

Vector Tiles Pilot Extension Engineering Report

Table of Contents

1. Summary	4
1.1. Requirements & Research Motivation	4
1.2. Findings and Challenges encountered	5
1.3. Prior-After Comparison	6
1.4. Recommendations for Future Work	7
1.5. Document contributor contact points	8
1.6. Foreword	8
2. References	9
3. Terms and definitions	10
3.1. Abbreviated terms	11
4. Overview	12
5. Concept of Operations	13
5.1. Data Preparation	13
5.2. Web Services / Web APIs	13
5.3. GeoPackage Provisioning	13
5.4. Integrated Clients	13
6. Meeting the Challenge	15
6.1. Pilot Architecture	15
6.2. Conceptual Model	16
6.2.1. Conceptual Model Challenges	18
6.2.2. SLD Model Versus Mapbox Styles Model	20
6.3. Technology Integration Experiments	24
7. Implementation Approaches	26
7.1. Data Preparation	26
7.1.1. Producing Tiled Feature Data	26
7.1.2. Producing Stylesheets	27
7.2. Web Service Implementations	28
7.2.1. Web Feature Service	28
7.2.2. Web Map Tile Service (Mapping 1)	34
7.2.3. Web Map Tile Service (Mapping 2)	39
7.3. GeoPackage Provisioning	40
7.3.1. Producing GeoPackages	40
7.3.2. Supporting Attributes in GeoPackage	43
7.3.3. Supporting Style Sheets in GeoPackage	45
7.3.4. Deploying GeoPackages	47
7.4. Integrated Clients	47
7.4.1. Binding to a WMTS, WFS and/or GeoPackage	47
7.4.2. Requesting Tiled Feature Data from a WMTS	50

7.4.3. Requesting Tiled Feature Data from a WFS	62
7.4.4. Modifying Style Sheets	68
7.4.5. Displaying Tiled Feature Data	74
7.4.6. Querying Tiled Feature Data	97
8. Discussion	102
8.1. Differences between WMTS and Mapbox Layers	102
8.2. Differences Between SLD and Mapbox Styles	102
8.3. Using Offerings to Correlate Tiled Feature Data Layers to Style Sheets	102
8.3.1. Analysis	102
8.4. Using OWS Contexts to Describe Map Views	103
8.4.1. Analysis	103
Appendix A: GeoPackage Tiled Feature Data Extensions (Informative)	104
A.1. Tiled Feature Data Extension	104
A.1.1. Extension Title	104
A.1.2. Introduction	104
A.1.3. Extension Author	104
A.1.4. Extension Name or Template	104
A.1.5. Extension Type	104
A.1.6. Applicability	104
A.1.7. Scope	105
A.1.8. Specification	105
A.2. GeoPackage Mapbox Vector Tiles Extension	106
A.2.1. Extension Title	106
A.2.2. Introduction	106
A.2.3. Extension Author	106
A.2.4. Extension Name or Template	106
A.2.5. Extension Type	107
A.2.6. Applicability	107
A.2.7. Scope	107
A.2.8. Specification	107
A.3. GeoPackage GeoJSON Vector Tiles Extension	107
A.3.1. Extension Title	107
A.3.2. Introduction	107
A.3.3. Extension Author	108
A.3.4. Extension Name or Template	108
A.3.5. Extension Type	108
A.3.6. Applicability	108
A.3.7. Scope	108
A.3.8. Specification	108
A.4. GeoPackage Styles Extension	108
A.4.1. Extension Title	109

A.4.2. Introduction	109
A.4.3. Extension Author	109
A.4.4. Extension Name or Template	109
A.4.5. Extension Type	109
A.4.6. Applicability	109
A.4.7. Scope	109
A.4.8. Specification	109
A.5. GeoPackage OWS Context Extension	110
A.5.1. Extension Title	110
A.5.2. Introduction	110
A.5.3. Extension Author	110
A.5.4. Extension Name or Template	111
A.5.5. Extension Type	111
A.5.6. Applicability	111
A.5.7. Scope	111
A.5.8. Specification	111
Appendix B: GeoPackage Extensions Requirements (Normative)	114
B.1. GeoPackage Tiled Feature Data Extension	114
B.1.1. gpkg_contents	114
B.1.2. gpkg_extensions	114
B.1.3. gpkgext_tfd_layers	115
B.1.4. gpkgext_tfd_fields	116
B.2. GeoPackage Mapbox Vector Tiles Extension	117
B.2.1. gpkg_extensions	117
B.2.2. User Defined Tiles Tables	118
B.3. GeoPackage GeoJSON Vector Tiles Extension	118
B.3.1. gpkg_extensions	118
B.3.2. User Defined Tiles Tables	118
B.4. GeoPackage Styles Extension	119
B.4.1. gpkg_extensions	119
B.4.2. gpkgext_stylesheets	119
B.4.3. gpkgext_stylesheets	120
B.5. GeoPackage OWS Context Extension	121
B.5.1. gpkg_extensions	121
B.5.2. gpkgext_contexts	121
B.5.3. gpkgext_context_resources	124
B.5.4. gpkgext_context_offerings	127
Appendix C: The OpenAPI Styles API	129
C.1. Overview	129
C.2. API definition	130
C.2.1. Encoding	130

C.2.2. Landing page	130
C.2.3. Style set	131
C.2.4. Style	134
C.2.5. Collection metadata	135
C.2.6. Conformance declaration	137
C.3. Open issues and future work	137
Appendix D: WMTS 1.0 Styles API Profile Specification	140
D.1. Introduction	140
D.2. Declaration Of Profile	140
D.3. Style URL Templates (RESTful)	140
D.3.1. GET	140
D.3.2. PUT	141
D.3.3. DELETE	141
D.3.4. OPTIONS	141
D.4. GetStyle Operation (KVP)	142
D.5. Security Considerations	142
D.6. Limitations And Future Work	143
Appendix E: Revision History	144
Appendix F: Bibliography	145

Publication Date: 2019-04-30

Approval Date: 2019-04-14

Submission Date: 2019-04-02

Reference number of this document: OGC 18-101

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

Category: OGC Public Engineering Report

Editor: Jeff Yutzler

Title: Vector Tiles Pilot Extension Engineering Report

OGC Public Engineering Report

COPYRIGHT

Copyright © 2019 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 NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

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

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

indicate compliance with any LICENSOR standards or specifications.

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

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

Chapter 1. Summary

The purpose of the OGC Vector Tiles Pilot Extension (VTPEExt) was to address portrayal and style encoding concerns that were discovered in the initial phase of the Vector Tiles Pilot (VTP). During the VTPEExt, participants selected a common baseline style used by all participants and in some cases created additional style offerings. The work conducted during the VTPEExt has adhered to the established findings from the initial VTP documented in the VTP Summary Engineering Report (ER) [1].

This document describes the following:

- the research and evaluation to determine approach(es) to apply styling to Mapbox and GeoJSON Tiled Feature Data through Web Feature Service (WFS) 3.0, Web Map Tile Service (WMTS) 1.0, and GeoPackage (GPKG) 1.2,
- the styling approach, challenges, and interoperability considerations discovered during the initiative, and
- any extensions required or best practices recommended to facilitate development, encoding, offering, and exchange of styles. This includes how styles are offered from servers, how the desired style offering can be selected by the client from multiple server style offerings (e.g. GetStyles request), and how clients can apply their own styles.

Throughout this summary, references are made to the later chapters of the ER to allow for easy discovery of specific technical explanation without repeating them here.

1.1. Requirements & Research Motivation

Following the extension of WFS, WMTS, and GeoPackage to support tiled feature data and the creation of draft specifications to conceptualize tiled feature data created during the VTP, stakeholders observed the need to address styling for tiled feature data. In order to fully realize the proposed tiled feature data extensions, methods by which to ensure a consistent standard across the OGC standards baseline needed to be addressed. The requirements and recommendations for the VTPEExt were derived from concerns identified during the initial VTP. These requirements and recommendations are outlined in [Table 1](#) along with their associated Concept of Operations (CONOPs) items detailed in [Concept of Operations](#).

Table 1. Recommendations and Requirements

Recommendation	Requirement	CONOPs Reference
Establish a method for creating Tiled Feature Data styles independently of the production of Tiled Feature Data.	Implement the capability to allow for the production of Tiled Feature Data Styles	5.1

Recommendation	Requirement	CONOPs Reference
Investigate methods for serving and requesting Tiled Feature Data styles using OGC standards.	Demonstrate the application of Tiled Feature Data styles across WFS, WMTS and GPKG services for Desktop, Web and Mobile.	5.2 and 5.4
Demonstrate portrayal of Tiled Feature Data styles for varying display environments in a standardized manner.	Implement the use of three styles (topographic, satellite overlay, and night / high contrast) to cover the a demonstrable range of display environments.	5.2 to 5.4
Establish a method for storing Tiled Feature Data styles in a form suitable for workflows in Denied, Degraded, Intermittent or Limited (DDIL) environments.	Implement a method which stores the Tiled Feature Data styling in a GeoPackage while keeping the style decoupled from the data.	5.3
Implement Tiled Feature Data styles in the OGC Style Layer Descriptor (SLD) and Mapbox style (MBstyle) formats.	The VTPEExt outputs should use both SLD and MBstyle standards and address the associated encoding implications for both OGC Web Services and GeoPackage extensions.	5.1 to 5.4

The previous list of recommendations and requirements showed that further consideration was needed to establish an approach for implementing tiled feature data styling across the OGC standards baseline. This includes OGC web services (WFS, WMTS), integrated clients on multiple platforms (web browser, desktop, and mobile), multiple style standards and formats (Mapbox, SLD), and multiple data formats (Mapbox Vector Tiles, GeoJSON Vector Tiles, and GeoPackages). Along with these requirements, there was also a need to demonstrate the integrated use of tiled feature data styles in existing applications and technologies to prove the feasible use of styles in an operational context.

1.2. Findings and Challenges encountered

Participants were able to demonstrate the portrayal of the common baseline tiled feature data styles across all three platforms, with very little visual difference between the Mapbox and SLD style formats. This portrayal provides a standardized visualization without the need for users to be aware of which format they are viewing. Each approach for the WFS, WMTS, and GeoPackage clients was slightly different depending on the standard implementations and the associated client types (see [Implementation Approaches](#)). The participants validated that they successfully addressed one or more of the above requirements through Technology Integration Experiments (TIEs). These TIEs built on the initial VTP TIEs and added style implementations. The outputs from the TIEs are presented visually on a [Youtube Channel](https://www.youtube.com/playlist?list=PLQsQNjNIDU86fv8nJP0KT9C81ZbwvldWO) [https://www.youtube.com/playlist?list=PLQsQNjNIDU86fv8nJP0KT9C81ZbwvldWO] and are recorded in the section [Technology Integration Experiments](#).

The initial WFS and WMTS visualization of the three feature styles was achieved fairly early in the process as part of the exploratory integration of the styles into the existing VTP demonstrations. This involved both extending the server side WFS and WMTS services to allow for the styles to be retrieved and adding the ability to call these new server side capabilities to web clients. For both service standards, additional API elements were added to allow access to both Mapbox and SLD styles for the same service. In the section [Implementation Approaches](#), participants outlined how their capabilities incorporate styles into the existing standards. These implementations did encounter limitations of the existing service standards regarding the separation of styles from the feature data. This was addressed by a number of participants who used a number of alternative methods to allow styles to be appropriately referenced. Nonetheless, participants agreed that WMTS requires modification to handle styles separately from the feature data.

While implementing these standard extensions, some participants explored the possibility of supporting both server-side and client-side rendering in their integrated clients. This capability provides additional flexibility in DDIL scenarios. Some participants provided clients that render the styles locally. One of the clients provides a simple tool for choosing whether the styles are rendered server side or client side. Since users may wish to alter a style without relying on an online service, support for style editing in addition to style rendering was also explored.

Implementing style functionality in the GeoPackage extension constituted much of the effort during the VTPEExt. The primary challenge was storing the styles in a manner which allows for easy discovery of the styles without additional overhead for client applications and users ([GeoPackage Provisioning](#)). In addition, participants investigated approaches for allowing GeoPackages to store OWS Context documents which in turn reference styles for tiled feature data ([Compusult](#)). The continued development of this style functionality involved considerable overlap with the GeoPackage Extensions work and discussion with the GeoPackage Standards Working Group (SWG).

Participants noted that the server-side and the GeoPackage-style approaches should contribute to a common [Conceptual Model](#) that would be flexible enough to support both online and offline implementations. The conceptual model devised represents the ideal solution with styles sitting independently from features. However, as explained above, the existing service standards prevented the complete separation of styles from the features. Therefore, the goal outlined in the conceptual model was only partially achieved. The scope of the VTPEExt did not allow for updating the existing service standards to provide this complete separation. That said, the implementations partially achieved the objective so this work should represent a solid basis for the emerging Open Portrayal Framework that will be explored during Testbed-15.

The possible need for style conversion between the two formats, Mapbox and SLD, was also identified. Some of the participants provided conversion support while others used preexisting conversion functionality, such as in GeoServer, to demonstrate conversion capabilities. Again, the observation was made that support for style conversion should be standardized in the future for WFS, WMTS, and GeoPackage.

1.3. Prior-After Comparison

Prior to VTPEExt, there was no consensus regarding the appropriate implementation of styling for tiled feature data. Although Mapbox styles were successfully used during the VTP, there was no

standardized portrayal across all demonstrators. In addition there was no method for users to create or apply styles to tiled feature data, which is a major benefit that Vector Tiles afford, or methods for using SLD standards for styling Vector Tiles served by Web Map Service (WMS) and WMTS. Furthermore, there was no established method for styling feature data within GeoPackages. This hindered the ability for GeoPackage clients to portray feature data in a common way.

This work has produced demonstrable additions to the proposed VTP extensions, tested across compatible Commercial Off-The-Shelf (COTS) applications using different style encodings, to show the appropriate use of styles with the proposed OGC Vector Tile Extensions.

By addressing the above requirements in the context of real-world application, the demonstrators account for the use of styles by end users as well as the technical functionality required to serve Tiled Feature Data. For instance, the demonstrators include the ability for users to change style colors, apply new styles and create new styles via a user interface.

1.4. Recommendations for Future Work

The results from the VTPEExt provide a solid foundation for the emerging Open Portrayal Framework, which will be explored further in the Portrayal Thread of Testbed-15.

1. There is not currently an OGC standard for styles encoded in JavaScript Object Notation (JSON). Some stakeholders have indicated that a JSON-based encoding would be simpler to implement than SLD. While Mapbox Styles is a JSON encoding, it has certain limitations that make it inappropriate for some applications. Any work on a new encoding should be based on the emerging Open Portrayal Framework conceptual model.
2. Limitations were discovered in the WMTS model that make it impossible to add new styles to an existing system. An OpenAPI-based approach similar to the approach used in WFS 3.0 would provide valuable flexibility.
3. A draft [Styles API](#) has been developed and implemented during VTPEExt based on the WFS 3.0 Core API and the emerging Open Portrayal Framework conceptual model. The scope of the API has been limited to the aspects that have been in focus in VTPEExt. A number of [open issues and items for future work](#) have been identified.
4. Recent experiments outside of the VTP have demonstrated that it is possible to publish tiled feature data via a serverless architecture, i.e., one where the tiles are published directly to a web accessible location (e.g., a file bucket) using predictable URLs, not via web services such as WMTS and WFS. OGC should investigate how this approach fits in with the rest of the OGC Standards Baseline.
5. There are a number of possibilities for future work pertaining to attributes inside GeoPackage. This Pilot investigated a number of approaches for storing attributes outside of the tiled feature data but still inside the GeoPackage. The decision made by the participants was a compromise based on expediency that does not necessarily satisfy all of the requirements.
6. GeoPackage does not currently have an interoperable mechanism for symbols. A candidate approach was proposed, but other activities took priority. Future work to address this gap would allow symbols to be shared and displayed consistently in different GeoPackage clients.

1.5. Document contributor contact points

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

Contacts

Name	Organization
Jeff Yutzler	Image Matters
Theo Brown	Helyx SIS
Sam Meek	Helyx SIS
Carl Reed	Carl Reed and Associates
Clemens Portele	interactive instruments
Andrea Aime	GeoSolutions
Stefano Bovio	GeoSolutions
Adam Parsons	Compusult
Keith Pomakis	CubeWerx
Jerome Jacovella-St-Louis	Ecere
Terry Idol	OGC
Gobe Hobona	OGC
Jeff Harrison	AGC
Matt Sorenson	Strategic ACI

1.6. 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 2. 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 17-069, OGC® Web Feature Service 3.0: Part 1 - Core Candidate Standard, 2018 [https://rawgit.com/opengeospatial/WFS_FES/master/docs/17-069.html]
- IETF: RFC-7946 The GeoJSON Format, 2016 [<https://tools.ietf.org/html/rfc7946>]
- IETF: RFC-1951 DEFLATE Compressed Data Format Specification version 1.3, 1996 [<https://tools.ietf.org/html/rfc1951>]

Chapter 3. 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.

- Extended GeoPackage

A GeoPackage that contains any additional data elements (tables or columns) or SQL constructs (data types, indexes, constraints or triggers) that are not specified in this encoding standard.

- GeoPackage file

A platform-independent SQLite database file that contains GeoPackage data and metadata tables with specified definitions, integrity assertions, format limitations and content constraints.

