers want to deliver multi-layer MVTs, that are without explicit collections, through the Tiles API.

One possible approach would be to deliver each layer as a collection and then build multi-layer tiles via the "tiles from more than one collection" requirement class that is specified by the draft OGC API – Tiles specification. This is however not ideal in all scenarios. For example:

- One might not want to make the single collections available
- One might not want to expose the "tiles from more than one collection" extension

The participants also noted that an alternative approach would be to serve multi-layer MVT files from a single collection in the OGC Tiles API, as there is nothing preventing use of an "itemType: unknown" for this purpose.

The VTP2 participants therefore recommend that the concept of a Collection is given a definition that is general enough (in OGC API - Common) to allow service implementers to advertise certain types of structure (schemas, MVT tile layer descriptions) when such resources are available and useful. This should be considered by the OGC API - Common SWG in its work on OGC API - Common - Part 2: Collections. Such generalization of the concept would make it useful to other data models (e.g. a collection as a map or as a coverage).

12.1.6. TileJSON editors should register a media type for the specification

Currently there is no registered media type for TileJSON. This creates difficulty for applications attempting to retrieve TileJSON files from an API. There is a need for a media type for TileJSON to be registered with Internet Assigned Numbers Authority (IANA), instead of using "application/json".

NOTE | TileJSON is not an OGC standard.

VTP2 recommends that the editors of TileJSON should be encouraged to register a media type for TileJSON at IANA. An example of how the draft OGC API - Tiles specification could support TileJSON, with the WorldCRS84Quad, is shown in the scheme at: <https://services.interactive-instruments.de/t15/daraa/tiles/WorldCRS84Quad/metadata>

NOTE | V3 document: <https://github.com/mapbox/tilejson-spec/tree/3.0/3.0.0> (unlike version 2.2, this document contains a list of layers and attributes, among other things).

12.1.7. OGC API - Common should allow for collections that do not advertise their content types

As stated in [Section 11](#), the pilot found a need for "generic" collections that do not advertise much about their contents. In the case of tiles, the Tiles API would allow tiles to be offered without exposing the internal layering structure as separate collections. Implementations may also provide sufficient metadata for client applications to efficiently retrieve the collections. VTP2 recommends that OGC API - Common should be updated to allow for generic collections that do not advertise their content types. This recommendation was posted as a comment to [issue #36](#) [https://github.com/opengeospatial/oapi_common/issues/36#issuecomment-598086780] on the OGC API - Common GitHub repository.

12.1.8. Development of Mapbox Vector Tiles and TileJSON as OGC Community Standards

The pilot demonstrated the potential utility of TileJSON and Mapbox Vector Tiles (MVT) within implementations of OGC API - Tiles. The originators of the TileJSON and MVT specifications should be encouraged to submit the specifications into the OGC for consideration as OGC Community Standards.

This would require the originators of the specifications to submit the specifications into the OGC process.

NOTE | TileJSON and MVT are not OGC standards.

12.2. Future Work

The following were identified by the pilot participants as areas for future work in OGC Innovation Program initiatives such as pilots and testbeds:

- Exploring the use of Collections.
- Variable Width Tile matrices (two examples being [GNOSISGlobalGrid](#) [<http://schemas.opengis.net/tms/1.0/json/examples/GNOSISGlobalGrid.json>] and [CDBGlobalGrid](#) [<http://maps.ecere.com/geoapi/tileMatrixSets/CDBGlobalGrid>]). One of the reasons for looking into variable width tile matrices is that currently one needs four Tile Matrix Sets to cover the Earth pole-to-pole, when using current non-WebMercator TMS. It would therefore be useful to have a Tile Matrix Set that can cover the Earth pole-to-pole but that has better accuracy than WebMercator.

- Use cases that do not only focus on visualization, e.g. the ability to re-construct geometry in support of data analysis. In this regard, a compact format that can be streamed, for example [FlatGeoBuff](https://github.com/bjornharrtell/flatgeobuf) [https://github.com/bjornharrtell/flatgeobuf], could be investigated. For Mapbox vector tiles, future use cases could include data extraction that is geared towards analysis and editing rather than visualization and interactivity.
- Implementation of tiled data as part of processing workflows (e.g. in parallel and/or distributed processing environments).
- Better specification of the CRS used in tiled GeoJSON (CRS84 vs the tile matrix set's own CRS) needs to be analyzed.
- Experimentation with FlatGeobuf should occur. FlatGeobuf is a variation of geobuf geared towards simple features and focusing on performance and compactness.
- Development of beta compliance tests for OGC API - Tiles, OGC API - Styles, OGC API - Maps, and others that are needed. The compliance tests would be implemented as executable test suites for running in TEAM Engine - OGC's validator.
- Testing Larger Datasets.

Appendix A: Proposed GeoPackage Extensions (Informative)

A.1. Vector Tiles Extension

Extension Title

Vector Tiles

Introduction

The GeoPackage Vector Tiles extension defines the rules and requirements for encoding tiled feature data in a format commonly known as "vector tiles" into a GeoPackage data store.

WARNING

This extension does not define an encoding for vector tiles. To be usable, an additional extension such as [Mapbox Vector Tiles Extension](#) or [GeoJSON Vector Tiles Extension](#) must also be used.

This extension, like all GeoPackage extensions, is intended to be transparent and to not interfere with GeoPackage-compliant software packages that do not support the extension.

Extension Author

Image Matters LLC, in collaboration with the participants of the OGC Vector Tiles Pilot and OGC Testbed-15.

Extension Name or Template

`im_vector_tiles` (If this extension is adopted by the OGC, then `gpkg_vector_tiles` will be named as an alias.)

Extension Type

This extension provides new requirements dependent on GeoPackage [Clause 2.2 \(tiles\)](#) [<http://www.geopackage.org/spec121/index.html#tiles>].

Applicability

This extension defines an alternate way to encode feature information into a GeoPackage.

Scope

read-write

Specification

If this extension is in use, then all of the [Tiles Option](http://www.geopackage.org/guidance/getting-started.html#tiles) [http://www.geopackage.org/guidance/getting-started.html#tiles] applies. The individual tiles (`tile_data` in a tile pyramid user data table) are vector tiles.

NOTE Individual vector tiles MAY be deflate compressed.

`gpkg_extensions`

To use this extension, add the following rows to this table.

Table 14. `gpkg_extensions` Table Rows

table_name	column_name	extension_name	definition	scope
<code>gpkgext_vt_layers</code>	NULL	<code>im_vector_tiles</code>	<i>a reference to this file</i>	<i>read-write</i>
<code>gpkgext_vt_fields</code>	NULL	<code>im_vector_tiles</code>	<i>a reference to this file</i>	<i>read-write</i>

NOTE The values in the `definition` column SHOULD refer in some human-readable way to this extension specification. If the extension is adopted by the OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

`gpkg_contents`

Like any other content type, add a row for each tile set, using a `data_type` of "vector-tiles".

`gpkg_spatial_ref_sys`

Like any other content type, the Spatial Reference System (SRS) for the content to be stored must be registered in this table. See clause 1.1.2 in the core GeoPackage standard. While any valid SRS may be used, Web Mercator (EPSG:3857) maintains compatibility with MVT.

New Table Definitions

There are two additional required metadata tables, `gpkgext_vt_layers` and `gpkgext_vt_fields`, that mirror the [vector_layers key from the JSON object from the metadata from MBTiles](https://github.com/mapbox/mbtiles-spec/blob/master/1.3/spec.md#vector_layers) [https://github.com/mapbox/mbtiles-spec/blob/master/1.3/spec.md#vector_layers]. This allows client software to understand the feature schemas without having to open individual tiles. As with other GeoPackage tables, this extension takes no position on how either of these tables are to be used by a client.

`gpkgext_vt_layers`

The `gpkgext_vt_layers` table describes the layers in a vector tiles set. The columns in this table are:

- `id` is a primary key
- `table_name` matches the entry in `gpkg_contents`
- `name` is the layer name
- `description` is an optional text description

- `minzoom` and `maxzoom` are the optional integer minimum and maximum zoom levels
- `attributes_table_name` is the optional name of an attributes table containing the attributes (when not embedded in the vector tiles)
- `geometry_type_name` is the name of the geometry type for this layer as per [the same column in gpkg_geometry_columns](#) [<http://www.geopackage.org/spec121/#r25>]

`gpkgext_vt_fields`

The `gpkgext_vt_fields` table describes the fields (attributes) for a tiled feature data layer. The columns in this table are:

- `id` is a primary key
- `layer_id` is a foreign key to `id` in `gpkgext_vt_layers`
- `name` is the field name
- `type` is either "String", "Number", or "Boolean"

NOTE This table is not to be used for layers with a non-null `attributes_table_name`.

A.2. Mapbox Vector Tiles Extension

Extension Title

Mapbox Vector Tiles

Introduction

The GeoPackage Mapbox Vector Tiles extension defines the rules and requirements for encoding vector tiles in a GeoPackage data store as Mapbox Vector Tiles. This extension is based on the [Mapbox Vector Tiles \(MVT\) specification](#) [<https://www.mapbox.com/vector-tiles/specification/>] [version 2.1](#) [<https://github.com/mapbox/vector-tile-spec/tree/master/2.1>]. Note that this format uses [Google Protocol Buffers](#) [<https://github.com/google/protobuf>] as the content encoding for each tile.

This extension, like all GeoPackage extensions, is intended to be transparent and to not interfere with GeoPackage-compliant software packages that do not support the extension.

Extension Author

Image Matters LLC, in collaboration with the participants of the OGC Vector Tiles Pilot and OGC Testbed-15.