- Stylable Layer Set

A StylableLayerSet is a set of layers (those identified as associated with that StylableLayerSet) to which a particular set of style sheet documents (those associated with that StylableLayerSet) can be applied to. The multiple layers within a multi-layer tile set would typically be associated with a unique StylableLayerSet. A StylableLayerSet could also be shared by multiple tile sets meant to be used together (especially when each tile set contains a single layer), or by a group of tile sets or layers following the same schema(s).

- Style

A style organizes the rules of symbolizing instructions to be applied by a rendering engine on one or more geographic features and/or coverages. (from working group consensus Jan 18, 2019)

- Style Sheet

A style sheet is a container for styling rules for a single layer or for multiple layers.

- Styles API

A Web API for accessing, uploading, deleting and editing styles.

- Tile

A tessellated representation of geographic data, often part of a set of such elements, covering a spatially contiguous extent which can be uniquely defined by a pair of indices for the column and row along with an identifier for the tile matrix (adapted from OGC 07-057r7)

- Tile Set

A definition how tiles are organized. It contains a definition of the geographic extent and geographic location as well as a coordinate reference system

NOTE

A comment on the correct terminology for 'Vector Tiles' is appropriate before continuing. The decision made by the participants is as follows: **When explicitly referring to the conceptual model and addressing tiled point, linestring and polygon data in a formal context the correct terminology is Tiled Feature Data. When discussing the initial pilot, the extension work and the associated extensions using the terms 'Vector Tile(s)' is accepted.**

3.1. Abbreviated terms

- BLOB Binary Large Object
- COTS Customer Off The Shelf
- DDIL Denied, Degraded, Intermittent, or Limited
- GPKG GeoPackage
- KML Keyhole Markup Language
- MVT Mapbox Vector Tiles
- OGC Open Geospatial Consortium
- OWS OGC Web Services
- RTE Related Tables Extension
- SRS Spatial Reference System
- SWG Standards Working Group
- VT Vector Tiles, Vector Tiling, Vectiles
- VTP Vector Tiles Pilot
- VTPEExt Vector Tiles Pilot Extension
- WFS Web Feature Service
- WMS Web Map Service
- WMTS Web Map Tile Service

Chapter 4. Overview

- Section 5 presents a CONOPs for the use of tiled feature data. This includes a number of use cases that are examined in further detail later in the document.
- Section 6 describes how Pilot participants delivered the capabilities identified as needed by the CONOPs. This section includes the overall pilot architecture, strategic approaches, and the TIEs that were performed as part of the Pilot.
- Section 7 presents the implementation approaches for WFS 3, WMTS, and GeoPackage styling extensions that were demonstrated during the Pilot.
- Section 8 describes discussion topics that came up during the Pilot. This includes topics other working groups may wish to discuss outside of the confines of the VTPEExt.
- Appendix A presents informative GeoPackage Tiled Vector Data Extensions specifications that were developed as part of the Pilot.
 - Appendix A.1 presents the GeoPackage Tiled Feature Data Extension. This extension provides support for tiled feature data through the GeoPackage *tiles* option. Instead of PNG or JPG files, each tile BLOB is a Vector Tile.
 - Appendix A.2 presents the GeoPackage Mapbox Vector Tiles Extension. This extension allows the content of a GeoPackage tile BLOB to be a Mapbox Vector Tile as per the [Mapbox Vector Tiles \(MVT\) specification](https://www.mapbox.com/vector-tiles/specification/) [https://www.mapbox.com/vector-tiles/specification/] [version 2.1](https://github.com/mapbox/vector-tile-spec/tree/master/2.1) [https://github.com/mapbox/vector-tile-spec/tree/master/2.1].
 - Appendix A.3 presents the GeoPackage GeoJSON Vector Tiles Extension. This extension allows the content of a GeoPackage tile BLOB to be a GeoJSON file.
 - Appendix A.4 presents the GeoPackage GeoPackage Styles Extension. This extension allows for styles and symbols to be stored as BLOBs inside dedicated tables.
 - Appendix A.5 presents the GeoPackage OWS Context Extension. This extension provides a way to store information describing a list of geospatial resources, including but not limited to maps, their layers, and the styles of those layers.
- Appendix B presents the GeoPackage extensions in Appendix A as normative requirements that could be the basis for an OGC standard.
- Appendix C presents API building blocks for managing and fetching styles via a Web API based on WFS 3.0 Core standard.
- Appendix D presents a WMTS 1.0 profile that defines a set of KVP operations and RESTful endpoints providing a client with the ability to GET, PUT and DELETE style definitions.
- Appendix E presents the revision history of this document.
- Appendix F contains a Bibliography.

Chapter 5. Concept of Operations

5.1. Data Preparation

1. After acquiring or producing feature data, a data manager produces tiled feature data.
2. A cartographer produces style sheets that define the rules for portraying tiled feature data on a map. These are organized into style sets which correspond to a single (feature) application schema. There may be more than one option for each style set.

5.2. Web Services / Web APIs

1. A data manager prepares a dataset consisting of features.
2. An administrator deploys the dataset to a geospatial data server and configures the server to serve the data in one or more of the following ways:
 - a. as features via a Web API that conforms to the draft Web Feature Server 3.0 standard (WFS),
 - b. as tiled feature data via a web service that conforms to the Web Map Tile Server (WMTS) standard or via a Web API that extends the WFS API mentioned in the previous item
 - c. as server-rendered bitmap image tiles of the features
3. An administrator deploys style sheets (see [Data Preparation, #2](#)) to the server.

5.3. GeoPackage Provisioning

1. A data manager produces one or more GeoPackages containing tiled feature data. In particular, these GeoPackages will allow operators to function in Denied, Degraded, Intermittent, and Limited (DDIL) environments.
 - a. In the simpler case, the data manager produces GeoPackages without modifying the source vector tiles.
 - b. In the more complex case, the data manager modifies the source vector tiles to remove the attribute information and store the attributes in relational tables. This approach supports more efficient data storage, querying, and analysis.
2. Optionally, the data manager adds style sheets (see [Data Preparation, #2](#)) to the GeoPackage to support subsequent tiled feature data visualization.
3. Optionally, the data manager adds offerings that correlate available tiled feature data layers to available style sheets.
4. Optionally, the data manager adds OWS Contexts to the GeoPackage to allow an operator to select from a predefined set of map views.
5. A data manager deploys one or more GeoPackages to the target device.

5.4. Integrated Clients

1. An operator uses an integrated client to bind with one or more web services containing tiled

feature data layers. Once that binding has occurred, the analyst may be able to do the following:

- a. Retrieve vector tiles from a server
 - b. Select an available style sheet (see [Web Services / Web APIs, #3](#)) to portray the tiled feature data with specific styling rules
2. An operator uses an integrated client to open one or more GeoPackages containing tiled feature data layers. Once the GeoPackage has been opened, the operator may be able to do the following:
- a. If style sheets (see [GeoPackage Provisioning, #1b](#)) have been added to the GeoPackage, the analyst may select a style sheet for each tiled feature data layer.
 - b. If offerings have been added (see [GeoPackage Provisioning, #3](#)), the analyst may be able to select from a list of offerings that reference both tiled feature data layers and associated style sheets.
 - c. If OWS Contexts (see [GeoPackage Provisioning, #4](#)) have been added to the GeoPackage, the analyst may select from them directly.
3. An operator uses an integrated client to perform visualization and/or analysis on tiled feature data. Potential capabilities include the following:
- a. Display the tiled feature data on a map, using a default style and/or a specific style sheet
 - b. Query the tiled feature data to filter or isolate an individual feature
 - c. Perform analysis on tiled feature data, which could include something as basic as displaying specific features and attributes (e.g., in tabular form) or something more complex

Chapter 6. Meeting the Challenge

6.1. Pilot Architecture

The architecture of the pilot is illustrated in [Figure 1](#).

It shows the three types of clients that are intended to consume tiled feature data and produce and use styles:

- Desktop clients,
- Web clients, and
- Mobile clients.

On the producer side, there are also three starting points that are explored for storing and providing access to tiled feature data and styles:

- Servers that provide access to features via the WFS 3.0 API. In this case, the the API has been extended to provide access tiles and styles as additional resource types in addition to the features.
- Servers that support the WMTS 1.0 standard and in VTPEExt provide access to the tiled feature data. In VTPEExt the servers also support an extended interface for managing and accessing styles for portraying the tiled feature data (server-side or client-side portrayal).
- SQLite databases according to the GeoPackage 1.2 standard. Additional tables have been defined for storing tiles and style sheets.

Mapbox Vector Tiles and GeoJSON are used to encode the tiled feature data, Mapbox Style and SLD are used to encode styles in style sheets.

OGC Vector Tiles Pilot (VTP) Architecture

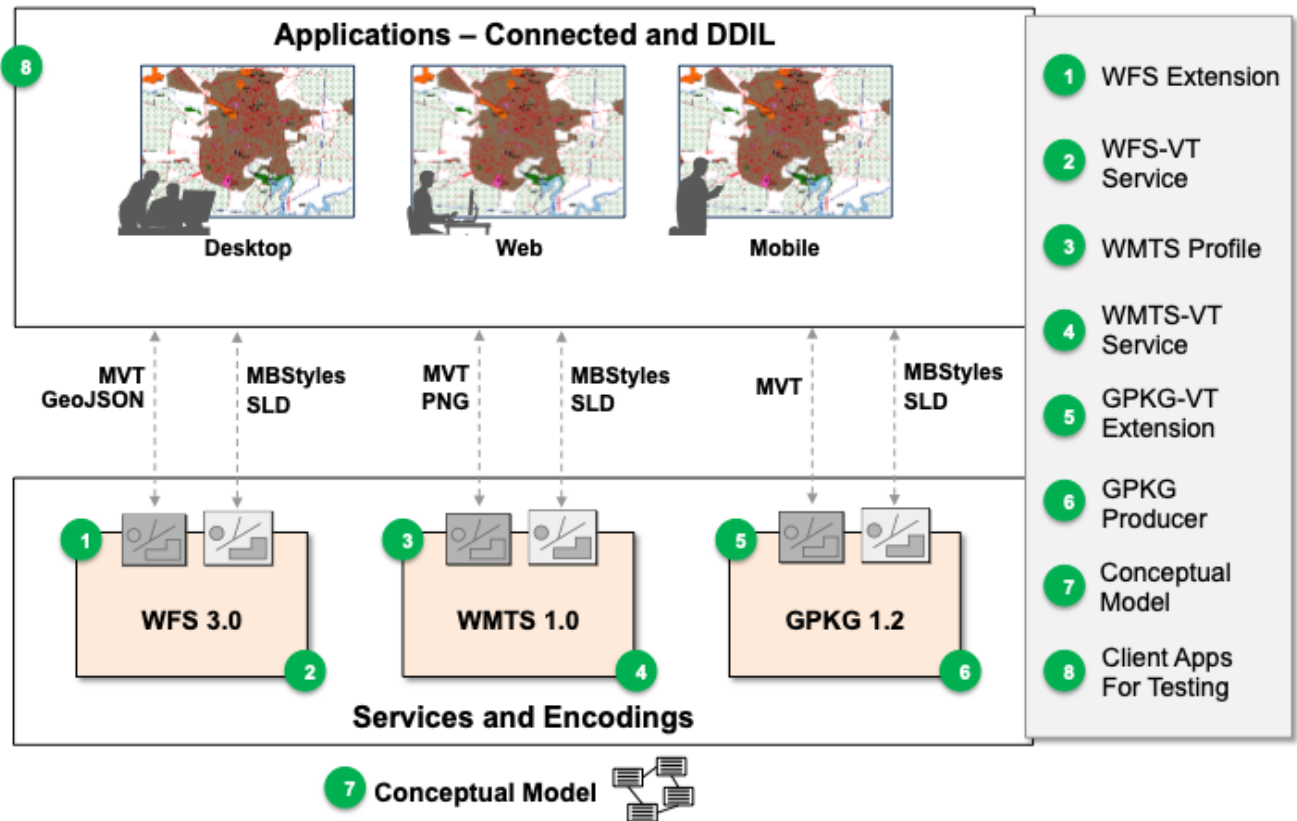


Figure 1. Vector Tiles Pilot Architecture

This architecture attempts to address tiled feature data consistently across the relevant suite of OGC standards, that is based on a common conceptual model. This approach provides implementers with guidance for tiled feature data no matter their use case.

NOTE

This document sometimes uses the term "WFS" as a shorthand notation for a server that provides a Web API according to the draft WFS 3.0 Core standard, extended with additional resource types for tiles and styles, following the same API design approach of WFS 3.0 and using OpenAPI. The API building blocks for the tile and style resources, however, are unlikely to be standardized as part of the WFS 3.0 series since the scope of WFS 3.0 are features. Instead tiles and styles should be specified in other standards (revisions of existing OGC standards or new OGC standards).

6.2. Conceptual Model

This project introduces the concepts of stylesheets and style sets. Once a style set is established and associated with one or more tile sets, it is then possible to provide the user with a set of options for portraying those tile sets. This work has further developed the Tiled Feature Data Conceptual Model created during the initial VTP [2] to account for Feature Tile styling, see Figure 2. This Conceptual Model underpins the Vector Tiles Pilot Architecture above (Figure 1, labeled number 7).

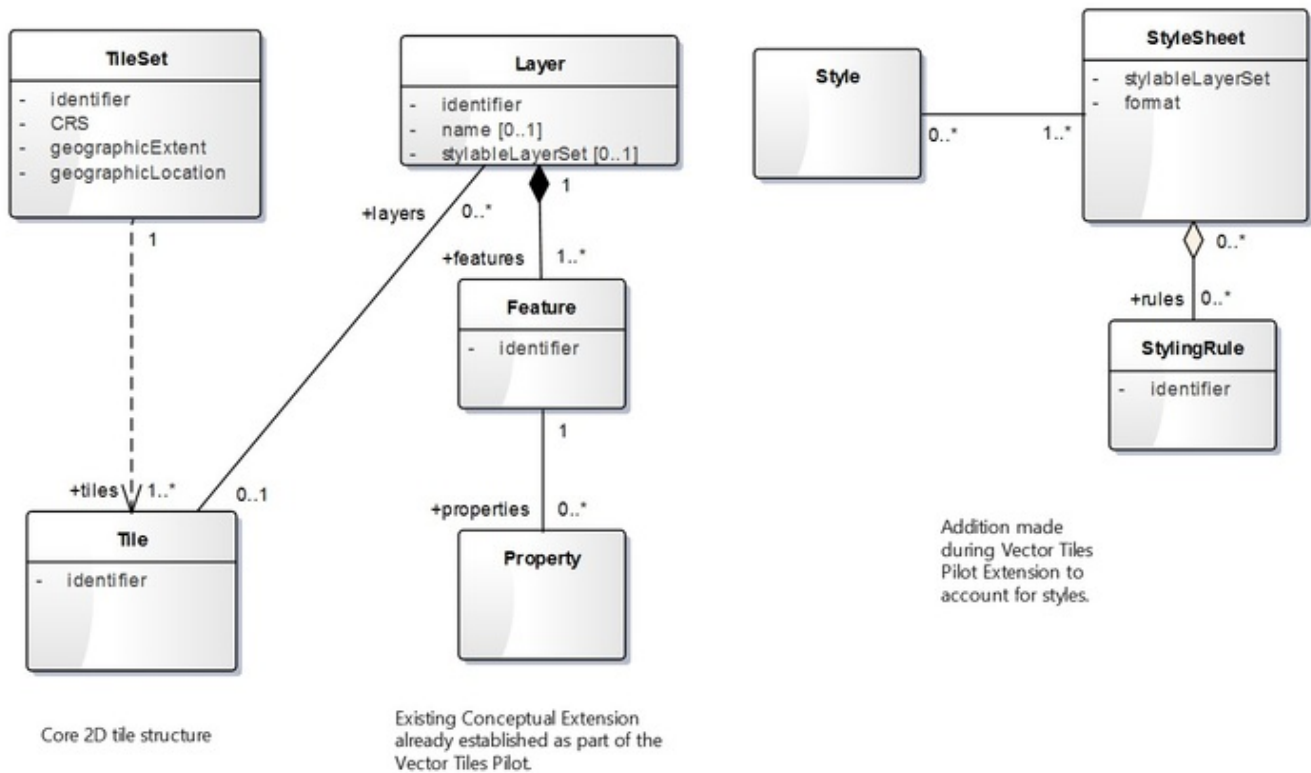


Figure 2. Tiled Feature Data Conceptual Model

The extension to the Tiled Feature Data Conceptual Model is described by the classes 'Style', 'StyleSheet' and 'StyleRule'.

- The **Style** class provides a choice of style, such as "Topographic", "Night", or "Satellite Overlay".
- The **StyleSheet** class is the primary concept in the extension for storing a collection of style rules. The two properties of **StyleSheet** depicted in the conceptual model are identifiers. The combination of these two identifiers and a **Style** provides a single **StyleSheet**.
 - The **format** property of the **StyleSheet** class stores the format type, such as "Mapbox" or "SLD".
 - The **stylableLayerSet** property provides a further identifier, allowing for a more granular choice of style options. The **stylableLayerSet** identifier corresponds to the equivalent identifier in a **Layer**. This relates a **StyleSheet** to a set of features. Notice the **Feature** class also has a **stylableLayerSet** property. Therefore, if a **StyleSheet** has the **stylableLayerSet** OGCVectorTilePilot (as an example), this means that this **StyleSheet** is for styling the **Features** which have the **stylableLayerSet** OGCVectorTilePilot. This provides the ability to define views of data. If five **Features** have the **stylableLayerSet** OGCVectorTilePilot then a **StyleSheet**, under **Night Style**, with the **format** Mapbox and the **stylableLayerSet** OGCVectorTilePilot can be used to specify a night style in the desired format which is appropriate for a specific group of Features (in this case 5) which have the **stylableLayerSet** property OGCVectorTilePilot.
- The **StyleRule** class corresponds to the **Rule** class in the draft OGC Symbology Conceptual Core Model [3], as it is used to organize symbolizing instructions for a single feature.

Included in the Conceptual model on left side is the **stylableLayerSet** property in the **Layer** class. This allows for a data set creator to assign **Features** in Tiled Feature Data to a group for styling.

In the Conceptual Model ([Figure 2](#)) the style additions are intentionally left disconnected from the Tiled Feature Data model on the left hand side. This is to demonstrate that the styling extension depicted has been designed to be flexible enough to be applied to other data set types in the future, not just Tiled Feature Data. This also accounts for the independent creation of styles, meaning styles can be created and managed by cartographers independent of the Tiled Feature Data and their associated Tile Sets.

The whole Conceptual Model is representative of the agreed definition of a style (see [Terms and definitions](#)), which is repeated below for convenience:

A style organizes the rules of symbolizing instructions to be applied by a rendering engine on one or more geographic features and/or coverages.

In relation to the above model, the connected classes on the right-hand side ([Style](#), [StyleSheet](#), and [Style Rule](#)), are the elements which organize the rules of symbolizing instructions to be applied by a rendering engine.

The connected classes on the left-hand side represent an example of "one or more geographic features and/or coverages". In the above case the features are tiled feature data.

6.2.1. Conceptual Model Challenges

Devising a conceptual model for the VTPExt work was challenging due to the variety across all intended implementations. These included three OGC standards, two style formats, and three client implementations each with schemas and approaches. The description of this challenge below is intended to provide a non-technical explanation of the contrast between these elements why this had an impact on conforming to a single Conceptual Model. The technical explanations and implementation solutions for these challenges are in the following chapters and are referenced where appropriate.

WFS 3.0 and WMTS 1.0 follow considerably contrasting paradigms. The WFS 3.0 OpenAPI-based approach is inherently flexible and extensions (e.g., the WFS 3.0 landing page or the WFS collection level) can easily be added. The WFS 3.0 flexibility, as illustrated in [Figure 3](#), shows that a WFS can use any desired API structure to separate the styles from the layers.

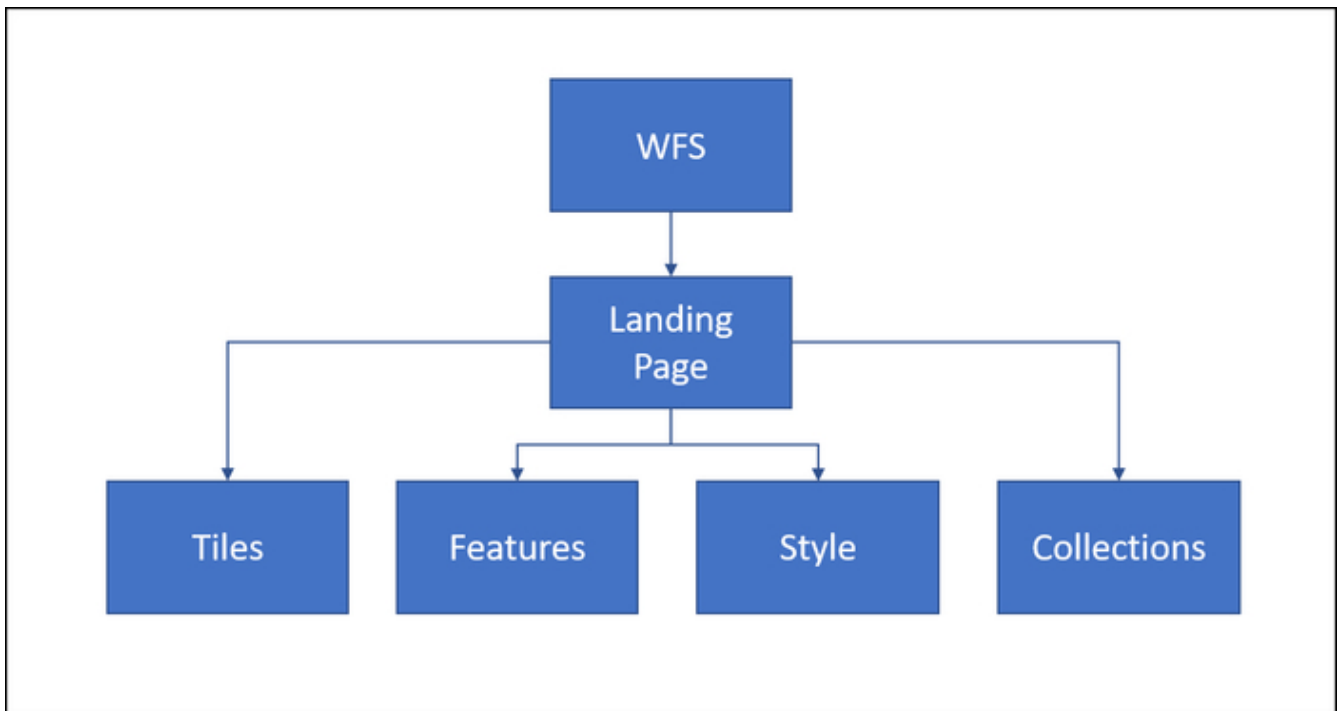


Figure 3. Basic WFS Structure

In contrast, the WMTS 1.0 document approach, as illustrated in [Figure 4](#), does not have this flexibility. This is because the base resource of a WMTS is the Capabilities Document, which contains the set of layers. A layer may contain multiple styles, each referring to a different pre-rendered tile cache. The Capabilities Document can only reference styles that are in a Layer and with this structure, there is no way to store styles independently of a layer. The interim solution was to use style references in the capabilities document rather than adding styling to existing WMTS.

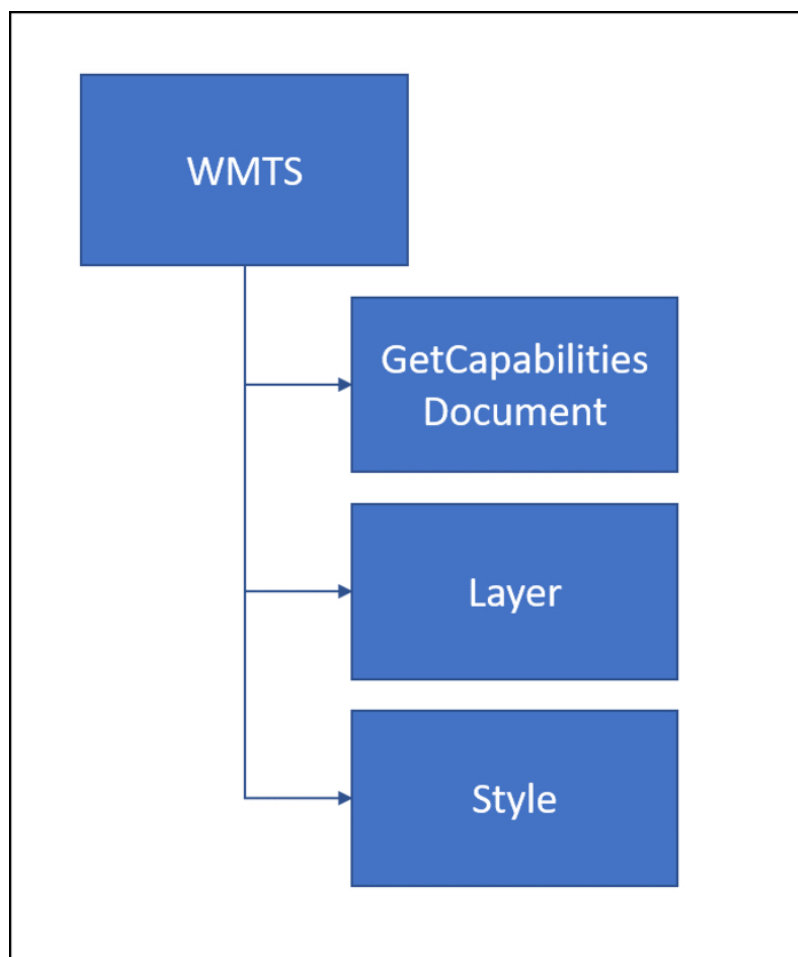


Figure 4. Basic WMTS Structure

6.2.2. SLD Model Versus Mapbox Styles Model

The formats chosen for this work are OGC's SLD style and the Mapbox Style format produced by Mapbox. These contrast significantly which provided another challenge to overcome. In the SLD model, a layer is defined by one or more styles. Each of these styles is defined by one or more feature-type styles, where each feature-type style is specific to a feature set that is available to the server. However, the Mapbox Styles model does not work this way. In the Mapbox Styles model, a style is a top-level object. The definition of a Mapbox style can be requested or specified outside of the definition of a layer. Also, a Mapbox layer is not defined by styles.

The proposed WFS style API does not suffer from these differences. The WFS does not have any pre-existing notions of layers and styles. Also, a WFS feature type is a conceptual match for a Mapbox layer, so the Mapbox Styles model is a more natural fit.

For WMTS, things are more complicated. The WMTS service as defined by OGC is built on top of the SLD model, with a strictly defined model for the relationship between layers and styles. That is, each WMTS layer has one or more styles. The WMTS interface does not allow the client to request a tile of a specific feature set, only of a specific style of a specific WMTS layer. For this reason, the natural endpoints for a WMTS styles API are:

- `{wmtsRestEndpointBaseUrl}/layers`
- `{wmtsRestEndpointBaseUrl}/layers/{layerId}`
- `{wmtsRestEndpointBaseUrl}/layers/{layerId}/{styleId}`

As a result, there is no top-level concept of a style in WMTS. That is, one cannot refer to a WMTS style outside of the context of the WMTS layer that it is part of. When a tile of a WMTS layer is requested, it must be requested with respect to a specific style. In addition, a WMTS layer without any styles has no content because the styles of a layer indicate what feature sets to render.

The Mapbox Styles model does not have the equivalent of a WMTS layer. A WMTS layer is not the same thing as a Mapbox layer, and a WMTS style isn't the same thing as a Mapbox style. The WMTS is not entirely incompatible with the Mapbox Styles model, however, CubeWerx has identified two mappings that could be used.

Mapping #1 (WMTS Layer \approx Mapbox Layer)

In this mapping, a WMTS layer is considered the rough equivalent of a Mapbox layer, and the set of all styles with same ID across all of the WMTS layers of the server is considered the rough equivalent of a Mapbox style sheet. In order for this mapping to work, a WMTS layer must be defined to serve exactly one feature set.

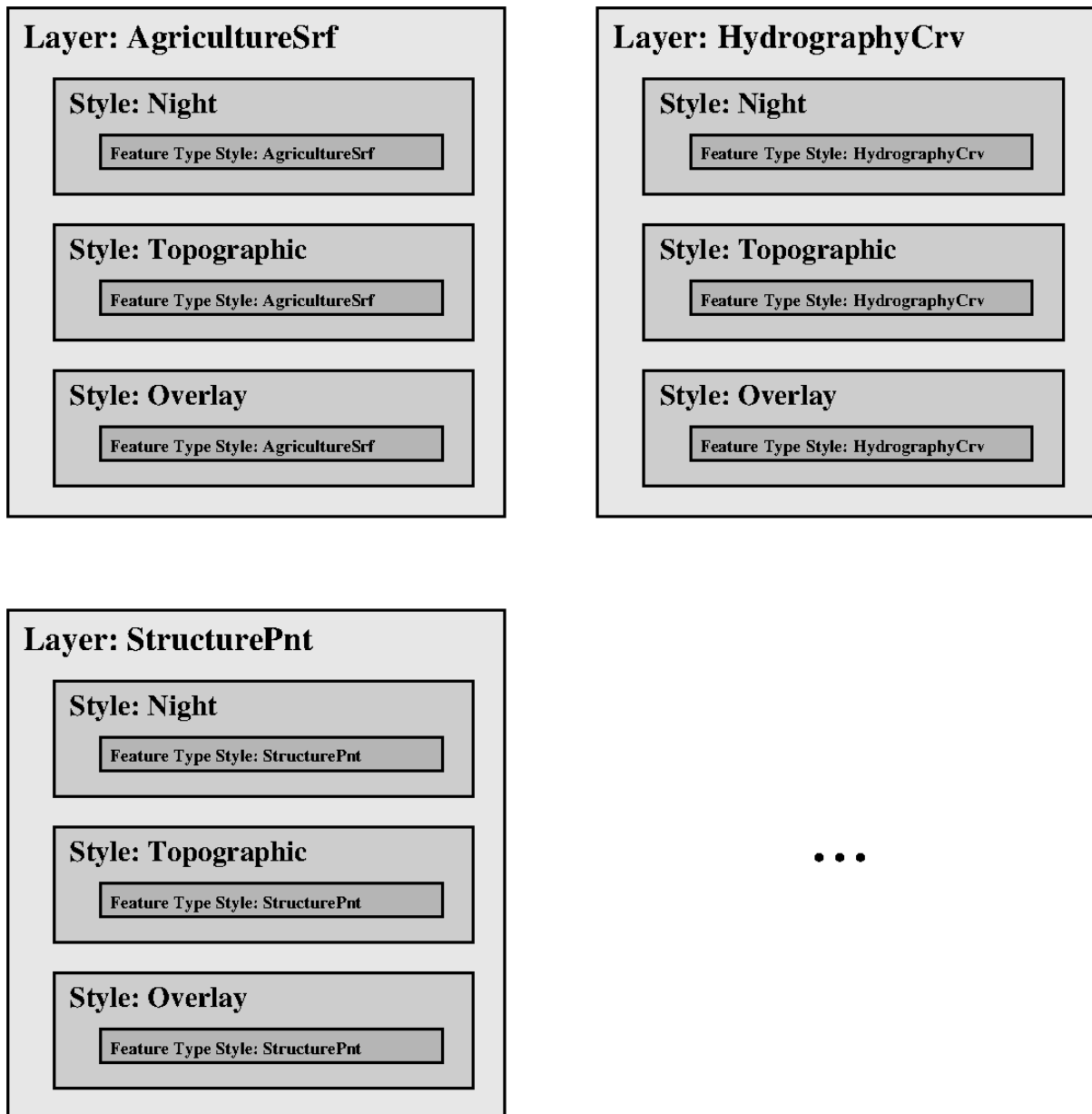


Figure 5. WMTS Mapping #1

This is where the `styles/{styleId}` endpoints as implemented by [CubeWerx's WMTS Mapping #1 Implementation](#) come into play. Each of these endpoints represents a style sheet, or in WMTS terms, the definition of a single style that's defined across multiple layers.

This mapping has two major disadvantages:

1. Since a WMTS tile always contains exactly one WMTS layer, the client would be required to fetch the tiles for each feature set separately. So if it takes N tiles to fill a map, and there are M feature sets, the client would need to request and render $N \times M$ separate tiles, which is not very feasible.
2. There is an awkward disconnect between the resources represented by the `styles/{styleId}` endpoint and the way styles are referenced in the rest of the WMTS API. For example, it would require the client to equate styles with equivalent IDs across multiple layers when presenting the user with styling options and when making the tile requests.

Mapping #2 (WMTS Style \approx Mapbox Style Sheet)

In this mapping, a WMTS style is considered the rough equivalent of a Mapbox style sheet, and the feature sets rendered by that style are considered the rough equivalent of a Mapbox layer. Therefore, a WMTS layer is nothing more than a set of style sheets. When a WMTS client requests a tile from a WMTS server, what it's doing is requesting a tile of a specific stylesheet of a WMTS layer. Every feature set in the `StylableLayerSet` list of that stylesheet is rendered into the tile, in the styles defined by that stylesheet.