Extension Name or Template

`im_vector_tiles_mapbox` (If this extension is adopted by the OGC, then `gpkg_mapbox_vector_tiles` will be named as an alias.)

Extension Type

This extension defines an encoding for the [Vector Tiles Extension](#).

Applicability

This extension defines a specific encoding for Vector Tiles in a GeoPackage.

Scope

read-write

Specification

If this extension is in use, then all of the [Vector Tiles Extension](#) applies.

User Data Tables

Like other tile-based content, the physical data for a tile set is stored in a [user-defined tiles table](#) [<http://www.geopackage.org/guidance/getting-started.html#user-data-tables>]. The `tile_data` is a Google Protocol Buffer as [defined by MVT](#) [https://github.com/mapbox/vector-tile-spec/blob/master/2.1/vector_tile.proto].

`gpkg_extensions`

To use this extension, add a row to this table for each tile pyramid user data table.

Table 15. `gpkg_extensions` Table Rows

table_name	column_name	extension_name	definition	scope
<i>tile pyramid user data table name</i>	<code>tile_data</code>	<code>im_vector_tiles_mapbox</code>	<i>a reference to this file</i>	<i>read-write</i>

NOTE

The values in the `definition` column SHOULD refer in some human-readable way to this extension specification. If the extension is adopted by the OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

`gpkgevt_layers`

The Mapbox Vector Tiles specification does not distinguish between geometry types with a multiplicity of one (point, linestring, and polygon) and those with a multiplicity greater than one (multipoint, multilinestring, and multipolygon). In order to maintain consistency with `gpkg_geometry_columns`, the allowed values for the `geometry_type_name` column are as follows:

- GEOMETRY (corresponds to UNKNOWN in MVT, which is experimental and optional)
- MULTIPOINT (corresponds to POINT in MVT)
- MULTILINESTRING (corresponds to LINESTRING in MVT)
- MULTIPOLYGON (corresponds to POLYGON in MVT)

A.3. GeoJSON Vector Tiles Extension

Extension Title

GeoJSON Vector Tiles

Introduction

The GeoPackage Vector Tiles extension defines the rules and requirements for encoding vector tiles in a GeoPackage data store using [The GeoJSON Format](https://tools.ietf.org/html/rfc7946) [https://tools.ietf.org/html/rfc7946].

This extension, like all GeoPackage extensions, is intended to be transparent and to not interfere with GeoPackage-compliant software packages that do not support the extension.

Extension Author

Image Matters LLC, in collaboration with the participants of the OGC Vector Tiles Pilot and OGC Testbed-15.

Extension Name or Template

`im_vector_tiles_geojson` (If this extension is adopted by the OGC, then `gpkg_geojson_vector_tiles` will be named as an alias.)

Extension Type

This extension defines an encoding for the [Vector Tiles Extension](#).

Applicability

This extension defines a specific encoding for GeoJSON Vector Tiles in a GeoPackage.

Scope

read-write

Specification

If this extension is in use, then all of the [Vector Tiles Extension](#) applies.

User Data Tables

Like other tile-based content, the physical data for a tile set is stored in a [user-defined tiles table](http://www.geopackage.org/guidance/getting-started.html#user-data-tables) [http://www.geopackage.org/guidance/getting-started.html#user-data-tables]. The `tile_data` is a [GeoJSON Feature Collection](https://tools.ietf.org/html/rfc7946#section-3.3) [https://tools.ietf.org/html/rfc7946#section-3.3].

The GeoJSON Format does not restrict feature collections from having heterogeneous feature types (layers), nor does it define a standard way to distinguish the various feature types in a feature collection. This creates additional complexity for developers that must be accounted for. If more

than one layer is present in the feature collection, then a conventional feature ID must be used. The ID is a composite value containing the following:

- the feature type name
- the delimiter .
- a numeric feature ID for that feature

gpkg_extensions

To use this extension, add a row to this table for each tile pyramid user data table.

Table 16. *gpkg_extensions* Table Rows

table_name	column_name	extension_name	definition	scope
<i>tile pyramid user data table name</i>	<i>tile_data</i>	<i>im_vector_tiles_geojson</i>	<i>a reference to this file</i>	<i>read-write</i>

NOTE

The values in the **definition** column SHOULD refer in some human-readable way to this extension specification. If the extension is adopted by the OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

gpkgext_vt_layers

The GeoJSON Format does not restrict feature collections from having features with heterogeneous geometry types, nor does it define a standard way to declare the various geometry types in use. This creates additional complexity for developers that must be accounted for. It is reasonable for a layer to have features with at least one geometry type, for example one with a multiplicity of one (Point, Linestring, or Polygon) and one with a multiplicity greater than one (MultiPoint, MultiLinestring, or MultiPolygon). Therefore, GeoJSON-based vector tiles layers should have a **geometry_type_name** of "GEOMETRY" unless it can be determined that all of the geometries of that layer have the same geometry type. Then the corresponding **core geometry type name** [http://www.geopackage.org/spec121/#geometry_types] is to be used.

In addition, there must be a row in this table for each layer in the tile set. If there is more than one layer, the corresponding features are identified through their composite ID (see above).

A.4. GeoPackage Portrayal Extension

Extension Title

Portrayal

Introduction

This extension provides a mechanism for styles and symbols needed to implement portrayal in a GeoPackage.

Extension Author

Image Matters LLC, in collaboration with the participants of OGC Testbed-15 and the OGC Vector Tiles Pilot 2.

Extension Name or Template

`im_portrayal` (will become `gpkg_portrayal` if adopted by the OGC)

Extension Type

New requirement dependent on [GeoPackage Core \(Clause 1\)](http://www.geopackage.org/spec/#core) [http://www.geopackage.org/spec/#core].

Applicability

This extension allows for styles and symbols to be stored in a GeoPackage. How those styles are used is outside of the scope of this specification.

Scope

read-write

Specification

`gpkg_extensions`

To use this extension, add the following rows to this table.

Table 17. `gpkg_extensions` Table Rows

table_name	column_name	extension_name	definition	scope
<code>gpkgext_styles</code>	null	<code>im_portrayal</code>	<i>a reference to this file</i>	<code>read-write</code>
<code>gpkgext_symbols</code>	null	<code>im_portrayal</code>	<i>a reference to this file</i>	<code>read-write</code>
<code>gpkgext_stylesheets</code>	null	<code>im_portrayal</code>	<i>a reference to this file</i>	<code>read-write</code>
<code>gpkgext_symbol_content</code>	null	<code>im_portrayal</code>	<i>a reference to this file</i>	<code>read-write</code>
<code>gpkgext_symbol_images</code>	null	<code>im_portrayal</code>	<i>a reference to this file</i>	<code>read-write</code>

NOTE

The values in the **definition** column SHOULD refer in some human-readable way to this extension specification. If the extension is adopted by the OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

New Table Definitions

Following are definitions of the tables for this extension. As with other GeoPackage tables, this

extension takes no position on how either of these tables are to be used by a client.

gpkgext_styles

This table contains styles. The columns of this table are:

- `id` is a primary key
- `style` is text naming a specific style
- `description` is an optional text description
- `uri` is a resolvable URI

NOTE

Where possible, URIs should resolve to a network-accessible resource. When that is not possible, they should be unique. If no existing URI scheme is available, a URI can take the form of: `gpkgstyle::::[org>::[style]`

gpkgext_stylesheets

This table contains stylesheets. The columns of this table are:

- `id` is a primary key
- `style_id` is a foreign key to `gpkgext_styles`
- `format` is the format of the stylesheet (e.g., `mbstyle` or `sld`)
- `stylesheet` is the actual stylesheet BLOB (since some stylesheets are binary)

gpkgext_symbols

This table contains symbols. The columns of this table are:

- `id` is a primary key
- `symbol` is text naming a specific symbol
- `description` is an optional text description
- `uri` is a resolvable URI

NOTE

Where possible, URIs should resolve to a network-accessible resource. When that is not possible, they should be unique. If no existing URI scheme is available, a URI can take the form of: `gpkgsym::::[org>::[style>::[symbol]`

gpkgext_symbol_images

This table contains images representing symbols, with optional support for sprites. The columns of this table are:

- `id` is a primary key
- `symbol_id` is a foreign key to `gpkgext_symbols`
- `content_id` is a foreign key to `gpkgext_symbol_content`
- `width`, `height` are optional parameters that are required for sprites or for when there are

multiple versions of the same image with different sizes

- `offset_x`, `offset_y`, `pixel_ratio` are optional parameters for sprite information (NULL if the entire symbol is used)

`gpkgext_symbol_content`

This table contains the content (data) for symbols. The columns of this table are:

- `id` is a primary key
- `format` is the media type (formerly MIME type, e.g., `image/svg+xml` or `image/png`) of the symbol
- `content` is the actual symbol BLOB
- `uri` is a resolvable name to uniquely reference a specific content entry, e.g., for use in Mapbox GL styles "sprite" property to reference a particular sprite sheet

A.5. GeoPackage Semantic Annotation Extension

Extension Title

Semantic Annotations Extension

Introduction

A semantic annotation is a semantically grounded term that can be applied to another concept. Use of this extension enables semantic annotations to be applied to any business object in the current GeoPackage.

Extension Author

Image Matters LLC, in collaboration with the participants of OGC Testbed-15.

Extension Name or Template

`im_semantic_annotations` (will become `gpkg_semantic_annotations` if adopted by the OGC)

Extension Type

New requirement optionally dependent on the [GeoPackage Schema Extension](http://www.geopackage.org/spec121/#extension_schema) [http://www.geopackage.org/spec121/#extension_schema].