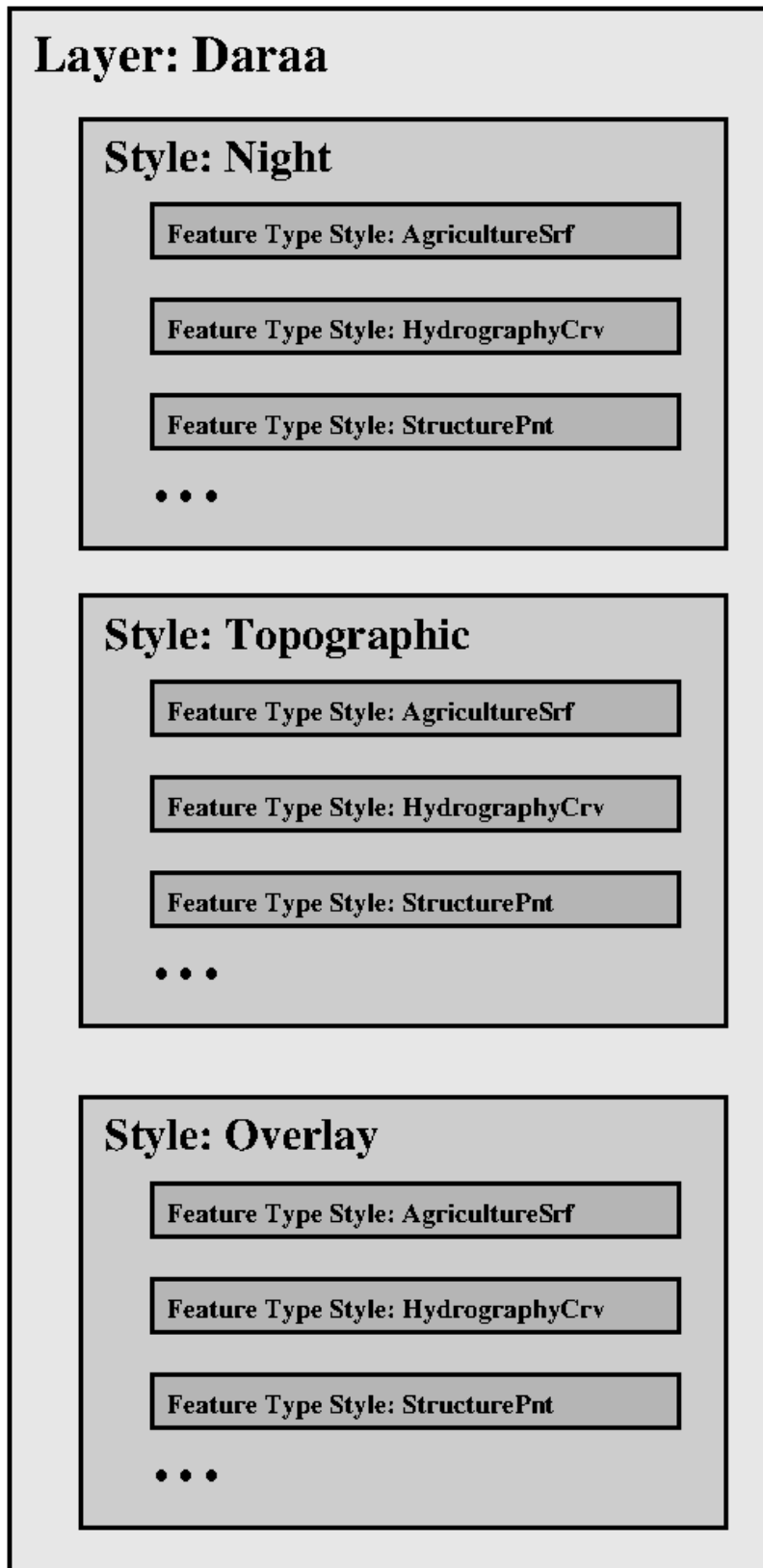


Figure 6. WMTS Mapping #2

This mapping has the following operations:

- **GET** `layers/{layerId}` to request all of the stylesheets of a WMTS layer
- **GET** `layers/{layerId}/{styleId}` to request a specific stylesheet of a WMTS layer
- **PUT** `layers/{layerId}/{styleId}` to define or redefine a stylesheet for a WMTS layer
- **DELETE** `layers/{layerId}/{styleId}` to remove a stylesheet of a WMTS layer.

This mapping does not suffer from either of the disadvantages of Mapping #1, since each tile would contain multiple feature sets (Mapbox Layers), and the awkward disconnect between the `styles/{styleId}` endpoint and the way styles are referenced in the rest of the WMTS API can be avoided.

One disadvantage of this mapping, though, is that a client cannot request a single feature set without PUTting a custom style containing just the desired feature set (assuming the user is even authorized to PUT). One way around this is to have the WMTS server serve an individual layer for each feature type in addition to the conglomerate layer. A client can determine the relationship between the conglomerate layer and the component layers in one of two ways:

1. by GETting the desired style of the conglomerate layer, collecting the list of reference feature sets, and assuming the convention that each of these feature sets are also requestable as a WMTS layer with the same ID as the feature set, or
2. by adopting a layer ID convention where the ID of a component layer is prefixed with the ID of the conglomerate layer and a colon (":"). E.g., if "Daraa" is the conglomerate layer, the client could know by convention that the layers "Daraa:AgricultureSrf" and "Daraa:CulturePnt" (also advertised by the WMTS capabilities document) are component layers of "Daraa".

Component layers have the advantage of being compatible with Mapping #1 as well.

6.3. Technology Integration Experiments

The table below presents the results of the Technology Integration Experiments (TIEs) for this Pilot Extension.

Table 2. GeoPackage TIEs

Producers\Clients	Image Matters	Compusult	Ecere
CubeWerx	X	X	X
Compusult	X	X	X
Ecere	X	X	X

Table 3. WMTS TIEs

Services\Clients	Compusult	Ecere	GeoSolutions (in-kind)
CubeWerx	X	X (no attributes support for lack of schema)	Not tested
CubeWerx (static)	X	(resource API not yet supported)	Not tested
GeoSolutions	X	(no GetStyles request for KVP API)	X (NOTE: Client uses static styles with WMTS vector tiles)
Ecere	X	X (no attributes support for lack of schema)	Not tested

Table 4. WFS TIEs

Services\Clients	Ecere	GeoSolutions
ii	X (no attributes support for lack of schema)	X
Ecere	X	X
GeoSolutions	X	X (no attributes support for lack of schema)

Chapter 7. Implementation Approaches

7.1. Data Preparation

7.1.1. Producing Tiled Feature Data

As part of this project, two formats for tiled feature data were used, namely Mapbox Vector Tiles (MVT) and GeoJSON Vector Tiles.

CubeWerx

The CubeWerx WMTS has been augmented to produce both Mapbox Vector Tiles (MVT) and GeoJSON Vector Tiles in addition to tiles in the standard JPEG and PNG image formats. These tiles are generated from the source data by simplifying them to the tile resolution. The tiles of the common zoom levels are pre-generated for optimum performance, and the rest are generated on demand and cached. A static WMTS has also been prepared, in which all of the tiles have been pre-generated.

interactive instruments GmbH

The provisioning of features and tiled feature data in VTPExt is unchanged from the Vector Tiles Pilot. The three datasets of OpenStreetMap data from Syria or Iraq that have been converted to the Topographic Data Store application schema of NGA have been deployed to a PostgreSQL database.

[ldproxy](https://interactive-instruments.github.io/ldproxy/) [https://interactive-instruments.github.io/ldproxy/], an open source software product that has been extended by interactive instruments in VTP and VTPExt to support the relevant API extensions, is then used to provide the datasets as features and as tiled feature data.

The tiles in Mapbox Vector Tiles and GeoJSON are generated from the feature data on demand. The most commonly used type of tiles (tiles with all features in the tile area and all the properties of each feature) are cached by the server in the file system of the server to increase performance in the most frequent use cases.

For more details see the Vector Tiles Pilot WFS 3.0 Engineering Report [4].

Compusult

The Compusult GeoPackage Producer runs as a web-browser based application, as well as being accessible via an OGC Web Processing Service (WPS) instance. The GeoPackage Producer supports producing Mapbox Vector Tile and GeoJSON Vector Tile based GeoPackages in a user selected projection system and a uploaded vector source (Shapefile(s), GeoDatabase, SqliteDB, etc.). The producer has the ability to convert all feature types into a single MVT tileset or produce one tileset for each feature type.

Feature type geometries have their bounds clipped using a buffer to ensure clients can render freely without having to worry about artificial line segments from tile bounds or clipped line strokes. Automatic layer order for drawing purposes is detected by examining feature type size, bounds, and type. Attributes can also be minimized by removing known empty/no data strings from

feature attributes.

Ecere

Ecere's GNOSIS SDK provides the capability to import data from a number of geospatial data formats and standard web services into its tiled data store. This data store consists of pyramidal multi-resolution tiled layers, following the variable width GNOSIS Global Grid. The tiled data is stored in the open GNOSIS Map Tiles format and data attributes are stored in SQLite databases. Ecere's GNOSIS Cartographer GIS tool is used to import this data and produce GeoPackages from it. Ecere's GNOSIS Map Server can serve tiles or whole features directly from this data store. Both the GeoPackage producer as well as the map server have the ability to re-project the data to a different coordinate reference system (e.g Web Mercator rather than WGS-84), to re-tile the data according to a different tiling scheme (e.g. GoogleMapsCompatible), or to re-encode the data in a different format. In addition to GNOSIS Map Tiles, Mapbox Vector Tiles, GeoJSON, GeoECON, and Geography Markup Language (GML) tiles are currently supported.

7.1.2. Producing Stylesheets

GeoSolutions

GeoSolutions produced stylesheets by hand, writing each of the three styles (topographic, satellite overlay, and night) in the two different languages (SLD and Mapbox Styles). This first approach helped participants to understand differences between these encodings and implement proper language converters client side to get a final translated OpenLayers style. GeoServer is also able to transform MBStyle and GeoCSS to SLD automatically, this conversion has been tested in clients and compared with the manually setup styles.

Ecere

Although Ecere mainly used the style sheets produced by GeoSolutions for the purpose of this pilot phase, Ecere's GNOSIS Cartographer visualization tool has the ability to import, edit, and export style sheets. This tool currently supports exporting style sheets to SLD/SE, but this functionality is still being improved upon. Support for importing Mapbox GL styles was a major focus of the VTPEExt. Style sheets in GNOSIS Cascading Map Style Sheets format were handwritten as the export functionality for this format is still being implemented. Both the GeoPackage producer and the map server will eventually have the ability to produce styles on the fly in any requested supported format. With the ability to import styles definitions from multiple supported formats, it will effectively act as a styles translator. However, for the purpose of this pilot, style sheets documents were pre-loaded.

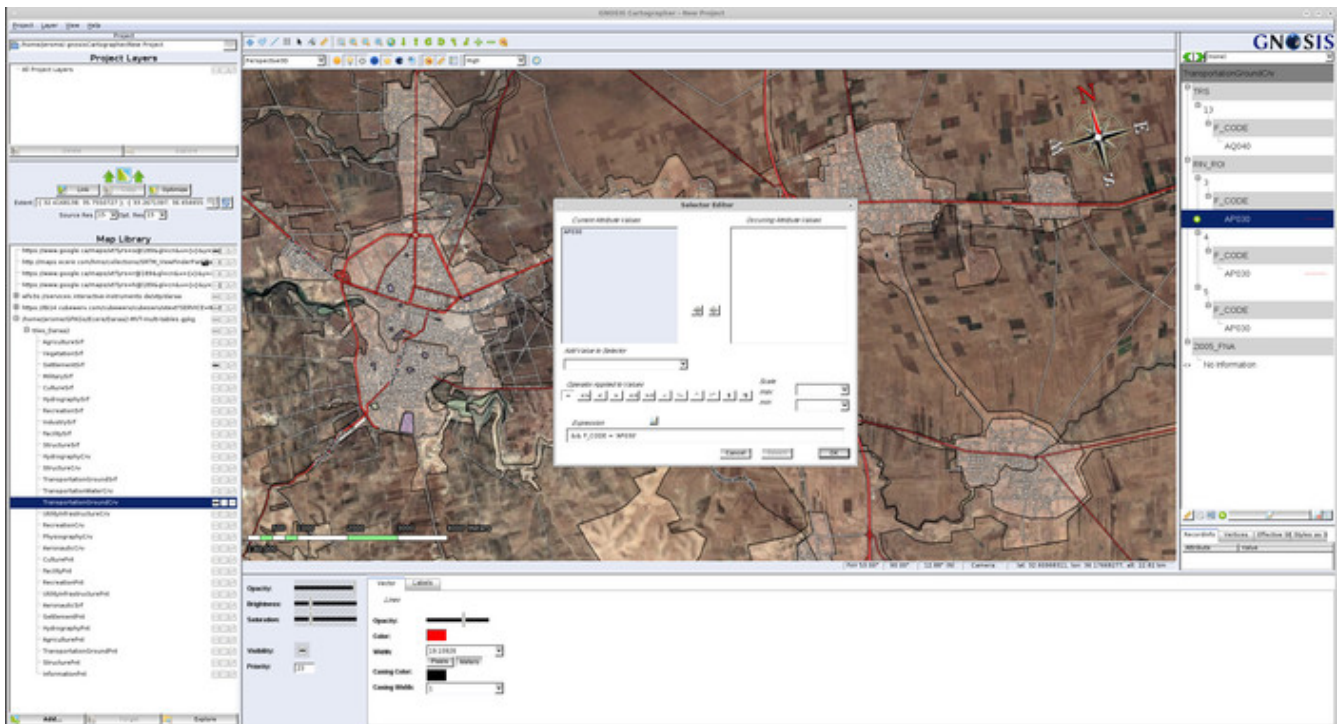


Figure 7. Styles editor in Ecere's GNOSIS Cartographer

7.2. Web Service Implementations

7.2.1. Web Feature Service

Analysis

Since tiled feature data is primarily used for visualization, it is generally not desirable to show all data at all zoom levels due to the possibility of overcrowding the tiles with lots of sub-pixel features. Of course, it is still possible, while inefficient, to associate the collection/layer with a style with no scale dependencies and no feature filtering, and let the generalization mechanism determine if a feature is big enough to fit.

GeoSolutions

The GeoServer WFS3 module has been extended during the pilot to deliver vector tiles and styles. GeoServer already had the ability to produce Mapbox Vector Tiles as a WMS and WMTS output. In both cases, the data sources are read in accordance to the retrieved bounding box and the style associated server side in the GeoServer configuration, which is used to drive scale dependencies and feature filtering, or in other words, determine what the contents of the vector tiles should be.

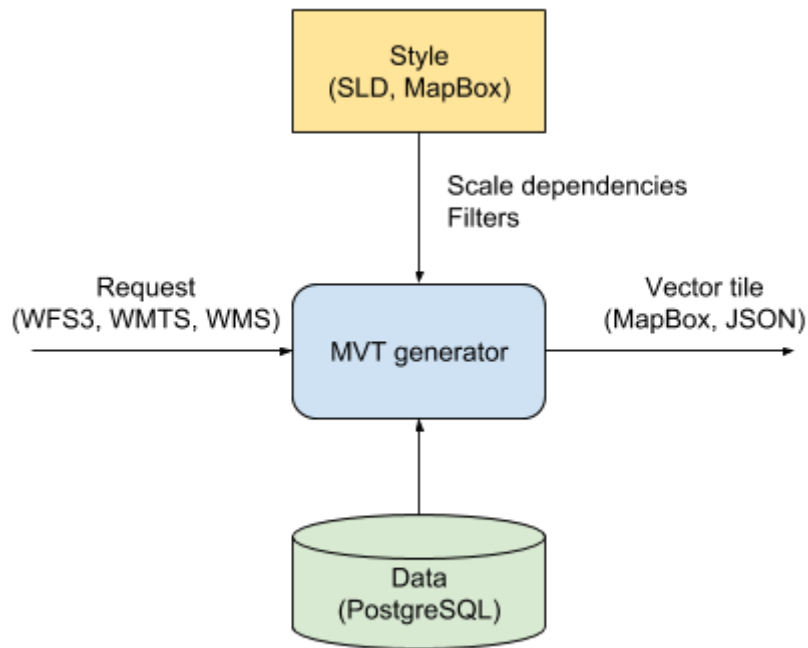


Figure 8. GeoServer Vector Tiles Production

Tiles in WFS3 are always generated on-the-fly from data, although a caching layer may be added in the future. Tiles are exposed via the following resources:

- `wfs3/tilingSchemes`, to retrieve all available tiling schemes
- `wfs3/tilingSchemes/{tilingSchemeId}`, to retrieve a specific tiling scheme by id
- `wfs3/collections/{collectionId}/tiles`, to retrieve all available tiling schemes for the collection
- `wfs3/collections/{collectionId}/tiles/{tilingSchemeId}/{zoomLevel}/{row}/{column}`, to retrieve a tile of the dataset, eventually specifying the format

The styles can be deployed in a variety of ways:

- Using the GeoServer user interface, to upload them or create them, eventually edit them, and associate them with layers as needs be
- Using the GeoServer RESTful administration API, which allows other applications to manipulate the configuration programmatically via service calls
- Using the WFS style extensions developed during the pilot.

In particular, the styling extension provides the following resources:

- `wfs3/styles`, to list all dataset level styles, allows creation of new styles via POST
- `/styles/{styleId}`, to retrieve a style body of a particular style (with a format specification, allowing for on the fly conversion when supported), but also to modify it, via PUT, or remove it, via DELETE
- `wfs3/collections/{collectionId}/styles`, which lists the styles associated to a particular collection, and allows creation of a new one via POST request
- `wfs3/collections/{collectionId}/styles/{styleId}` allowing to get the body of a collection

associated style, as well as creation, modification and removal via the PUT and DELETE methods.

Dataset and collection styles serve different workflows:

- A dataset style typically contains styling directives for a number of collections. In order to produce a map the client will first fetch the style, and then go back to the server to retrieve the necessary tiles from the referenced collections.
- A collection style typically only styles the collection at hand, so it is more suitable for a workflow where the client first decides which data/themes/collection to display, and only after needs to locate styles suitable to display them.

It is to be noted that this approach was discovered to be incompatible with GeoServer's own style handling, and more in general, with a general notion of cross layer style sharing. A style can be shared across layers under a few conditions, that are often met in real deployments:

- The style is so simple that it will not have style dependencies, e.g., a generic "red square point" style can be used against any point layer, like the GeoServer built in [point style](https://raw.githubusercontent.com/geoserver/geoserver/2.14.2/data/release/styles/default_point.sld) [https://raw.githubusercontent.com/geoserver/geoserver/2.14.2/data/release/styles/default_point.sld]
- The style is sophisticated enough that it will automatically determine the geometry type of the feature being displayed and do something sensible with it, like the [GeoServer generic style](https://raw.githubusercontent.com/geoserver/geoserver/2.14.2/data/release/styles/default_generic.sld) [https://raw.githubusercontent.com/geoserver/geoserver/2.14.2/data/release/styles/default_generic.sld]
- A style that references attributes in the dataset, in an environment where there are naming convention for particular attributes appearing in different layers, e.g. "indicator"

Based on these observations, a change of the styling API is suggested to make collections link to styles, instead of owning them and their bodies, for example:

- `wfs3/styles`, to list all styles, along with some attribute to determine if the style is meant to be a basemap or to depict a specific collection
- `wfs3/collections/{collectionId}styles`, listing pointers to the resources above, and allowing modifications via PUT to modify the "collection to style" associations

Finally, a performance related note. GeoServer WFS3 cannot currently deliver multi-layer collections, meaning a client needs to fetch vector tiles from several collections in order to build a complex map. Given that WFS3 is not schema driven, it would be actually possible to create a multi-layer collection delivering an aggregate vector tile (leveraging "layer groups", see also the WMTS GeoSolutions section), or add an extension to request multiple collections at the same time (being a dynamic request parameter, it may conflict with an eventual caching layer).

In order to deploy a GeoServer with vector tiles support, one can download a [development version of GeoServer](http://geoserver.org/release/master/) [http://geoserver.org/release/master/], add the [vector tiles extension](https://docs.geoserver.org/latest/en/user/extensions/vectortiles/index.html) [https://docs.geoserver.org/latest/en/user/extensions/vectortiles/index.html], and include the [WFS3](https://build.geoserver.org/geoserver/master/community-latest/geoserver-2.16-SNAPSHOT-wfs3-plugin.zip) [https://build.geoserver.org/geoserver/master/community-latest/geoserver-2.16-SNAPSHOT-wfs3-plugin.zip] community module. The source code for GeoServer and both modules above can be found at [GeoServer's GitHub account](https://github.com/geoserver/geoserver/tree/master/src) [https://github.com/geoserver/geoserver/tree/master/src].

interactive instruments GmbH

The provisioning of features and tiled feature data in VTPExt has been described in the section "[Producing Tiled Vector Data](#)" above.

Support for styles as additional resources has been added in VTPExt. [The OpenAPI Styles API](#) is used to create, update, fetch and delete style sheets. Only style sheets in the Mapbox Style language are supported in VTPExt. The style sheets are stored on the file system of the server.

[The OpenAPI Styles API](#) section includes a discussion about the design considerations and open issues.

interactive instruments also investigated whether tile and style resources could be useful for the API representation of the Web API. Unlike other OGC standards, WFS 3.0 not only focusses on data represented in JSON, eXtensible Markup Language (XML), databases, etc., but also on HyperText Markup Language (HTML) as an important representation of data. The conclusions were as follows:

- There is little utility in providing an HTML page for each tile because tiles are partitions of space to simplify the processing of the features by software, not by humans.
- For styles, there are multiple possibilities. A HTML representation could be a legend displaying the styling rules and symbols. Alternatively, it could even be an editor that allows to update the style (using the API), if the user has sufficient rights. (This capability was out-of-scope for the VTPExt.)
- Tiled feature data and styles can be used to provide an additional resource that provides an interactive map for each style. Links to the maps have been added to the HTML landing page of the dataset. The links are dynamically created from the set of available styles for the dataset.

Daraa

This is a test dataset for the OGC Vector Tiles Pilot. The data is OpenStreetMap data from the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA.

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

API Definition [OpenAPI 3.0](#)

- Maps**
- [night](#)
 - [overlay](#)
 - [topographic](#)

Figure 9. Screenshot of landing page for the Daraa dataset

The interactive map uses OpenLayers and fetches the Mapbox Vector Tiles from the server via the API. The map converts the Mapbox style sheet, which is also fetched from the server via the API, on-the-fly, to an OpenLayers style that is used to render the tiles.

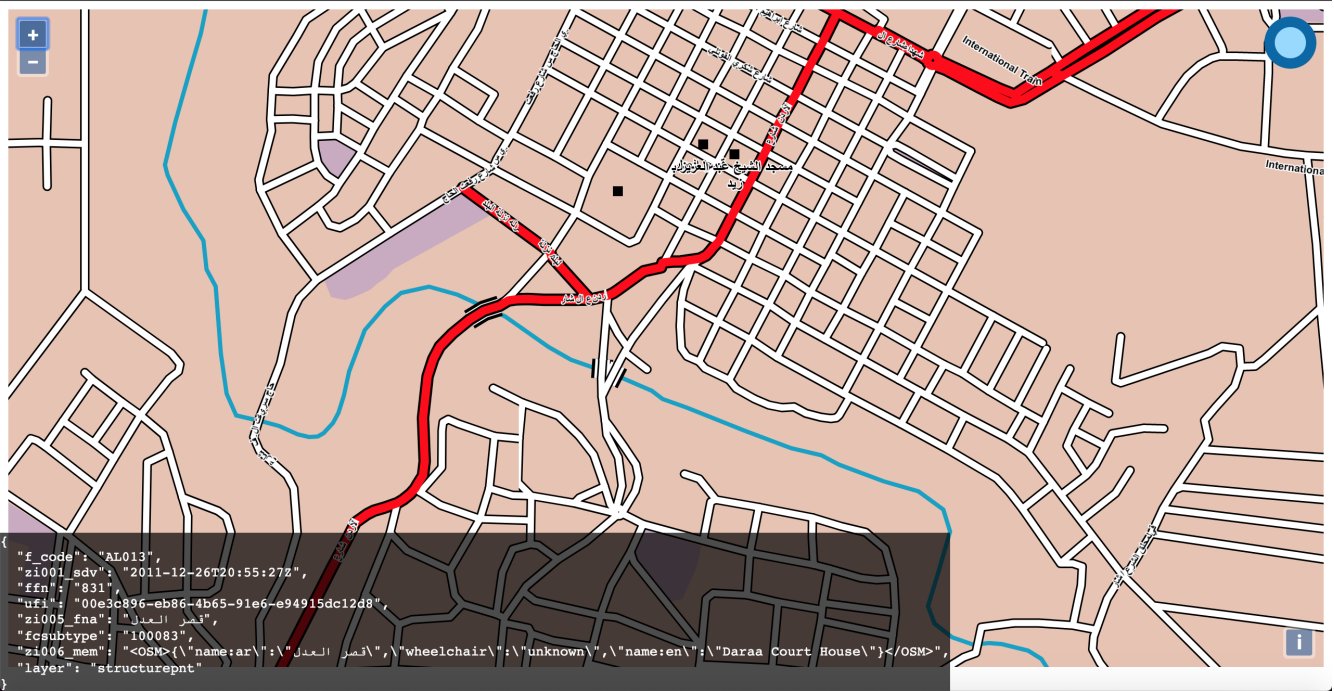


Figure 10. Map of Daraa dataset using the 'topographic' style

The map includes the capability to display or hide each layer of the map.

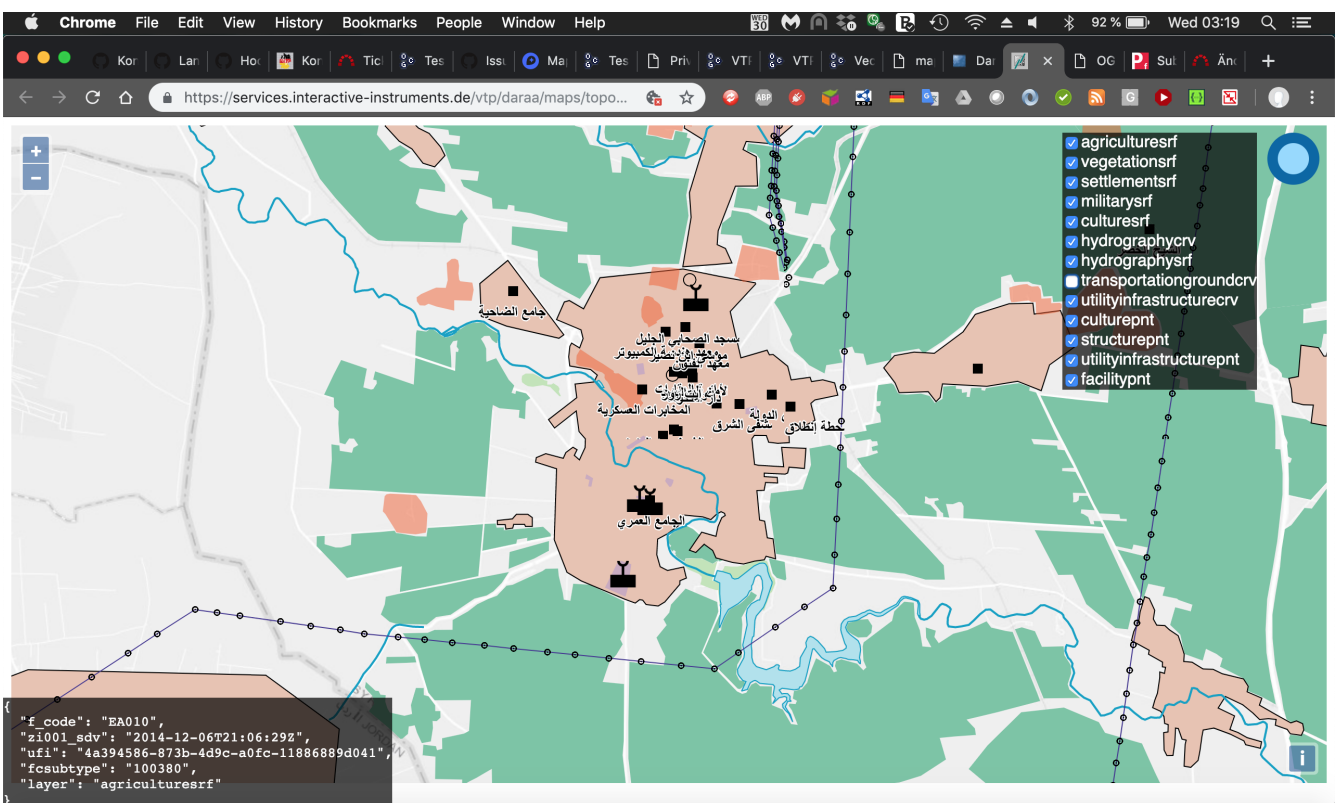


Figure 11. Map of Daraa dataset without the transportation features

Ecere

Ecere enhanced its GNOSIS Map Server with the capability to serve style sheets from its WFS3 service. For this initial implementation, only the styles end-point within a particular feature collection was added. Style sheets in SLD/SE, Mapbox GL styles, and GNOSIS Cartographic Map Style Sheets are made available for the VTP Daraa dataset.

The end-point for listing available styles for the Daraa2 meta-collection is available at:

<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles> <http://maps.ecere.com/hms/collections/vtp/Daraa2/styles?f=json> (JSON)

while specific styles and encodings are available at:

Night Style

<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles/night?f=mbstyle>
<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles/night?f=sld>
<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles/night?f=cmss>

Topographic Style

<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles/topographic?f=mbstyle>
<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles/topographic?f=sld>
<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles/topographic?f=cmss>

Overlay Style

<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles/overlay?f=mbstyle>
<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles/overlay?f=sld>
<http://maps.ecere.com/hms/collections/vtp/Daraa2/styles/overlay?f=cmss>

Styles are also accessible from within a sub-layer, e.g., the TransportationGround curves:

<http://maps.ecere.com/hms/collections/vtp/Daraa2/TransportationGroundCrv/styles>

However these style sheets currently simply link to the full style sheet. A future version will likely filter out rules, only keeping the styling rules pertaining to the selected sub-layer.

With the Ecere service serving a large organized collection of layers, a global styles list above the *collections* resource would require the concept of a *stylable layer set* to distinguish identical style names available for different set of layers served from the same end-point. This may be implemented in the future.

Another new capability which was implemented during the VTPExt was the generation of Mapbox Vector Tiles containing multiple layers. Those tiles are made available from:

<http://maps.ecere.com/hms/collections/vtp/Daraa2/tiles>

while a specific sub-layer is accessible from endpoints like the following:

<http://maps.ecere.com/hms/collections/vtp/Daraa2/TransportationGroundCrv/tiles>

See the Vector Tiles Pilot WFS3 Engineering report for more information on Ecere's WFS3 capability serving tiled vector feature data.

7.2.2. Web Map Tile Service (Mapping 1)

This implementation is based on [Mapping #1 \(WMTS Layer ≈ Mapbox Layer\)](#).

GeoSolutions

The GeoServer WMTS serves tiled feature data built with the same approach/architecture described in the GeoSolutions WFS server chapter, with the following interesting variations:

- The tile layer needs to be explicitly configured, along with the formats being produced/cached, allows usage of any available or user defined tileset
- Tiles can be either generated on request, if missing, or be pre-populated via a seeding procedure
- Unlike WFS 3.0, the WMTS protocol did not require any extension to serve Mapbox Vector Tiles, the existing support allows to simply specify them as a new output format.

The WMTS was also augmented with a styling API, implemented as a community module and made available on the GeoServer web site for anyone to use.

The WMTS styling module adds a few [ResourceURL](#) for each layer, providing access to styles usable with the given layer, e.g.:

```
<ResourceURL resourceType="defaultStyle" format="application/vnd.ogc.sld+xml"
template="https://backoffice-maps.geo-
solutions.it/geoserver/gwc/service/wmts/reststyles/layers/vtp:SettlementSrf/styles/set
tlementsrf_sld?f=application%2Fvnd.ogc.sld%2Bxml"/>
<ResourceURL resourceType="style" format="application/vnd.geoserver.mbstyle+json"
template="https://backoffice-maps.geo-
solutions.it/geoserver/gwc/service/wmts/reststyles/layers/vtp:SettlementSrf/styles/set
tlementsrf_mbstyle?f=application%2Fvnd.geoserver.mbstyle%2Bjson"/>
<ResourceURL resourceType="style" format="application/vnd.ogc.sld+xml"
template="https://backoffice-maps.geo-
solutions.it/geoserver/gwc/service/wmts/reststyles/layers/vtp:SettlementSrf/styles/set
tlementsrf_mbstyle?f=application%2Fvnd.ogc.sld%2Bxml"/>
<ResourceURL resourceType="style" format="application/vnd.geoserver.geocss+css"
template="https://backoffice-maps.geo-
solutions.it/geoserver/gwc/service/wmts/reststyles/layers/vtp:SettlementSrf/styles/set
tlementsrf_css?f=application%2Fvnd.geoserver.geocss%2Bcss"/>
<ResourceURL resourceType="style" format="application/vnd.ogc.sld+xml"
template="https://backoffice-maps.geo-
solutions.it/geoserver/gwc/service/wmts/reststyles/layers/vtp:SettlementSrf/styles/set
tlementsrf_css?f=application%2Fvnd.ogc.sld%2Bxml"/>
```

In the above block, the layer is associated with three different hand-written styles, [settlementsrf_sld](#), [settlementsrf_mbstyle](#), and [settlementsrf_css](#). The SLD style, [settlementsrf_sld](#), is offered as SLD only, while the for the MapBox and GeoCSS styles, [settlementsrf_mbstyle](#), and [settlementsrf_css](#) the capabilities document provides also alternate links, that transforms the style to SLD on the fly.

The above resources also support PUT and DELETE operations, allowing the client to modify, remove, and add styles associated to a particular layer.

Conceptually, the extension suffers from the following issues:

- The ResourceURL templates were meant for GET requests only. The capabilities document advertises neither support for the other operations nor the list of allowed content types. The client may perform an OPTIONS request to discover the other methods.
- Unlike WFS, there is no RESTful "dataset" level style management, as WMTS provides no service wide ability to advertise a [ResourceURL](#).

On the other side, WMTS was designed in GeoServer to also handle "layer groups", a special type of layer designed for basemaps that draws a list of basic layers in a given sequence, thus allowing a client to get multi-layer tiles. The styles associated to these layers contain an aggregation of all the style for all the layers in the group.

In order to deploy a GeoServer with vector tiles support one can download a [development version of GeoServer](#) [<http://geoserver.org/release/master/>], add the [vector tiles extension](#) [<https://docs.geoserver.org/latest/en/user/extensions/vectortiles/index.html>] and finally include the [WMTS styles](#) [<https://build.geoserver.org/geoserver/master/community-latest/geoserver-2.16-SNAPSHOT-wmts-styles-plugin.zip>] community module. The source code for GeoServer and both modules above can be found at [GeoServer's GitHub account](#) [<https://github.com/geoserver/geoserver/tree/master/src>].