Applicability

This extension can be applied to any GeoPackage business object (layers, features, tiles, styles, etc.).

Scope

read-write

Specification

gpkg_extensions

To use this extension, add the following rows to this table in addition to the rows required for the Schema Extension (if used).

Table 18. *gpkg_extensions* table row

table_name	column_name	extension_name	definition	scope
gpkgext_semantic_annota tions	null	im_semantic_annota tions	a reference to this file	read-write
gpkgext_sa_referen ce	null	im_semantic_annota tions	a reference to this file	read-write

NOTE

The values in the **definition** column SHOULD refer in some human-readable way to this extension specification. If the extension is adopted by OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

New Table Definitions

Following are definitions of the tables for this extension. As with other GeoPackage tables, this extension takes no position on how either of these tables are to be used by a client.

gpkgext_semantic_annotations

When this extension is in use, add a table with this name and the following columns:

- **id** is a primary key
- **type** is a semantically grounded type (category) for the annotation
- **title** is a human-readable title for the annotation
- **description** is an optional human-readable text description for the annotation
- **uri** is the resolvable URI for the semantic concept

gpkgext_sa_reference

When this extension is in use, add a table with this name and the following columns:

- **table_name** is the name of the table containing the business object
- **key_column_name** is the name of the integer column in the specified table that acts as a key; if no such column exists, **rowid** can be used
- **key_value** is the value of the key column that uniquely identifies the row
- **sa_id** is a foreign key to **gpkgext_semantic_annotations**

Using Semantic Annotations

To use semantic annotations, do the following:

1. Add rows to `gpkgext_semantic_annotations` for every annotation you want to use.
 - a. Optionally, use the Schema Extension to establish an enumeration for the types and further describe those types. See <http://www.geopackage.org/guidance/extensions/schema.html> for more details.
2. Add a row to `gpkgext_sa_reference` for every row of every table requiring the annotation. There can be a many-to-many mapping between business object rows and semantic annotations.

A.6. Vector Tiles Attributes Extension

This extension defines a relationship between features of a vector tiles layer and vector tiles containing those features. When this extension is used, it is possible to perform a relational query and isolate only the vector tiles containing relevant features. In some circumstances this has the potential to greatly improve application performance.

Background

The [GeoPackage Related Tables Extension](http://docs.openeospatial.org/is/18-000/18-000.html) [http://docs.openeospatial.org/is/18-000/18-000.html] (RTE) defines the rules and requirements for creating relationships in a GeoPackage data store between geospatial data tables and other tables that contain or reference related content such as attributes or media. As an example, this can be used to establish a many-to-many relationship between features (e.g., points, lines, or areas) and multimedia files. By definition, the "left" side of the relationship is the "base" data and the "right" side of the relationship is the "related" data. The mapping table links related rows in those tables of those types by reference to their primary keys.

When relating vector tiles with the attributes of the features in those tiles, the base data is the vector tiles and the related data is the attributes as illustrated by [Table Diagram](#). The "GeoPackage Extension for Related Tables" allows a GeoPackage to contain additional data that is related to geospatial (e.g., features) or attributes data. When relating tiled feature data with attributes, the tiled feature data is the "base" data and the attributes are the "related" data.

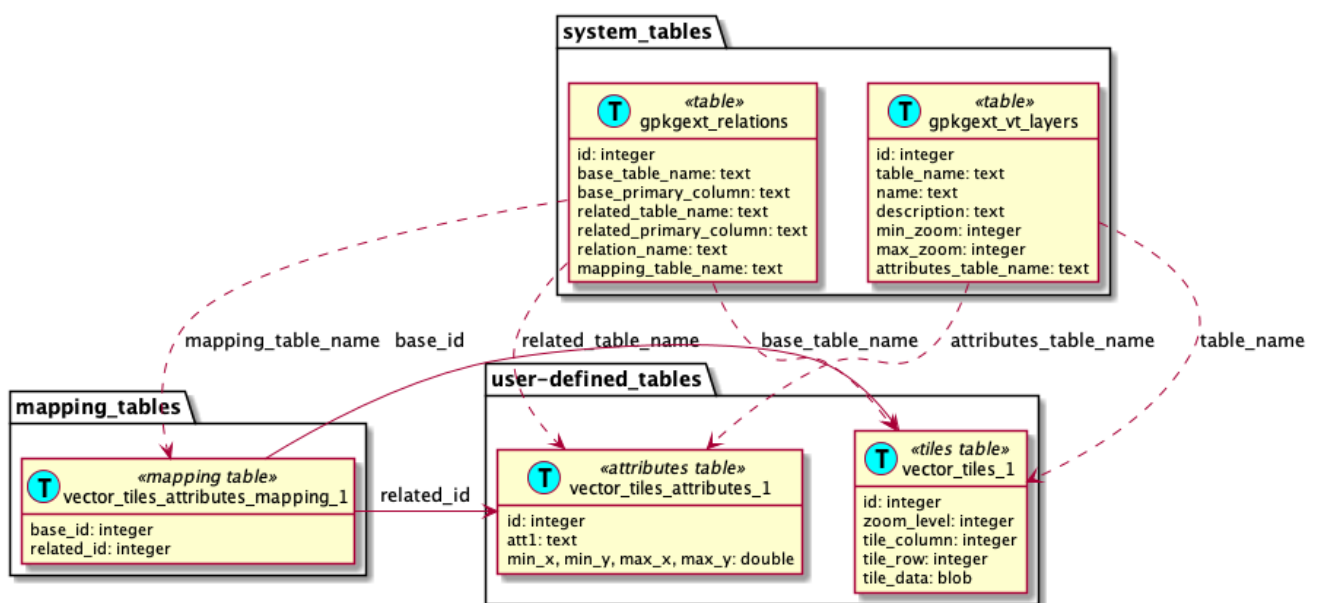


Figure 56. Table Diagram

NOTE

The RTE does not define a requirements class to map tiles tables with attributes tables. This section defines a requirements class that will fill this need. A note has been added to the proposed RTE standard to indicate that additional requirements classes are possible. For information on using the Related Tables Extension, see the [Getting Started Guide](https://github.com/opengeospatial/geopackage-related-tables/wiki/Getting-Started) [https://github.com/opengeospatial/geopackage-related-tables/wiki/Getting-Started].

gpkg_extensions

To use this extension, add the following rows to this table as described in [gpkg_extensions](http://www.geopackage.org/guidance/getting-started.html#gpkg_extensions) [http://www.geopackage.org/guidance/getting-started.html#gpkg_extensions].

Table 19. *gpkg_extensions* Table Rows

table_name	column_name	extension_name	definition	scope
<code>gpkgext_relations</code>	null	<code>related_tables</code>	https://github.com/opengeospatial/geopackage-related-tables	<code>read-write</code>
<i>name of actual User-defined Mapping Table</i>	null	<code>related_tables</code>	https://github.com/opengeospatial/geopackage-related-tables	<code>read-write</code>

NOTE

The values in the `definition` column SHOULD refer in some human-readable way to this extension specification. If the extension is adopted by the OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

gpkgext_relations

This table describes extended relationships. The table requires the following columns:

Table 20. *gpkgext_relations* Table Rows

Column	Description
<code>id</code>	primary key
<code>base_table_name</code>	Name of the vector tiles table
<code>base_primary_column</code>	<code>id</code> (all user-defined tiles tables have this column)
<code>related_table_name</code>	Name of the user-defined attributes table
<code>related_primary_column</code>	Name of the primary key column in <code>related_table_name</code>
<code>relation_name</code>	<code>vector_tiles_attributes</code>
<code>mapping_table_name</code>	Name of a User-defined Mapping Table

Add a row to this table for each vector tiles layer with attributes in an attributes table.

gpkgext_vt_layers

Set the `attributes_table_name` column to the appropriate table for each vector tiles layer with attributes in an attributes table.

User-defined Mapping Table

A [user-defined mapping table](http://www.geopackage.org/guidance/extensions/related_tables.html#user-defined-mapping-table) [http://www.geopackage.org/guidance/extensions/related_tables.html#user-defined-mapping-table] describes the many-to-many relationships between base data (tiles) and related data (features). A user-defined mapping table requires at least the following columns:

Table 21. *gpkgext_relations* Table Rows

Column	Description
<code>base_id</code>	tile ID (the primary key value of the base data table)
<code>related_id</code>	feature ID (the primary key value of the related data table)

TIP

By adding row to this table for each feature/tile instance, it is possible for a client to query for features by their attributes. However, full population of this table is not mandatory. It may be prudent to omit some entries for space reasons.

A.7. GeoPackage Tile Matrix Set Extension

Extension Title

Tile Matrix Set Extension

Introduction

This extension provides alignment with [OGC Two Dimensional Tile Matrix Set](http://docs.opengeospatial.org/is/17-083r2/17-083r2.html) [http://docs.opengeospatial.org/is/17-083r2/17-083r2.html] (OGC 17-083r2).

Extension Author

Ecere, in collaboration with the Image Matters and other participants of OGC Vector Tiles Pilot Phase 2.

Extension Name or Template

`ecere_tms` (will become `gpkg_tms` if adopted by the OGC)

Extension Type

New requirement dependent on the [GeoPackage Tiles Option](http://www.geopackage.org/spec121/#tiles) [http://www.geopackage.org/spec121/#tiles].

Applicability

This extension allows description and sharing of tile matrix sets, including those with variable extents and row widths.

Scope

read-write

Specification

gpkg_extensions

To use this extension, add the following rows to this table.

Table 22. *gpkg_extensions* table row

table_name	column_name	extension_name	definition	scope
gpkgext_tile_matrix_set	null	ecere_tms	a reference to this file	read-write
gpkgext_tile_matrix	null	ecere_tms	a reference to this file	read-write
gpkgext_tile_matrix_tables	null	ecere_tms	a reference to this file	read-write
gpkgext_tile_matrix_variable_widths	null	ecere_tms	a reference to this file	read-write
gpkg_tile_matrix_set	null	ecere_tms	a reference to this file	write-only
gpkg_tile_matrix	null	ecere_tms	a reference to this file	write-only

NOTE

The values in the **definition** column SHOULD refer in some human-readable way to this extension specification. If the extension is adopted by OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

Table Definitions

Following are definitions of the tables for this extension. As with other GeoPackage tables, this extension takes no position on how either of these tables are to be used by a client.

gpkgext_tile_matrix_set

When this extension is in use, add a table with this name and the following columns:

- **id** is a primary key
- **tms** is a human-readable title for the tile matrix set
- **description** is an optional human-readable description for the tile matrix set
- **uri** is a machine-readable URI for the tile matrix set

- `srs_id` is the SRS for the tile matrix set and is a foreign key to `gpkg_spatial_ref_sys.srs_id`
- `min_x`, `min_y`, `max_x`, and `max_y` represent the bounding box of the tile matrix set

`gpkgext_tile_matrix`

When this extension is in use, add a table with this name and the following columns:

- `id` is a primary key
- `tms_id` is a foreign key to `gpkgext_tile_matrix_set.id`
- `zoom_level` indicates the zoom levels present in the file
- `matrix_width` and `matrix_height` describe the size of the tile matrix in tiles
- `tile_width` and `tile_height` describe the size of each tile in pixels
- `pixel_x_size` and `pixel_y_size` describe the size of each pixel
- `left` and `top` are the top left corner of the tile matrix, which can potentially be different than the tile matrix set
- `scale_denominator` is an optional double representing scale denominator with respect to a "standardized rendering pixel size" of 0.28 mm × 0.28 mm

`gpkgext_tile_matrix_tables`

When this extension is in use, add a table with this name and the following columns:

- `table_name` is a primary key and also a foreign key to `gpkg_contents.table_name`
- `tms_id` is a foreign key to `gpkgext_tile_matrix_set.id`
- `min_level` and `max_level` are the minimum and maximum zoom levels

`gpkgext_tile_matrix_variable_widths`

When this extension is in use, add a table with this name and the following columns:

- `id` is a primary key
- `tm_id` is a foreign key to `gpkgext_tile_matrix.id`
- `min_row` and `max_row` describe the minimum and maximum tile row index where the given coalescence coefficient applies
- `coalesce` is an integer coalescence coefficient

View Definitions

To maintain compatibility with clients that are not aware of this extension, the following views are created.

`gpkg_tile_matrix_set`

When using this extension, replace this table with a view to maintain compatibility with GeoPackage clients that are not aware of this extension.

1. Copy all rows of `gpkg_tile_matrix_set` into `gpkgext_tile_matrix_set`. Where possible, use an

existing or well-known tile matrix set. If the tile matrix set URI cannot be determined, use the `table_name` as the URI.

2. For each row copied, add a row to `gpkgext_tile_matrix_tables`.
3. `DROP pkg_tile_matrix_set`.
4. `CREATE VIEW pkg_tile_matrix_set AS SELECT a.table_name, b.srs_id, b.min_x, b.min_y, b.max_x, b.max_y FROM pkgext_tile_matrix_tables a, pkgext_tile_matrix_set b WHERE a.tms_id = b.id`.

WARNING

When this view is in place, clients that are not aware of this extension will not be able to add new tile pyramids to the GeoPackage. The development of triggers to turn this view into an updateable view (TBD) would eliminate this restriction.

`gpkg_tile_matrix`

When using this extension, replace this table with a view to maintain compatibility with GeoPackage clients that are not aware of this extension.

1. Copy all rows of `gpkg_tile_matrix` into `gpkgext_tile_matrix`. Where possible, use an existing or well-known tile matrix set. If the tile matrix set URI cannot be determined, use the `table_name` as the URI.
2. `DROP pkg_tile_matrix`.
3. `CREATE VIEW pkg_tile_matrix AS SELECT a.table_name, b.zoom_level, b.matrix_width, b.matrix_height, b.tile_width, b.tile_height, b.pixel_x_size, b.pixel_y_size FROM pkgext_tile_matrix_tables a, pkgext_tile_matrix b WHERE a.tms_id = b.tms_id AND b.zoom_level >= a.min_level AND b.zoom_level <= a.max_level AND b.id NOT IN (SELECT DISTINCT tm_id FROM pkgext_tile_matrix_variable_widths);`

WARNING

When this view is in place, clients that are not aware of this extension will not be able to add new tile pyramids to the GeoPackage. The development of triggers to turn this view into an updateable view (TBD) would eliminate this restriction.

NOTE

This approach hides tables that have variable-width tile matrix sets from the compatibility view. An alternate approach (TBD) would create a shadow tile matrix set that excludes all variable-width tile matrix rows.

Appendix B: Additional Screenshots

B.1. GeoSolutions D101 Feature Server

An additional screenshot for the GeoSolutions feature server is presented below, in [Figure 56](#).



Figure 57. Collection description, including link to the data tiles

B.1.1. GeoSolutions D104 Client

Additional screenshots for the GeoSolutions client are presented below.

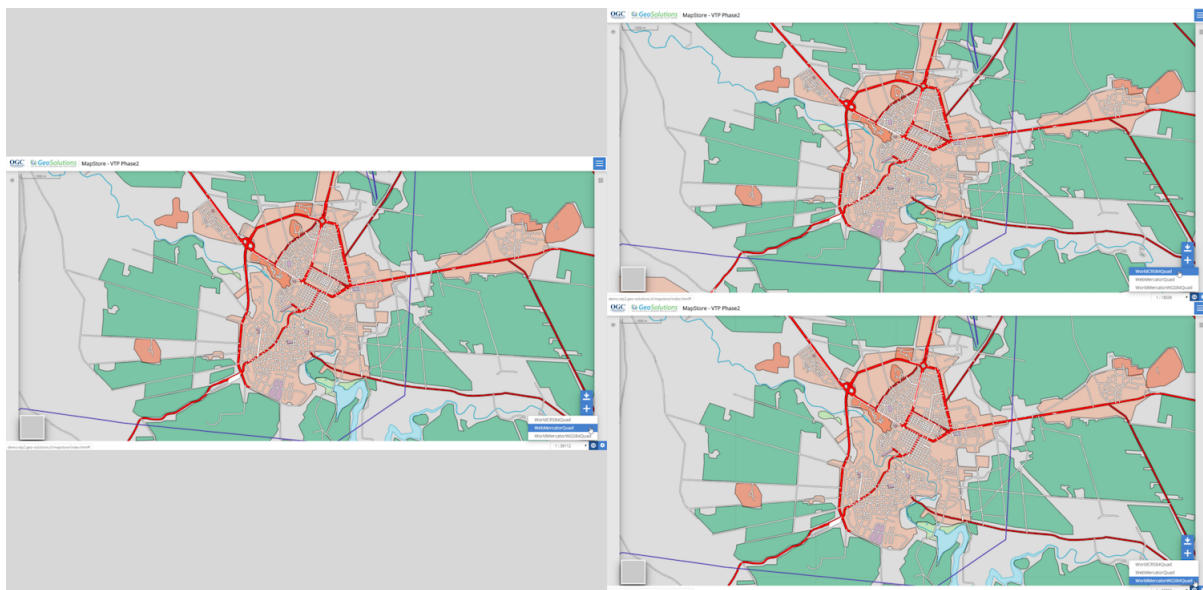


Figure 58. GeoSolutions MapStore Client uses different tile matrix set: WebMercatorQuad (left), WorldCRS84Quad (top right) and WorldMercatorWGS84Quad (bottom right)