CubeWerx

CubeWerx has provided a live WMTS at <https://tb14.cubewerx.com/cubewerx/cubeserv/vtext> and a static (filesystem-only) WMTS at <https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/WMTSCapabilities.xml>, both of which implement the proposed [WMTS 1.0 Styles API Profile Specification](#). For the purposes of demonstrating Mapping #1, both of these servers expose the following layers:

```
Daraa:AgricultureSrf
Daraa:CulturePnt
Daraa:CultureSrf
Daraa:FacilityPnt
Daraa:HydrographyCrv
Daraa:HydrographySrf
Daraa:MilitarySrf
Daraa:SettlementSrf
Daraa:StructurePnt
Daraa:TransportationGroundCrv
Daraa:UtilityInfrastructureCrv
Daraa:UtilityInfrastructurePnt
Daraa:VegetationSrf
```

Each of these layers is equipped with Night, Topographic, and Overlay styles, each of which style the single feature set corresponding to the layer. Tiles can be requested in MVT, GeoJSON, PNG, JPEG, or JOP (JPEG or PNG as appropriate). If one of the latter three formats is requested, the tile rendering is performed on the server side.

Individual style definitions can be requested or modified through endpoints of the form:

```
https://tb14.cubewerx.com/cubewerx/cubeserv/vtext/wmts/1.0.0/layers/{layer}/{style}.sld
https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/layers/{layer}/{style}.sld
```

as dictated by the style URL templates advertised in the servers' capabilities documents, e.g.,

```
https://tb14.cubewerx.com/cubewerx/cubeserv/vtext/wmts/1.0.0/layers/Daraa%3ATransportationGroundCrv/Night.sld
https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/layers/Daraa%3ATransportationGroundCrv/Night.sld
```

Style definitions can also be requested from the live WMTS through the KVP GetStyle operation, e.g.,

```
https://tb14.cubewerx.com/cubewerx/cubeserv/vtext?SERVICE=WMTS&VERSION=1.0.0&REQUEST=GetStyle&LAYER=Daraa%3ATransportationGroundCrv&STYLE=Night&FORMAT=application%2Fvnd.ogc.sld%2Bxml
```

Since this way of modeling styles, i.e., piecemeal per layer, is not aligned with the Mapbox model, participants have experimented with an additional set of endpoints that represent the definition of a single style that is defined across multiple layers. These endpoints are available on the live and static CubeWerx WMTSs at:

```
https://tb14.cubewerx.com/cubewerx/cubeserv/vtext/wmts/1.0.0/styles/{style}.sld
https://tb14.cubewerx.com/cubewerx/cubeserv/vtext/wmts/1.0.0/styles/{style}.json
https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/styles/{style}.sld
https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/styles/{style}.json
```

Such resources are represented in SLD as a set of one or more layers, each of which has exactly one style, all with the specified style ID. Unfortunately, this creates an awkward disconnect between the resources represented by these endpoints and the way styles are referenced in the rest of the WMTS API. For this reason, these endpoints were not included in the proposed [WMTS 1.0 Styles API Profile Specification](#). Modelling such endpoints in WMTS requires more research and consideration.

Ecere

Ecere enhanced its GNOSIS Map Server with the capability to serve style sheets from its WMTS service. The Ecere WMTS service currently only supports Key Value Pair (REST support is not yet implemented). For this initial implementation, only the requests for styles associated with a particular feature collection were added. Style sheets in SLD/SE, Mapbox GL styles, and GNOSIS Cartographic Map Style Sheets are made available for the VTP Daraa dataset.

The list of available styles is included as part of the WMTS capabilities document using the `<Style>` tag of a `<Layer>` element.

<http://maps.ecere.com/wmts?SERVICE=WMTS&REQUEST=GetCapabilities>

The request for a given style (GetStyles operation) for the Daraa2 meta-collection takes the form of:

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer={layer}&style={style}&format={format}>

The Daraa styles made available are accessible from:

Night Style

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2&style=night&format=mbstyle>

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2&style=night&format=sld>

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2&style=night&format=cmss>

Topographic Style

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2&style=topographic&format=mbstyle>

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2&style=topographic&format=sld>

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2&style=topographic&format=cmss>

Overlay Style

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2&style=overlay&format=mbstyle>

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2&style=overlay&format=sld>

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2&style=overlay&format=cmss>

Styles are also accessible with a sub-layer selected, e.g., the TransportationGround curves:

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetStyles&layer=vtp:Daraa2:TransportationGroundCrv&style=night&format=mbstyle>

However, these style sheets currently simply link to the full style sheet. A future version will likely filter out rules, only keeping the styling rules pertaining to the selected sub-layer.

Another new capability which was implemented during the VTPExt was the generation of Mapbox Vector Tiles containing multiple layers. Those tiles for the Daraa2 dataset are made available from:

<http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetTile&layer=vtp:Daraa2&tileMatrixSet={tileMatrixSet}&tileMatrix={tileMatrix}&tileRow={tileRow}&tileCol={tileCol}&format=mvt>

while a specific sub-layer is accessible from endpoints like the following:

```
http://maps.ecere.com/wmts?service=WMTS&version=1.0.0&request=GetTile&
layer=vtp:Daraa2:TransportationGroundCrv&tileMatrixSet={tileMatrixSet}&
tileMatrix={tileMatrix}&tileRow={tileRow}&tileCol={tileCol}&format=mvt
```

The convention of using the colon (:) separator to establish the relationship between meta-layer (e.g. Daraa2) and sub layer (e.g. TransportationGroundCrv) was adopted.

See the Vector Tiles Pilot WMTS Engineering report [5] for more information on Ecere's WMTS capability serving tiled vector feature data.

7.2.3. Web Map Tile Service (Mapping 2)

This implementation is based on [Mapping #2 \(WMTS Style ≈ Mapbox Style Sheet\)](#).

CubeWerx

To demonstrate Mapping #2, the live CubeWerx WMTS at <https://tb14.cubewerx.com/cubewerx/cubeserv/vtext>, and the static CubeWerx WMTS at <https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/WMTSCapabilities.xml>, both of which implement the proposed [WMTS 1.0 Styles API Profile Specification](#), have also exposed the following conglomerate layer:

Daraa

This layer is equipped with Night, Topographic, and Overlay styles, each of which style all 13 feature sets. Tiles can be requested in MVT, GeoJSON, PNG, JPEG, or JOP (JPEG or PNG as appropriate). If one of the latter three formats is requested, the tile rendering is performed on the server side.

A style definition for this layer can be requested or modified through endpoints of the form:

```
https://tb14.cubewerx.com/cubewerx/cubeserv/vtext/wmts/1.0.0/layers/Daraa/{style}.sld
https://tb14.cubewerx.com/cubewerx/cubeserv/vtext/wmts/1.0.0/layers/Daraa/{style}.json
https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/layers/Daraa/{style}.sld
https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/layers/Daraa/{style}.json
```

as dictated by the style URL templates advertised in the servers' capabilities documents, e.g.,

```
https://tb14.cubewerx.com/cubewerx/cubeserv/vtext/wmts/1.0.0/layers/Daraa/Night.sld
https://tb14.cubewerx.com/cubewerx/cubeserv/vtext/wmts/1.0.0/layers/Daraa/Night.json
https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/layers/Daraa/Night.sld
https://tb14.cubewerx.com/cubewerx/staticDaraaWmts/layers/Daraa/Night.json
```

Style definitions can also be requested from the live WMTS through the KVP GetStyle operation, e.g.,

<https://tb14.cubewerx.com/cubewerx/cubeserv/vtext?SERVICE=WMTS&VERSION=1.0.0&REQUEST=GetStyle&LAYER=Daraa&STYLE=Night&FORMAT=application%2Fvnd.ogc.sld%2Bxml>

7.3. GeoPackage Provisioning

The data flow for GeoPackage provisioning (label 6 in [Figure 1](#)) is illustrated in [Figure 12](#). This data flow is agnostic to implementation details such as data formats and styling approaches.

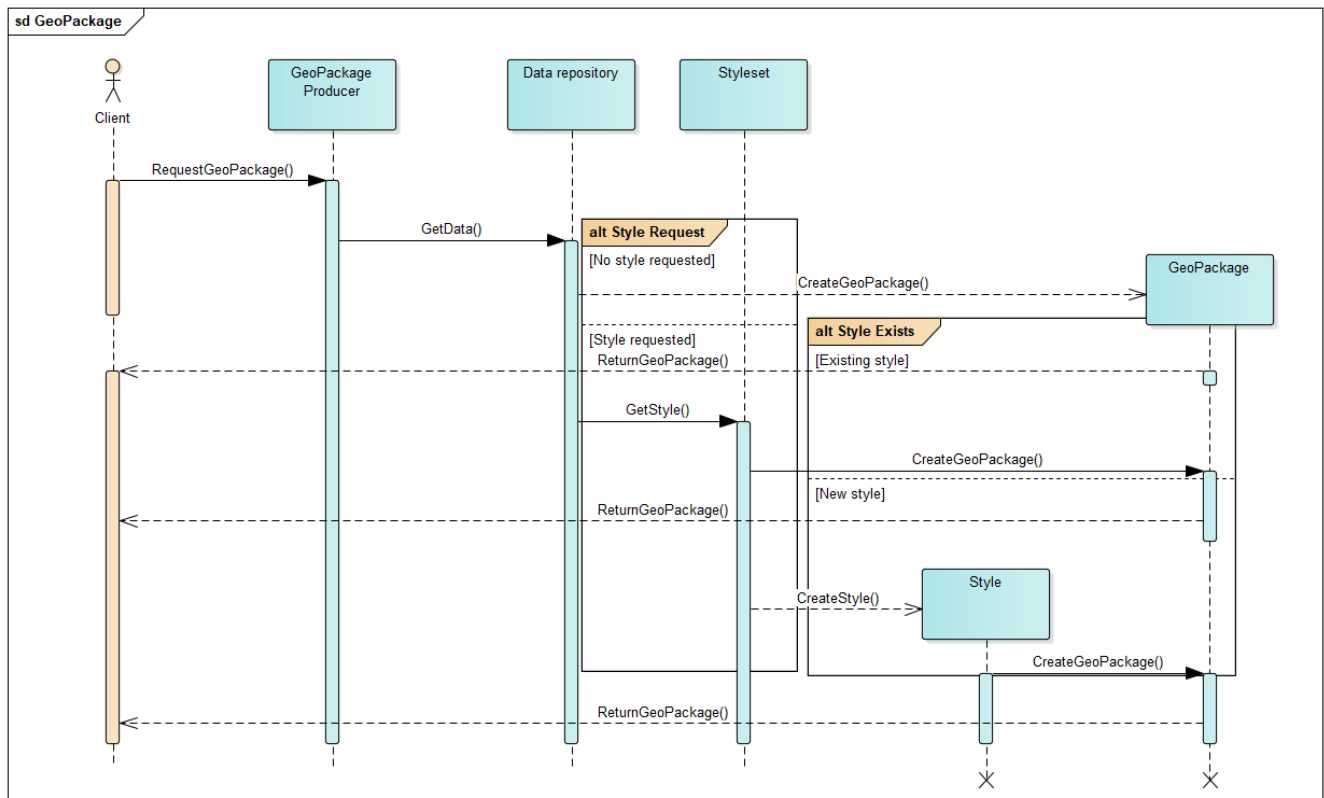


Figure 12. GeoPackage Data Sequence

7.3.1. Producing GeoPackages

In the previous ER, a set of extensions was proposed to support tiled feature data in a GeoPackage. These extensions, with minor modifications, are described in [Tiled Feature Data Extension](#), [GeoPackage Mapbox Vector Tiles Extension](#), and [GeoPackage GeoJSON Vector Tiles Extension](#). Draft specifications for these extensions are in [GeoPackage Extensions Requirements \(Normative\)](#).

Compusult

The Compusult GeoPackage Producer runs as a web-browser based application, as well as being accessible via an OGC Web Processing Service (WPS) instance. The GeoPackage Producer supports producing Mapbox Vector Tile and GeoJSON tile based GeoPackages in a user selected projection system and an uploaded vector source (Shapefile(s), GeoDatabase, SQLiteDB, etc.). The producer has the ability to convert all feature types into a single Mapbox Vector Tile or produce a single Vector Tile layer for each feature type.

Uploaded content may contain SLD or Mapbox Style documents which are mapped to appropriate

layers using the Tiled Feature Data extension along with the GeoPackage styles extension. Stylesets are determined based on folder structure within an uploaded source. If required for no bandwidth environments, external symbols are mapped to the `gpkext_symbols` table, and URN references are updated in the corresponding style document.

Furthermore, the producer also has the ability to embed feature attributes into the Mapbox Vector Tile or to use a [GeoPackage Related Tables Extension](https://www.geopackage.org/18-000.html) [https://www.geopackage.org/18-000.html] to produce attributes tables and appropriate mappings for optimal storage. The result of the Vector Tile Extension is a GeoPackage that is compliant to specifications of the National System for Geospatial Intelligence (NSG).