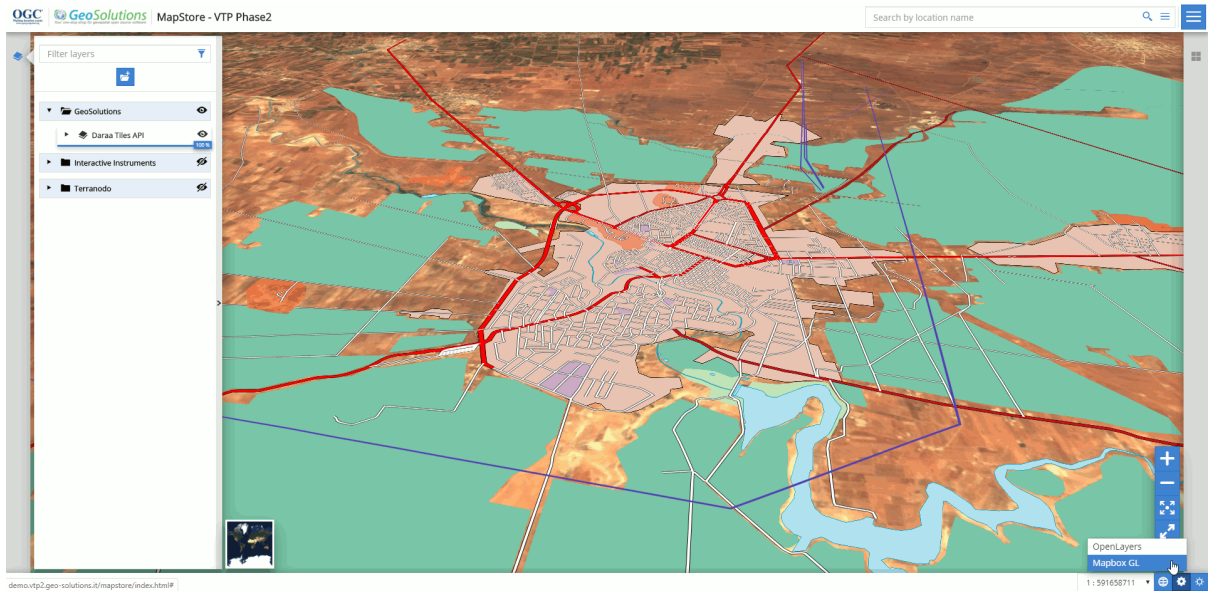


Figure 59. GeoSolutions MapStore Client with Mapbox GL-support

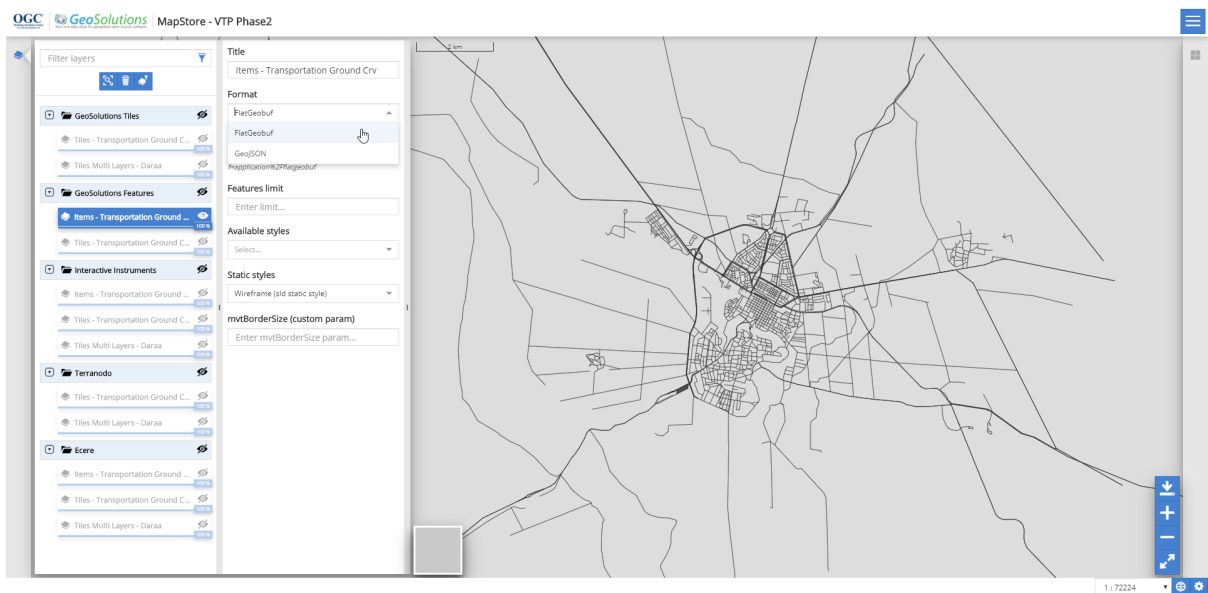


Figure 60. GeoSolutions MapStore Client shows feature items from GeoServer in flatgeobuf format

B.1.2. Ecere D103 Features, Tiles and Styles API

Additional screenshots of the Ecere Features, Tiles and Styles servers are below.

GNOSIS Map Server @ maps: x +
https://maps.ecere.com/geoapl/collections/NaturalEarth/raster/NE2_HR_LC_SR_W_DR

GNOSIS GNOSIS Map Server

NE2_HR_LC_SR_W_DR

(View [JSON](#) or [ECON](#) representation)

Type: raster
Scale: 1:4367830.1877243574709
Extent: { { lat: -90, lon: -180 }, { lat: 90, lon: 180 } }

[Tiles API for this collection](#)

[Styles for this collection](#)

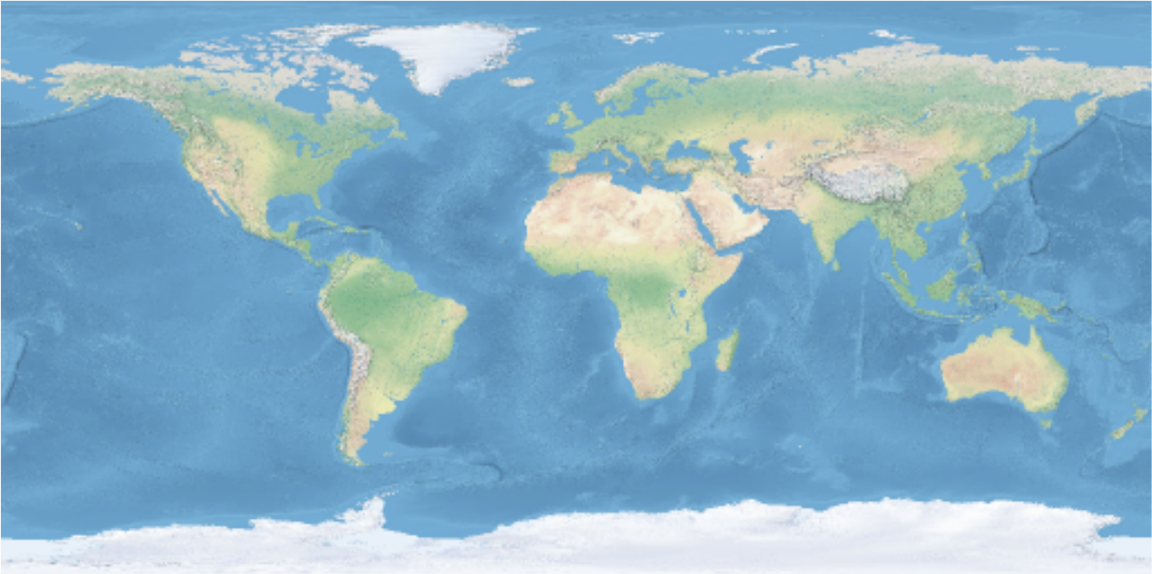


Figure 61. Natural Earth imagery data (available whole, tiled or clipped to BBOX)

GNOSIS Map Server @ maps.ecere.com - Chromium

GNOSIS Map Server

Daraa_DTED

(View [JSON](#) or [ECON](#) representation)

Type: coverage
Scale: 1:272989.3867327723419
Extent: { { lat: 31.8990833333333, lon: 34.8999166666667 }, { lat: 33.1000833333333, lon: 37.1009166666667 } }

[Tiles API for this collection](#)

[Styles for this collection](#)




Figure 62. Daraa DTED elevation data (gridded coverage, available whole, tiled or clipped to BBOX)

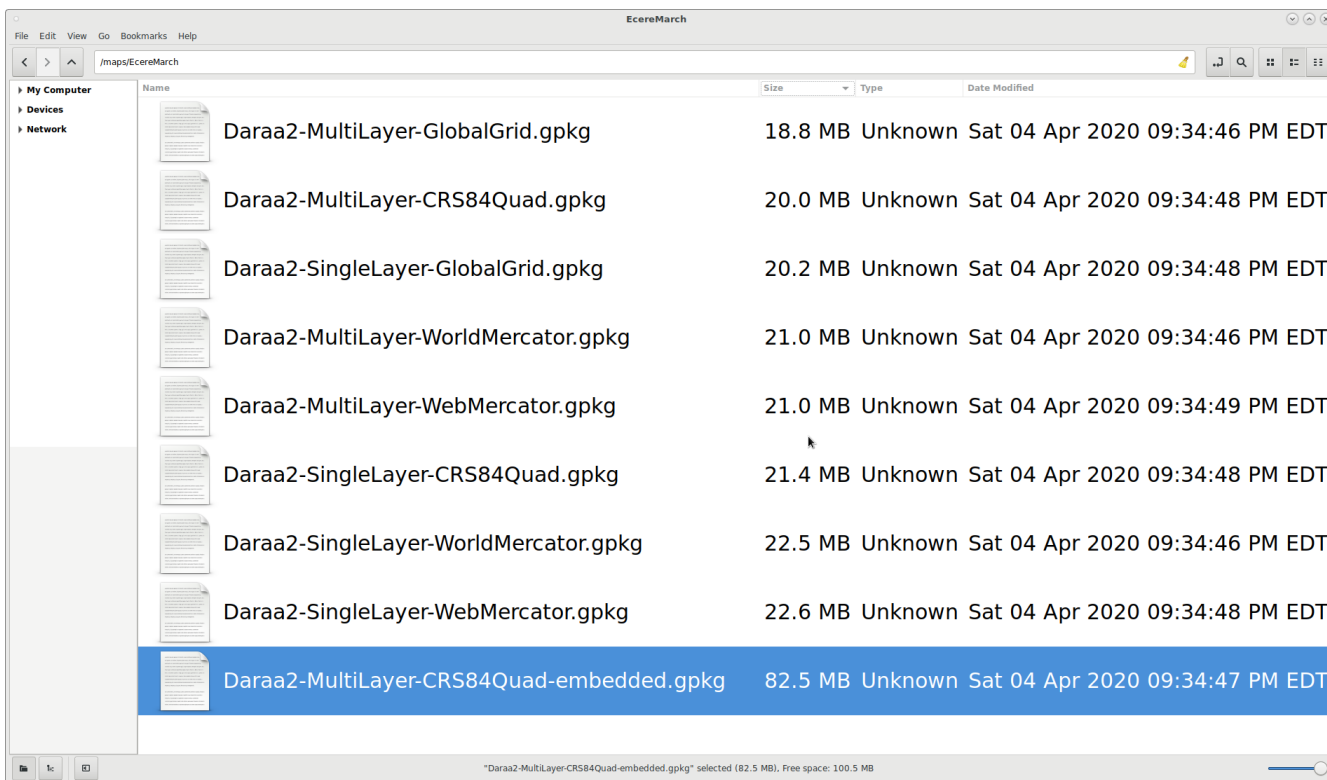


Figure 63. GeoPackage files generated including the OSM TDS vector data and elevation data (2 DTED Level 1 cells)

GNOSIS Map Server @ maps x +
<https://maps.ecere.com/geoapi/tileMatrixSets/CDBGlobalGrid>

GNOSIS GNOSIS Map Server

CDBGlobalGrid

(View [JSON](#) or [ECON](#) representation)

CRS: [EPSG:4326](#) (geographic)

Zoom Level	Scale	Rows Count	Columns (c=1)	Width (px)	Height (px)	Meters / pixel	Degrees / pixel	Top-Left Corner (lat, lon)	Delta Latitude	Delta Longitude (c=1)	Variable Width Rows Coalescing Coefficient (c)
-10	1:397,569,609.976	180	360	1	1	111,319.49079327356	1 °	90, -180	1 °	1 °	Row 0: 12 Rows 1..9: 6 Rows 10..14: 4 Rows 15..19: 3 Rows 20..39: 2 Rows 140..159: 2 Rows 160..164: 3 Rows 165..169: 4 Rows 170..178: 6 Row 179: 12
-9	1:198,784,804.988	180	360	2	2	55,659.74539663679	30'	90, -180	1 °	1 °	Row 0: 12 Rows 1..9: 6 Rows 10..14: 4 Rows 15..19: 3 Rows 20..39: 2 Rows 140..159: 2 Rows 160..164: 3 Rows 165..169: 4 Rows 170..178: 6 Row 179: 12
-8	1:99,392,402.494	180	360	4	4	27,829.87269831839	15'	90, -180	1 °	1 °	Row 0: 12 Rows 1..9: 6 Rows 10..14: 4 Rows 15..19: 3 Rows 20..39: 2 Rows 140..159: 2

Figure 64. CDB Global Grid tile matrix set (negative levels starting at 1x1 pixel)

0	1:388,251.572	180	360	1024	1024	108.71044022781	3.515625 "	90, -180	1 °	1 °	Row 0: 12
											Rows 1..9: 6
											Rows 10..14: 4
											Rows 15..19: 3
											Rows 20..39: 2
											Rows 140..159: 2
											Rows 160..164: 3
											Rows 165..169: 4
											Rows 170..178: 6
											Row 179: 12
1	1:194,125.786	360	720	1024	1024	54.3552201139	1.7578125 "	90, -180	30 '	30 '	Rows 0..1: 12
											Rows 2..19: 6
											Rows 20..29: 4
											Rows 30..39: 3
											Rows 40..79: 2
											Rows 280..319: 2
											Rows 320..329: 3
											Rows 330..339: 4
											Rows 340..357: 6
											Rows 358..359: 12
											Rows 0..3: 12
											Rows 4..39: 6

Figure 65. CDB Global Grid tile matrix set (positive levels — 1024x1024 pixels)

B.2. Ecere D103 Client

Additional screenshots for the Ecere client applications are presented below.

These screenshots include imagery from Google Maps (c) CNES / Airbus, Maxar Technology, TerraMetrics, NASA, U.S. Geological Survey, Landsat / Copernicus, global 3" elevation data from View Finder Panoramas by Jonathan de Ferranti, from Shuttle Radar Topography Mission (NASA, NGA, U.S. Geological Survey) and other sources, as well as VTP2 datasets, including profiles of OpenStreetMap data (c) OpenStreetMap Contributors.