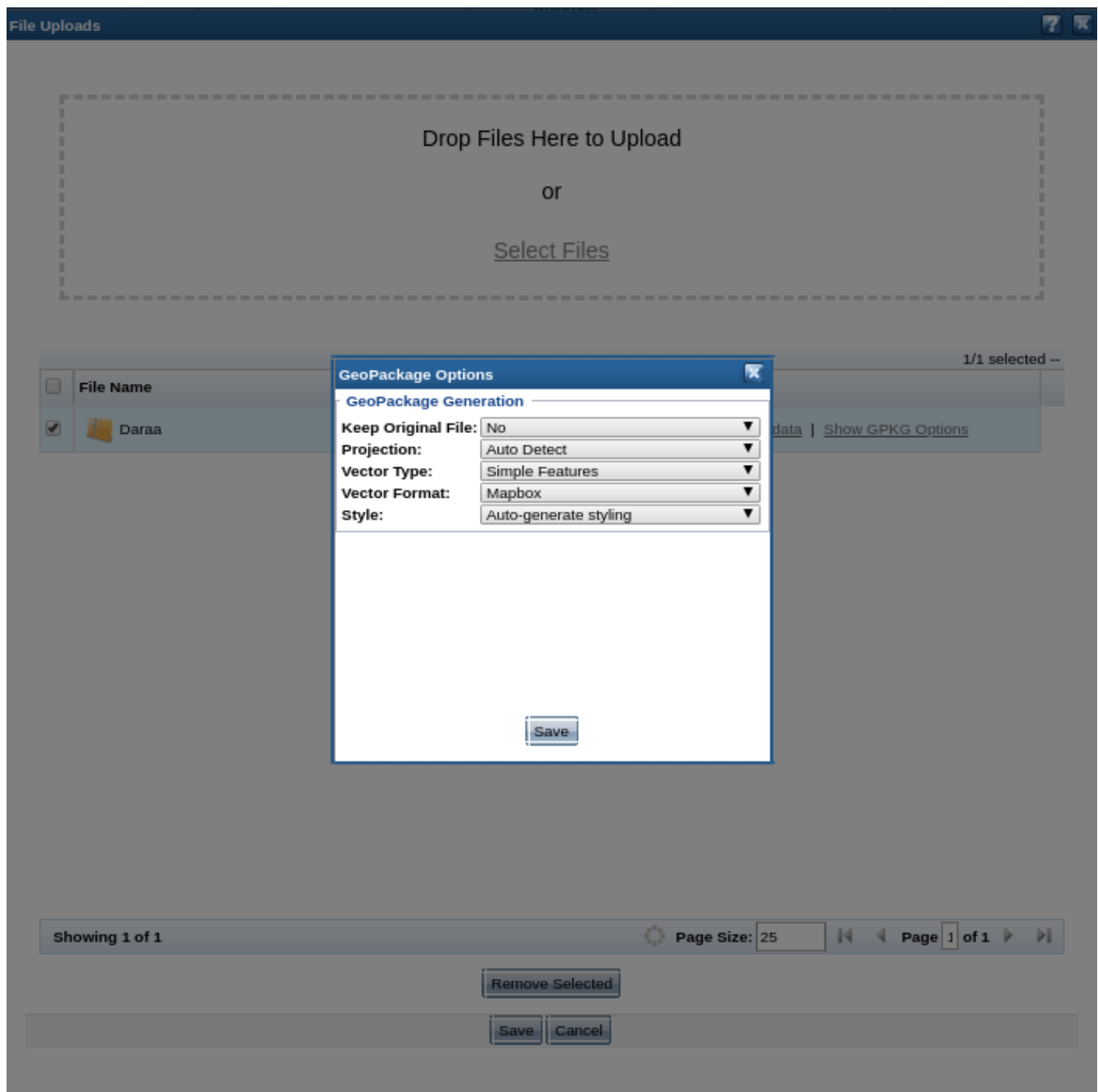


Figure 13. Compusult GeoPackage Producer Settings

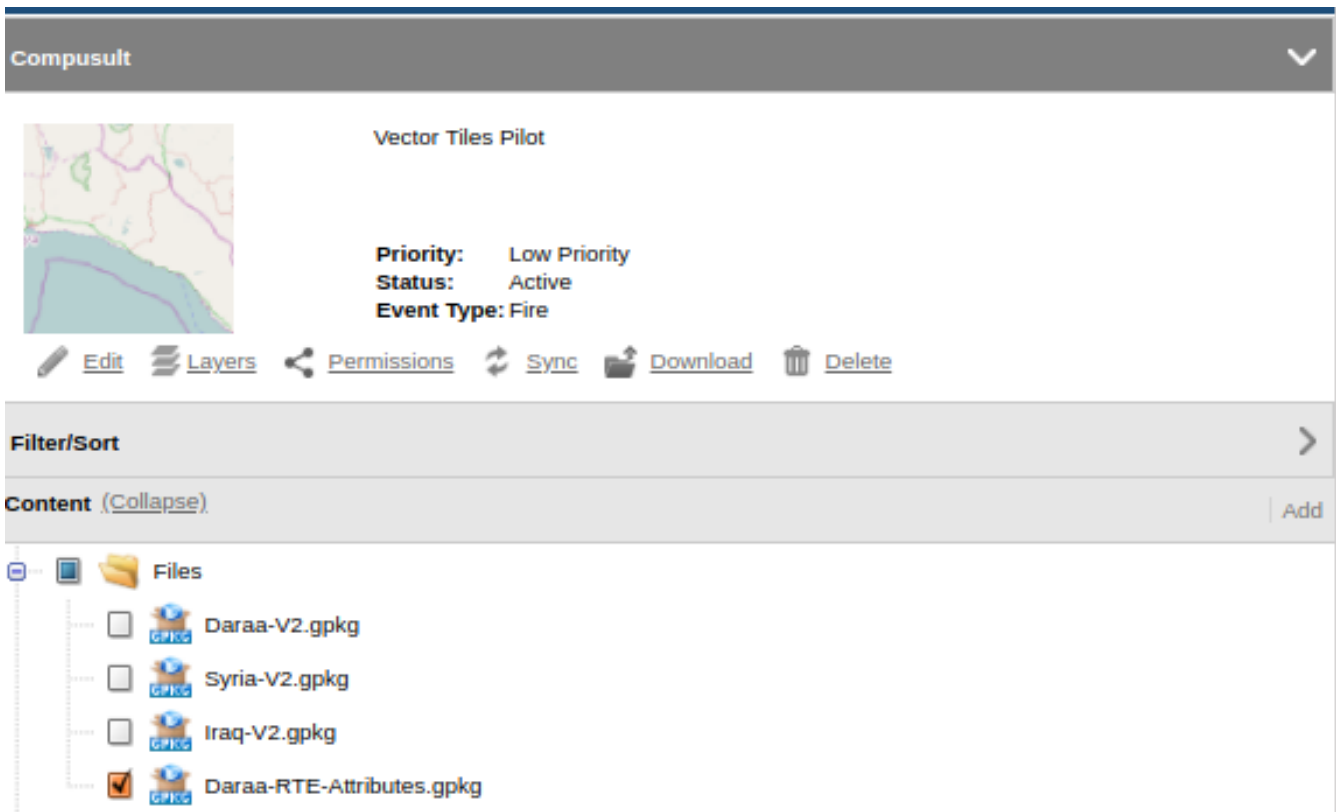


Figure 14. CompuSult GeoPackage Producer Result

CubeWerx

The CubeWerx GeoPackage producer is an administrator-level command-line tool. It can generate GeoPackages of MVT, GeoJSON, or PNG tiles, and has been augmented to support the [Tile Feature Data](#), [Mapbox Vector Tiles](#) and [GeoPackage Styles](#) extensions. (Currently, though, the contents of the `gpkgext_stylesheets` and `gpkgext_symbols` tables need to be filled in manually.)

Ecere

The Ecere GeoPackage producer is embedded within Ecere's GNOSIS Cartographer GIS tool. The GeoPackage producer was updated to match the minor changes to the emerging tiled feature data extension specifications. A set of layers can be selected for inclusion within the GeoPackage, and multiple options are available for generating different types of GeoPackages. These options include the following:

- whether to have attributes embedded within the tiles, or in attributes tables
- whether to generate tiles containing multiple layers, or individual tile sets for each layer
- tile format, including GNOSIS Map Tiles, Mapbox Vector Tiles, GeoJSON, GeoECON, and GML
- tiling scheme

The ability to include style sheets within the GeoPackage was improved upon to conform to the new styling extension agreed upon during this pilot phase (see section below for more details).

See the [Vector Tiles Pilot GeoPackage Engineering report \[6\]](#) for more information on Ecere's vector tiles GeoPackage producer.

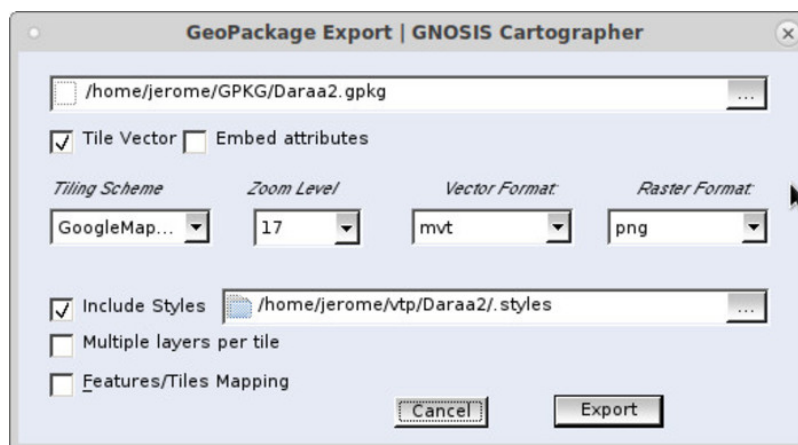


Figure 15. Exporting GeoPackages in Ecere's GNOSIS Cartographer

7.3.2. Supporting Attributes in GeoPackage

By default, GeoPackage clients should retrieve feature attributes directly from the vector tiles. This minimizes the burden on GeoPackage producers because the vector tiles do not need to be modified in any way. There are limitations to this approach:

- There is potentially a storage requirement cost because attributes are duplicated across tiles. This cost can manifest itself two ways:
 - when features appear in multiple zoom levels (this cost increases linearly based on the tile pyramid depth)
 - when features span multiple tiles (this cost is irregular depending on how many large features are present)
- Neither JSON nor Google Protocol Buffers support the querying that would be used to find the attributes for a particular feature. Either the data must be stored in memory or the entire vector tile must be scanned each time. This will scale poorly in many operational settings.
- It is not realistic to perform an attribute query and find the set of matching features because of the aforementioned lack of querying and because the results may span an unknown number of vector tiles. This will not scale to a non-trivial number of vector tiles.

In response, participants agreed to add a `attributes_table_name` column to `gpkgext_tfd_layers`. When a client sees a non-null value in this table, it should ignore the attributes in the vector tiles and instead use the attributes table.

Analysis

In general, embedded attribute GeoPackages are four times as large as their counterparts. This factor increases exponentially as tile matrix levels increase.

Heterogeneous Attribute Schemas

Neither GeoJSON nor MVT are prescriptive regarding attributes. Heterogeneous attribute schemas cannot be translated directly into relational tables. Tile sets that contain multiple layers are potentially a problem because they are more likely to have heterogeneous attribute schemas. However, since the GeoPackage producer attempting to use this attribute mechanism would have to modify the vector tiles anyway, it is reasonable to force the producer to split the tile set into

multiple tile sets to ensure that each tile set has a homogeneous schema. Possible approaches to mitigate this issue include the following:

- Make the heterogeneous schema homogeneous by combining the schemas together. Of course, this would lead to a lot of NULL values in the attributes table.
- Store attributes with tables of key/value pairs, much like how MVTs store attributes internally, with the attributes table referencing occurrences of key/values combinations. This would have the added advantage of greatly reducing storage overhead of duplicate values (particularly relevant for repeating character strings), but it would be more complicated to query.

Attributes Table Name

The `attributes_table_name` column in `gpkgext_tfd_layers` represents a set of tradeoffs.

One benefit of this approach is that it supports multiple layers within the same tileset even if the layers have independent attribute schemas. As indicated in [Displaying Tiled Feature Data Analysis](#), it is preferable to minimize the number of tilesets because that simplifies the number of drawing operations, a critical scalability point in mobile clients. However, there may be scenarios where it is be more scalable to keep each layer in its own tileset, particularly if particular views only require a subset of the available layers.

Potential drawbacks include the following:

- When this mechanism is in use, a client must discover the name of the primary key in the attributes table to correlate the features with the attributes. This is done through inspection of the attributes table schema. There is no requirement for an attributes table to have a primary key column because attributes may be also stored in views and the concept of primary keys for views does not exist in SQLite. The GeoPackage SWG recommends that attributes tables have primary keys and that if views are used instead of tables, that the first column of the view contains unique numeric values so that it can function like a primary key.
- This approach creates a dual logical path scenario. A client has to look for the attributes in two places, the vector tiles and the specified attributes table. GeoPackage SWG members have declared this to be an anti-pattern and have discouraged this approach in other areas. Their preference is for there to be a single logical path that is discoverable when the GeoPackage is loaded. In this case, clients that do not support this extension will ignore it and use the default.

Many Features to Many Tiles Extension (original Attributes Extension)

A *many features to many tiles extension* (originally named *Tiled Feature Data Attributes Extension*) was originally proposed during the VTP and documented in the GeoPackage Extensions ER, but was not used during the VTPEExt. This extension provided a way to establish a relationship between multiple tiles and multiple features, leveraging the Related Tables Extension. As a side advantage, this provided a way to link an attributes table with a tileset, using the `related_table_name` field of the `gpkgext_relations` table to locate the attributes table for a tile set. However, this did not support the use case of a single tileset containing multiple layers together with one attributes table per layer, as the `gpkgext_relations` table had no way to identify a specific layer. For this reason, the `attributes_table_name` column of `gpkgext_tfd_layers` approach was chosen instead to identify a table of attributes, supporting the queryable attributes use case.

This extension can still be considered for its ability to indicate which features are in which tiles, or which tiles contain certain features, by its mapping table specified by the Related Tables Extension. If such an extension is adopted, note that the `attributes_table_name` column of `gpkgext_tfd_layers` would still be useful together with this extension to identify which mapping table to use for which layer in the use case of a single tile set (with multiple layers) and multiple attributes table (one per layer). Without this, a client would not automatically know which tiles to search to find the geometries for specific features. This would lead to scalability problems if the extents for the query are not known.

A proposed alternative to this extension would be storing the geospatial extent of the feature in the attributes table, potentially as an R-tree entry additionally offering spatial indexing capability for attributes queries. This would directly provide the extent of such a feature (e.g., for 'zoom to extent' capability), as well as greatly restrict the number of tiles to examine to retrieve the geometry for that feature.

Compusult

The Compusult GeoPackage producer has the ability to embed feature attributes into its Tiled Feature Data or to use a [GeoPackage Related Tables Extension](https://www.geopackage.org/18-000.html) [https://www.geopackage.org/18-000.html] to produce attributes tables and appropriate mappings for optimal storage. When attributes are not embedded, the `attributes_table_name` is specified to ensure optimal compatibility when dealing with Tiled Feature Data containing more than one feature type. The Related Tables Extension is preferred to allow a client to query the features associated with a tile without ever having to inspect the tiles content. The GeoPackage producer also contains logic to allow a client to remove empty placeholder values that frequently appear in vector feature data such as '-999999' 'NULL' etc.

Ecere

The Ecere GeoPackage producer can either embed attributes within the tiles or generate an attributes table. The attributes table is identified by the `attributes_table_name` field of the `gpkgext_tfd_layers` table. A future version will probably store the extent of individual features to facilitate spatial queries, potentially conforming with the existing GeoPackage R-tree extension. Support for heterogeneous attributes stored using keys and values tables is also a capability being considered, which would be particularly useful for storing full OpenStreetMap data. The *many features to many tiles extension* is not currently supported but could eventually be optionally supported if it sees adoption.

7.3.3. Supporting Style Sheets in GeoPackage

A table is needed to store stylesheets in a GeoPackage. This table must also store style sets and options and formats for those style sets. The table design is presented in [Stylesheet Table Definition](#).

Analysis

A table was proposed in the GeoPackage Extensions ER, but that table was not sufficient to meet the needs identified in the CONOPS. There were some minor recommendations from the first proposal for this table:

- add a `description` column to provide some context for the row beyond the `styles_set` and `option`

columns which could be cryptic

- use the name `stylesheet` for the column containing the actual style BLOB instead of `data` or `style`.

In addition, a participant proposed a `layer` column for `gpkgext_stylesheets` that would correlate a stylesheet to the layer it would apply to. The purpose of this column is to declare what layer this style can be used for. While this may be handy as part of a system that solely uses tiled feature data, it would not be applicable to ordinary feature tables that also would benefit from a styling capability. Because of the limited utility, it is unlikely that this approach would be accepted by the GeoPackage SWG. Since the data manager would have the information needed to populate this column when producing the GeoPackage, the offering mechanism described below would work just as well.

NOTE

This ER also presents a proposed table for storing `symbols`, but this topic was not a focus of this pilot project.

Compusult

The Compusult GeoPackage producer scans the uploaded source content locating SLD or Mapbox Style documents which are mapped to appropriate layers using the Tiled Feature Data extension along with the [GeoPackage Styles Extension](#). A Tiled Feature Data layer is mapped to an SLD layer if any style has a FeatureTypeName matching the source vector data. Stylesets are determined based on folder structure within an uploaded source. If required for no bandwidth environments, external symbols are mapped to the `gpkext_symbols` table, and URN references are updated in the corresponding style document. In the future, a generic style data-store will be used to create/select feature type styles, allowing the client to choose to produce SLD or Mapbox style.

CubeWerx

CubeWerx has prepared two GeoPackages for the VT-Pilot Extension TIEs, with fully-populated `gpkgext_stylesheets` and `gpkgext_symbols` tables. One of these GeoPackages formulated its SLD styles as per [Mapping #1](#), while the other formulated its SLD styles as per [Mapping #2](#). The Mapbox styles are the same in each of the GeoPackages. CubeWerx considers it likely that Mapping #1 will be the preferred mapping for GeoPackages (at least for the SLD formulation), despite the fact that Mapping #2 is likely to be the preferred mapping for the WMTS API.

Rather than having the SLD encoding of the styles reference graphic symbols on an external web server, which would violate the principle that GeoPackages should be self-contained, the symbols in the `gpkgext_symbols` table are referenced by their `symbol_id`.

Ecere

The styles definitions included in the `gpkgext_stylesheets` by Ecere's GeoPackage producer are organized by stylable layer set and style, while supporting different encodings for the same style definition. The GeoPackages produced included SLD/SE, Mapbox GL Styles, and GNOSIS Cartographic Style Sheets. The process is currently driven by pre-generated styles documents, but will eventually be fully integrated with GNOSIS Cartographer's styling system, translating styles descriptions on-the-fly to the supported formats. Additionally, the ability to embed symbols (in a

`gpkgext_symbols` table) that can be referenced by the style sheet was added, allowing these GeoPackages to be fully self-sufficient.

7.3.4. Deploying GeoPackages

In this project, most implementers deployed GeoPackages manually to the target platform.

Compusult

Compusult deployed GeoPackages by adding them to a 'Portfolio' in its COTS software, or producing one from uploaded content. Compusult's GOMobile Desktop/Android client is able to search, discover, and add this content for rendering and analysis. GOMobile also allows users to upload local GeoPackages or to local them on the root of its device(Android). Compusult COTS software also has the ability to provide WMS/WMTS services based on uploaded/created GeoPackages using the internal tiled or simple feature data and its existing GeoPackage rendering client.

7.4. Integrated Clients

7.4.1. Binding to a WMTS, WFS and/or GeoPackage

Compusult (WMTS Client)

The Compusult WMTS client binds to a WMTS services using a publishing service to a Catalogue Service for the Web (CSW) client. Published services are exposed in the GOMobile client to be rendered. The WMTS client was updated to support GetStyles KVP operations as well as style Resource URL templates to support the retrieval of layer styles.