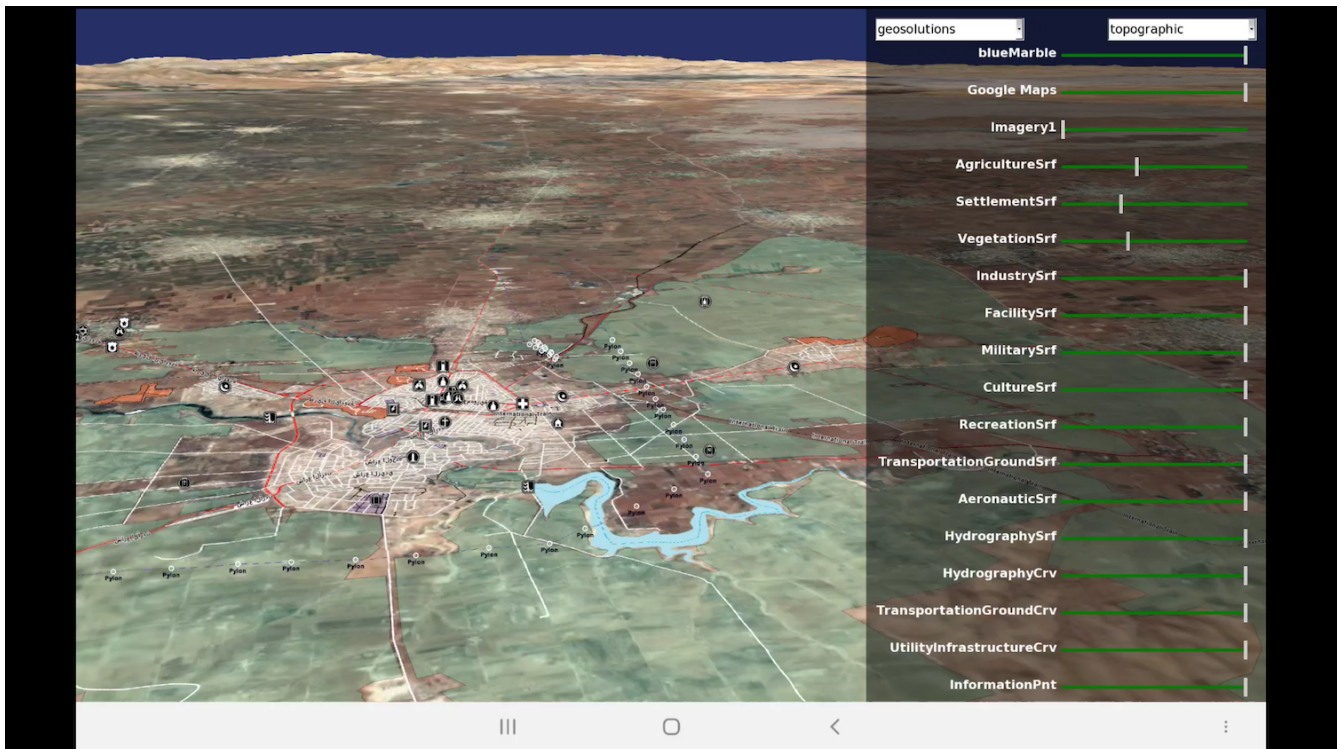


Figure 66. Ecere's mobile Android client accessing GeoSolutions Tiles API

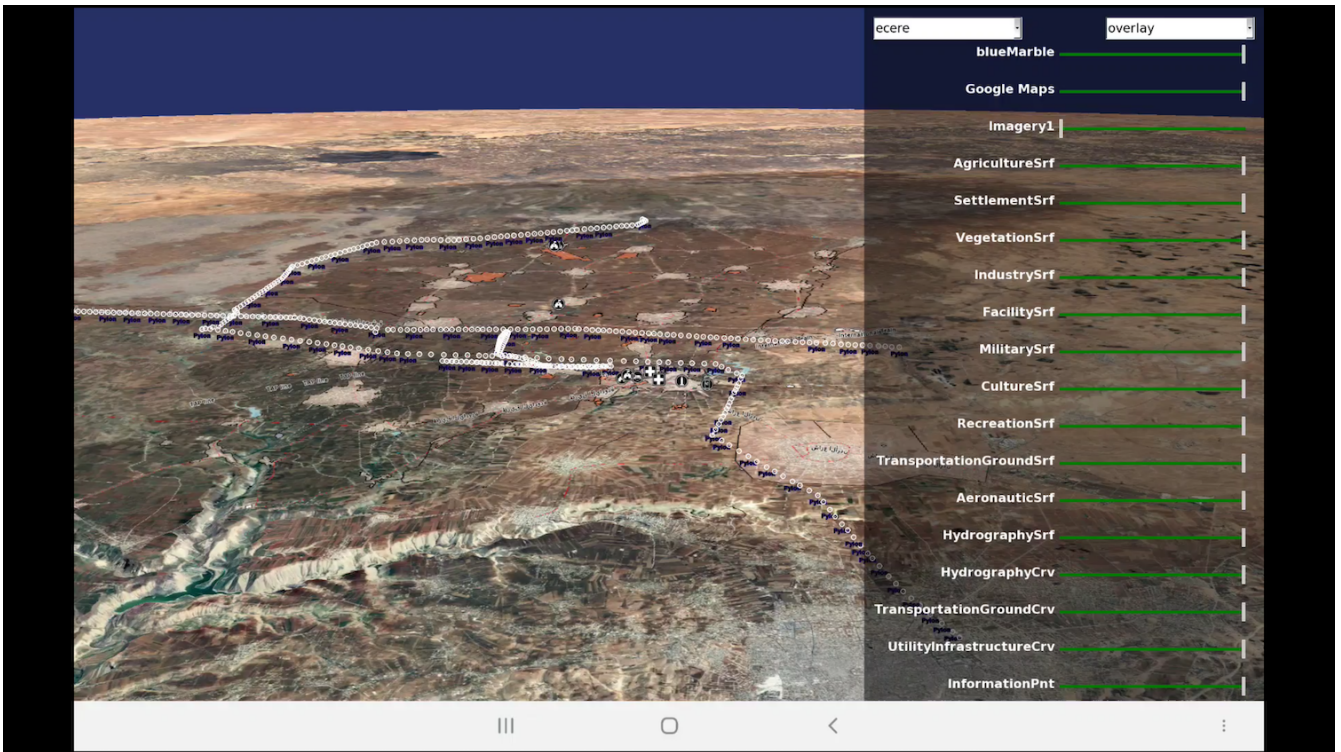


Figure 67. Ecere's mobile Android client accessing Ecere Tiles API

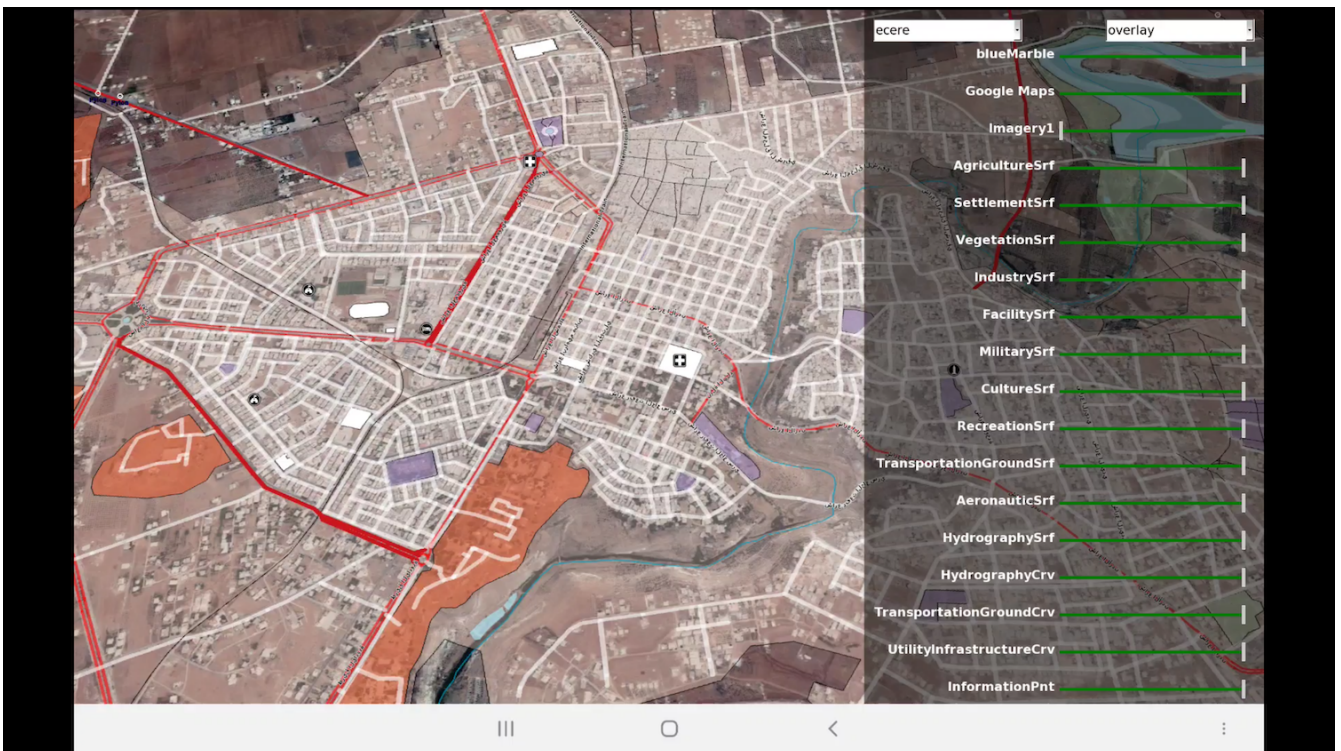


Figure 68. Ecere's mobile Android client accessing Ecere Tiles API (closer top-down view)

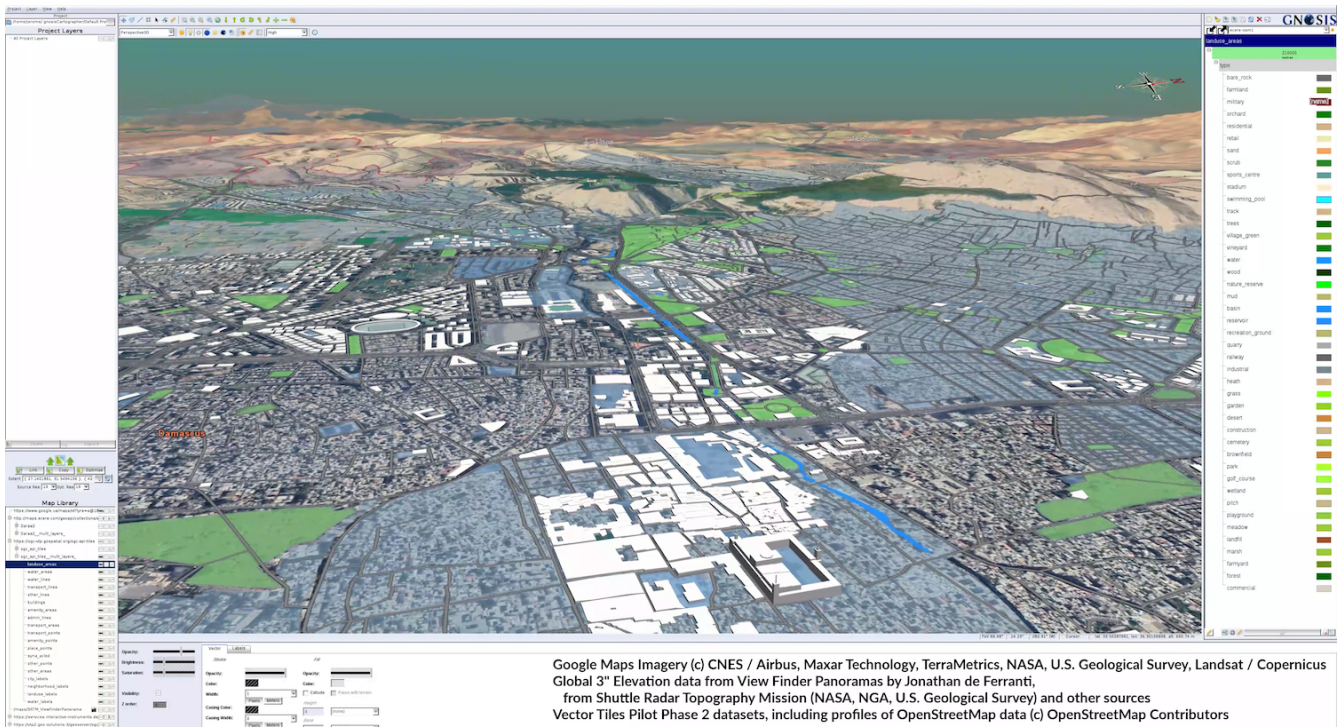


Figure 69. Damascus OpenStreetMap 3D Buildings from Terranodo Tiles API displayed in GNOGIS Cartographer



Figure 70. Singapore OpenStreetMap 3D Buildings from vector tiles stored in GNOGIS Data Store displayed in GNOGIS Cartographer

B.3. interactive instruments

The next three screenshots (Figure 70, Figure 71, and Figure 72) show the multi-layer vector tiles in an OpenLayers map using a simple wireframe style for all three tiling schemes:

Daraa

This is a test dataset for the Open Portrayal Framework thread in the OGC Testbed-15 as well as for the OGC Vector Tiles Pilot Phase 2. The data is OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.

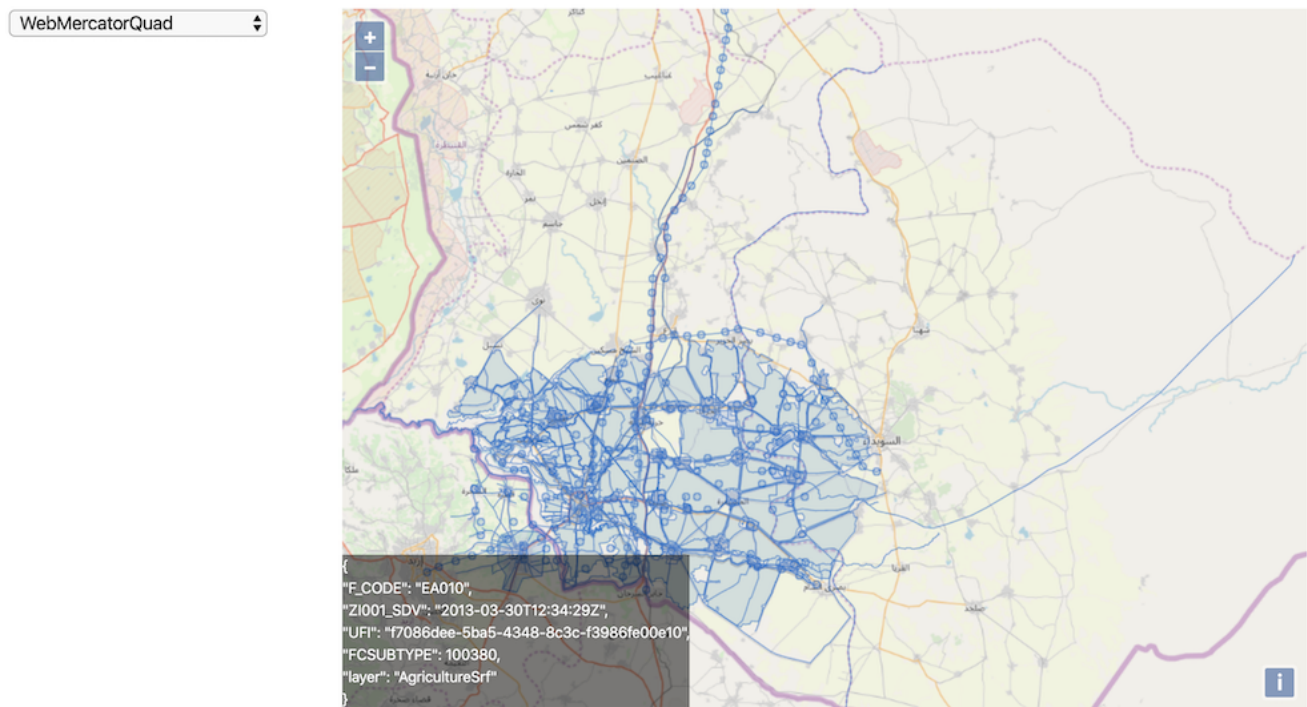


Figure 71. interactive instruments - Multi-layer vector tiles in the standard "WebMercatorQuad" tiling scheme

Daraa

This is a test dataset for the Open Portrayal Framework thread in the OGC Testbed-15 as well as for the OGC Vector Tiles Pilot Phase 2. The data is OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.

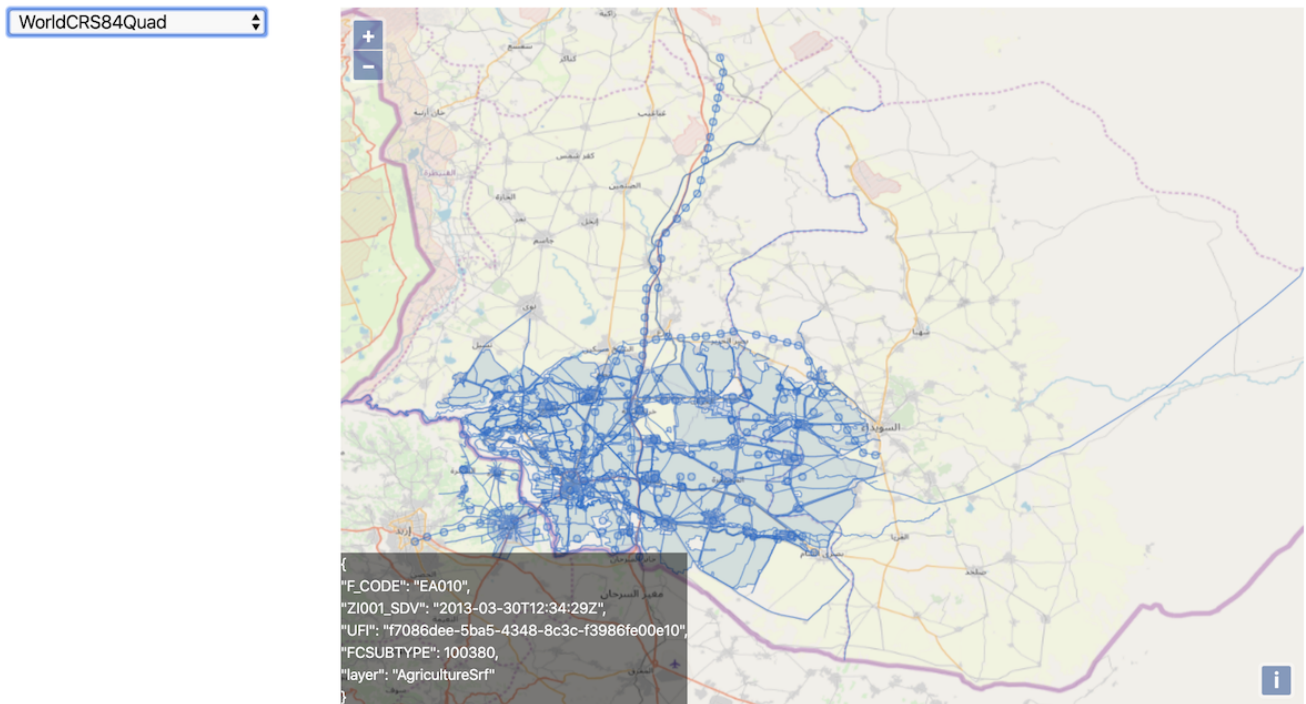


Figure 72. interactive instruments - Multi-layer vector tiles in the "WorldCRS84Quad" tiling scheme

Daraa

This is a test dataset for the Open Portrayal Framework thread in the OGC Testbed-15 as well as for the OGC Vector Tiles Pilot Phase 2. The data is OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.

WorldMercatorWGS84Quad ▾

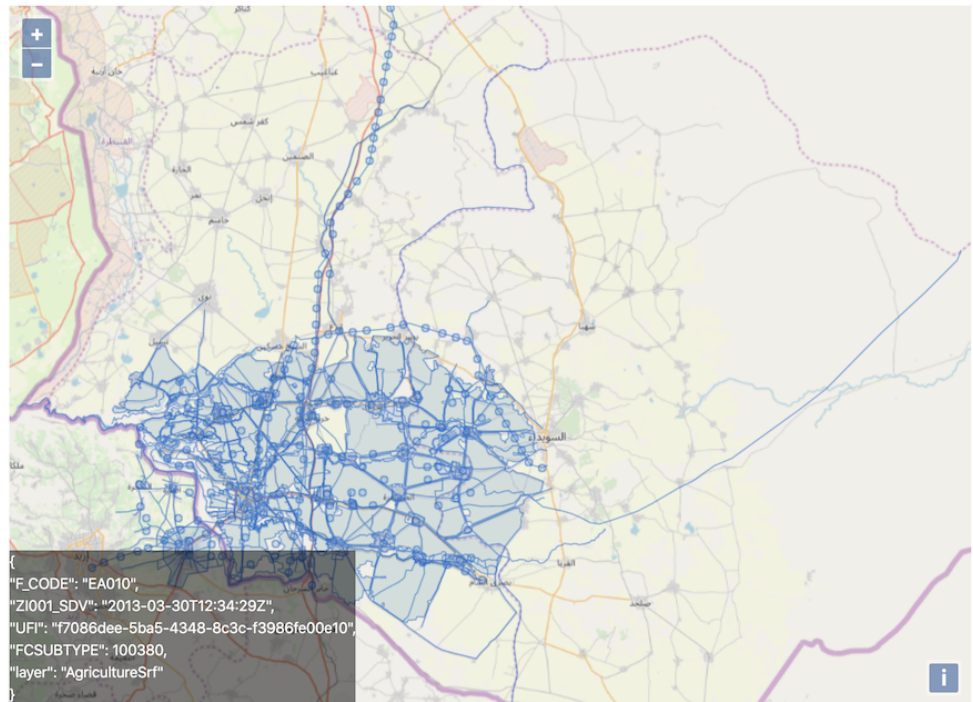


Figure 73. interactive instruments - Multi-layer vector tiles in the "WorldMercatorWGS84Quad" tiling scheme

Appendix C: Revision History

Table 23. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
2020-01-10	G. Hobona	.1	all	initial version
2020-03-25	G. Hobona	.2	all	Revision, based on Carl Reed's review
2020-04-09	G. Hobona	.3	all	Revision, based on Terry Idol's review

Appendix D: Bibliography

- [1] Simmons, S., Reed, C.: Technical Committee Policies and Procedures. OGC 05-020r24, Open Geospatial Consortium, <https://docs.opengeospatial.org/pol/05-020r24/05-020r24.html> (2016).
- [2] Meek, S.: OGC Vector Tiles Pilot: Summary Engineering Report. OGC 18-086, Open Geospatial Consortium, <https://docs.opengeospatial.org/per/18-086r1.html> (2019).
- [3] Open Geospatial Consortium: OGC API - Maps draft specification, <https://github.com/opengeospatial/OGC-API-Tiles/tree/master/core>.
- [4] Open Geospatial Consortium: OGC API - Tiles draft specification, <https://github.com/opengeospatial/OGC-API-Tiles/tree/master/core>.
- [5] Aime, A., Bovolo, S.: OGC Vector Tiles Pilot 2: Vector Tiles Filtering Language Engineering Report. OGC 19-084, Open Geospatial Consortium, <https://www.opengeospatial.org/docs/er> (2020).
- [6] NGA: National System for Geospatial Intelligence (NSG) Metadata Foundation (NMF) - Version 3.0. NGA.STND.0012_3.0, National Geospatial Intelligence Agency (2016).
- [7] Taleisnik, S.: OGC Vector Tiles Pilot 2: Tile Set Metadata Engineering Report. OGC 19-082, Open Geospatial Consortium, <https://www.opengeospatial.org/docs/er> (2020).
- [8] Yutzler, J.: Vector Tiles Pilot Extension Engineering Report. OGC 18-101, Open Geospatial Consortium, <http://docs.opengeospatial.org/per/18-101.html> (2019).
- [9] Portele, C.: OGC Testbed-15: Styles API Engineering Report. OGC 19-010r2, Open Geospatial Consortium, <http://docs.opengeospatial.org/per/19-010r2.html> (2019).
- [10] 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).
- [11] NGA: NSG vector tiles draft interoperability specification.
- [12] Klopfer, M.: OGC Testbed-12 General Feature Model Engineering Report. OGC 16-047r1, Open Geospatial Consortium, <http://docs.opengeospatial.org/per/16-047r1.html> (2016).