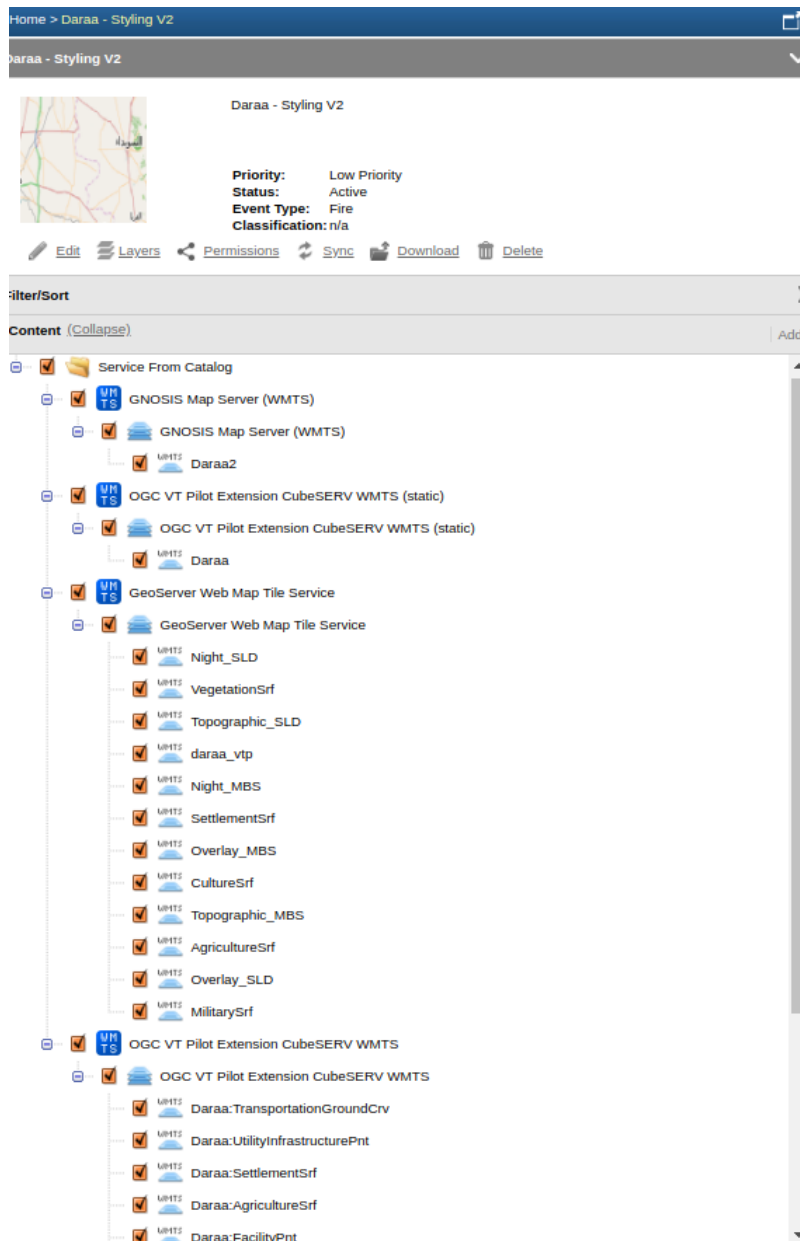


Figure 16. Compusult CSW Binding - WMTS Services

Compusult (GeoPackage Client)

The Compusult GeoPackage client binds to a GeoPackage in multiple ways. A client can upload the GeoPackage directly to GOMobile, or a GeoPackage can be uploaded to Compusult COTS software where it is added to the CSW and exposed as a raw file or a WMS/WMTS service based on its content. GOMobile accesses the services through the CSW and renders the content.

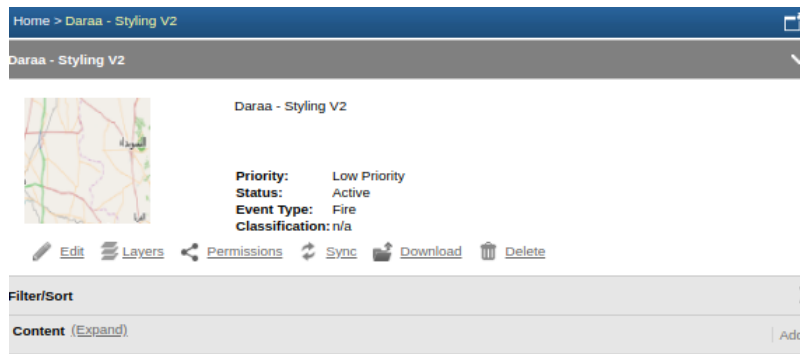


Figure 17. Compusult CSW Binding - GeoPackage Services

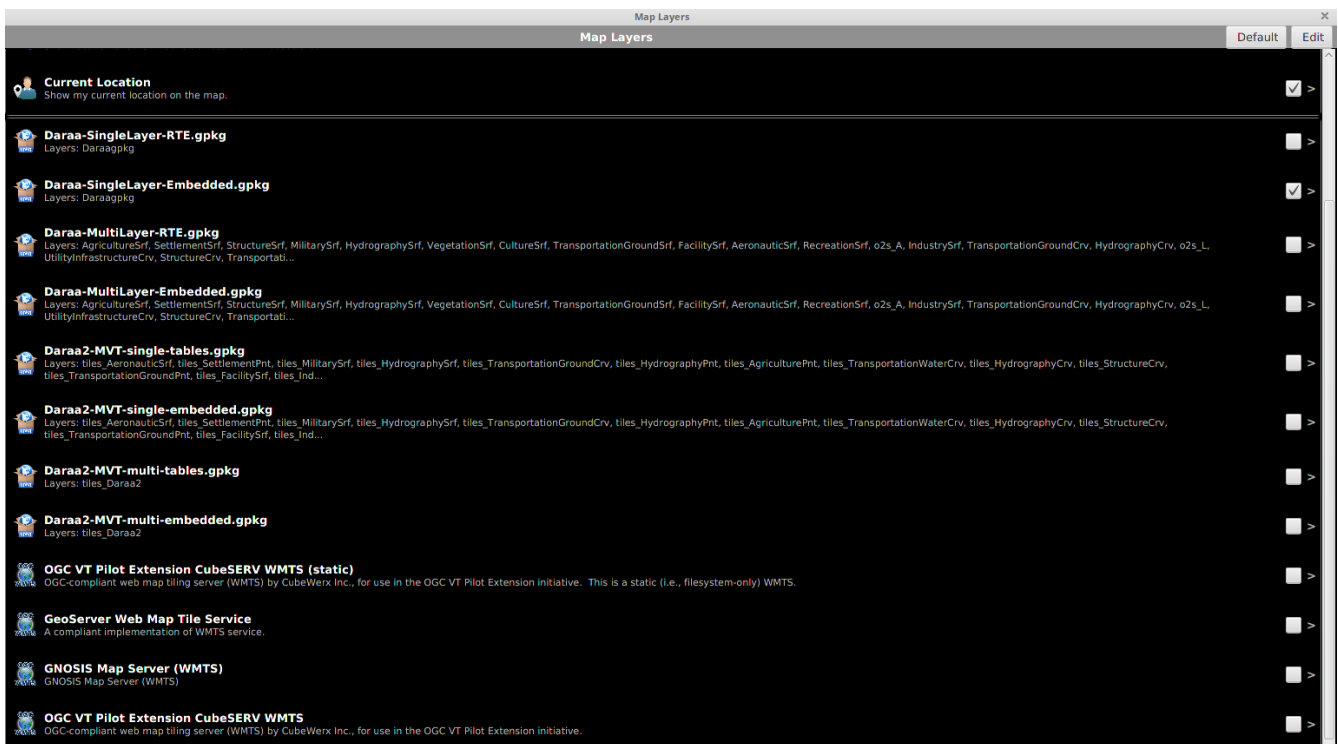


Figure 18. Compusult GOMobile Binding

Ecere (WFS / WMTS / GeoPackage Client)

Ecere used the same client, GNOSIS Cartographer, for all visualization experiments. As with any other supported geospatial data source, the binding is done by pointing the client to a WFS or WMTS service, or GeoPackage data source by specifying a URL, file, or directory path, from the Map Library's **Add...** button. The client will automatically recognize any supported geospatial data source and make it available for visualization. A new, consistent interface was added to present the list of default styles made available together with the data, regardless of the source type. Selecting such a style will automatically trigger its use for visualization. Multiple data sources can be visualized together, regardless of their origin. Although the exact same GNOSIS functionality demonstrated on the desktop is also available for the Web and Mobile platforms, there was no time in this pilot phase to demonstrate those capabilities on those other platforms.

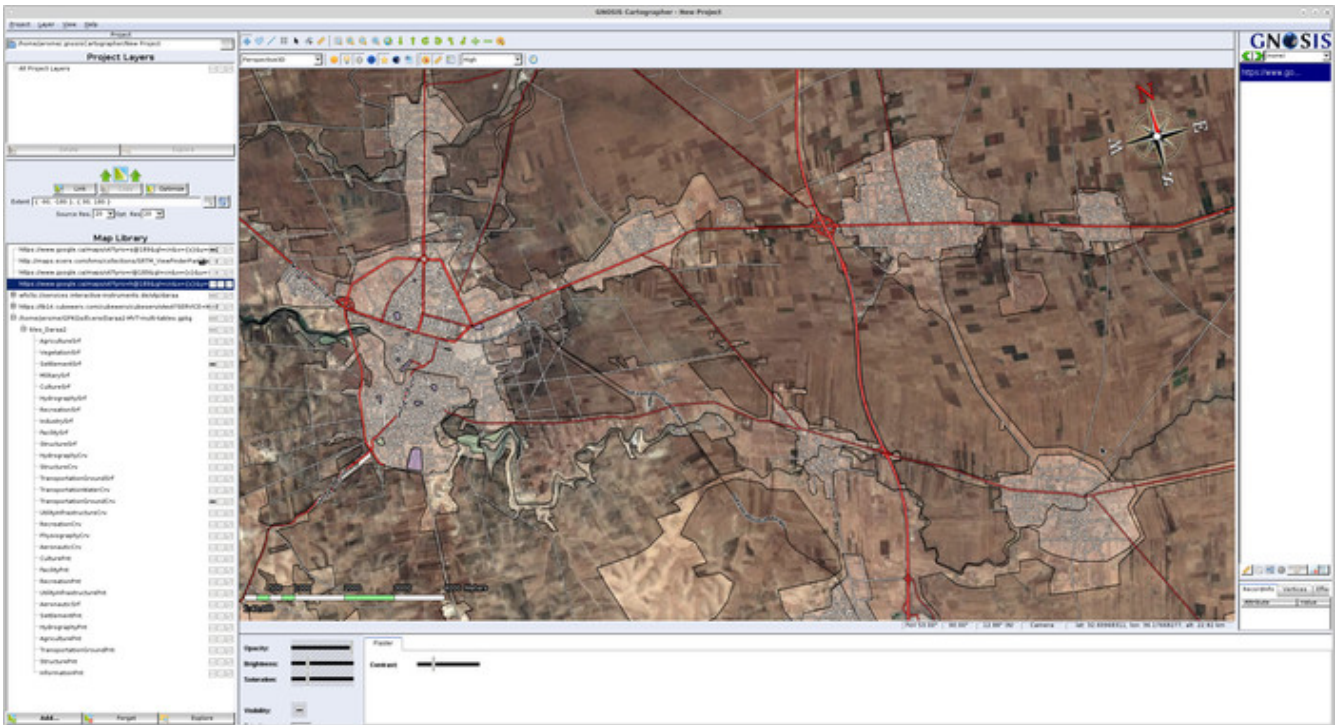


Figure 19. Ecere's GNOSIS Cartographer Visualizing multiple types of data sources

7.4.2. Requesting Tiled Feature Data from a WMTS

The WMTS Feature Tile extension allows for selecting tiled feature data and their associated styles as shown in Figure 20. This is by far the most simple of the extensions, as the existing WMTS specification allows for styles to be specified using the existing style parameter.

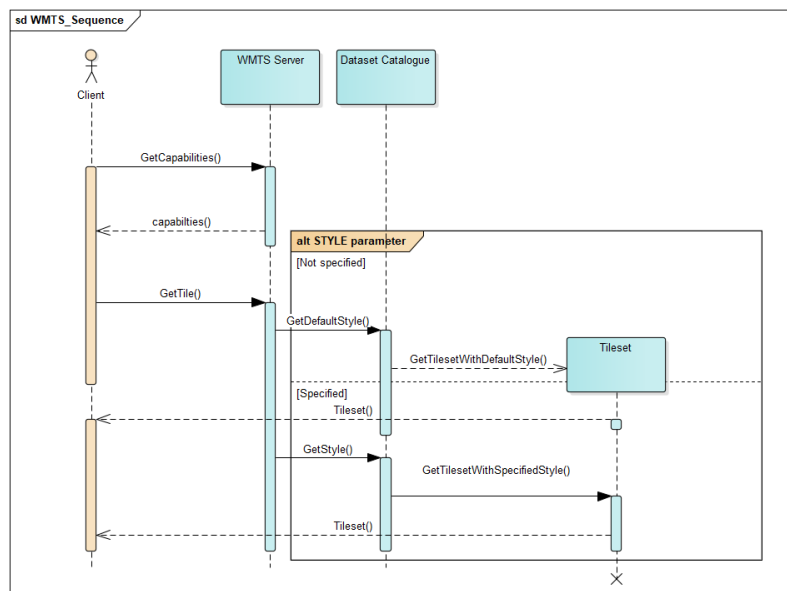


Figure 20. WMTS Style Sequence Diagram

GeoSolutions WMTS Simple Client

The application provides a simple environment to test style conversions client side (SLD to OpenLayers Style and MBStyle to OpenLayers Style) and get an overview of the editing workflow using WMTS vector tiles served via [GeoServer](http://geoserver.org/) [http://geoserver.org/].

The demo application, built with the [MapStore](https://mapstore.geo-solutions.it/mapstore/#/) [https://mapstore.geo-solutions.it/mapstore/#/] framework,

uses [OpenLayers](https://openlayers.org/) [https://openlayers.org/] as map renderer.

The initial configuration is stored in a JSON file listing layers and available styles, after the initialization, all style bodies are loaded and the application requests vector tiles via WMTS GetTile defined in the configuration file.

The application lists 3 variations of the same style Topographic, Night and Overlay.

link to demo repository: <https://github.com/geosolutions-it/ogc-vector-tiles-vtp/tree/master/MapStoreStyle>

link to live demo: <http://vtp2018.s3-eu-west-1.amazonaws.com/vtpext-wmts.html#/>

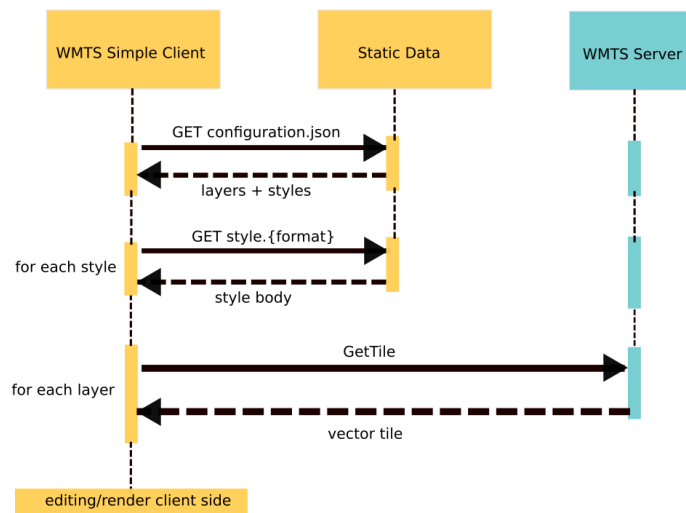


Figure 21. WMTS - Client Sequence Diagram

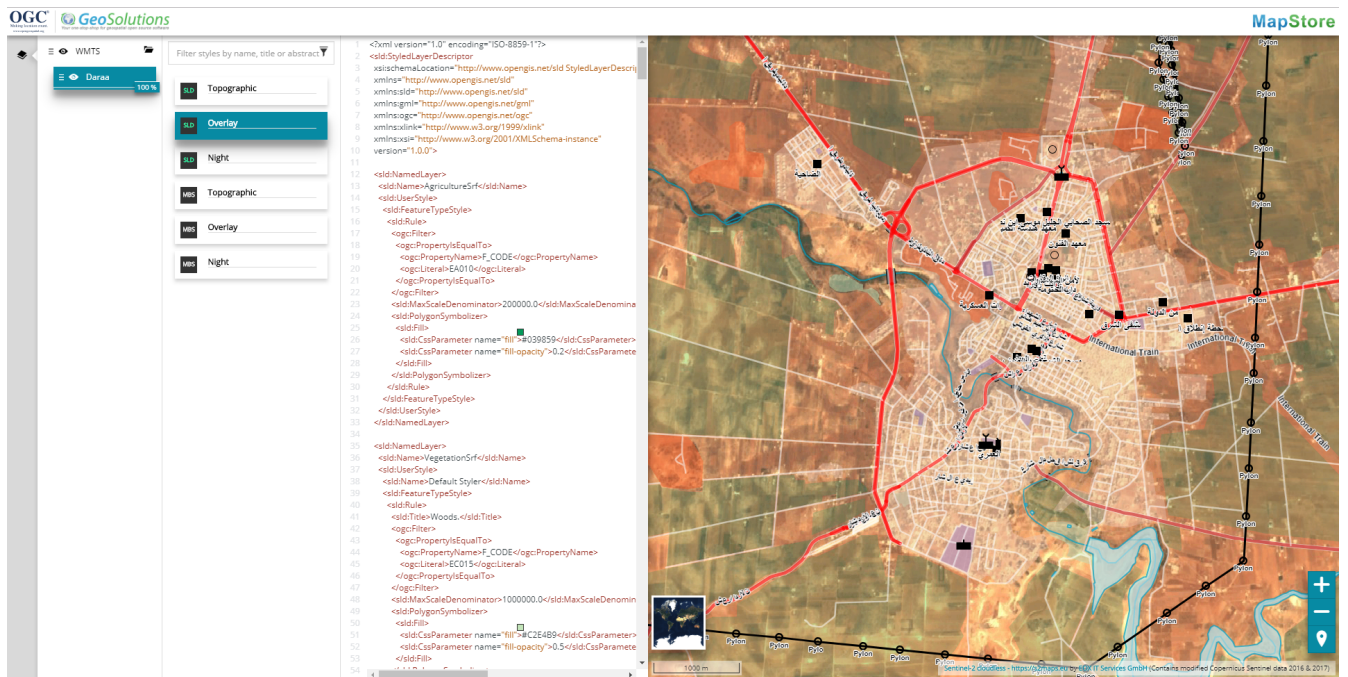


Figure 22. MapStore Client is Rendering Overlay Style in SLD Language with WMTS Vector Tiles

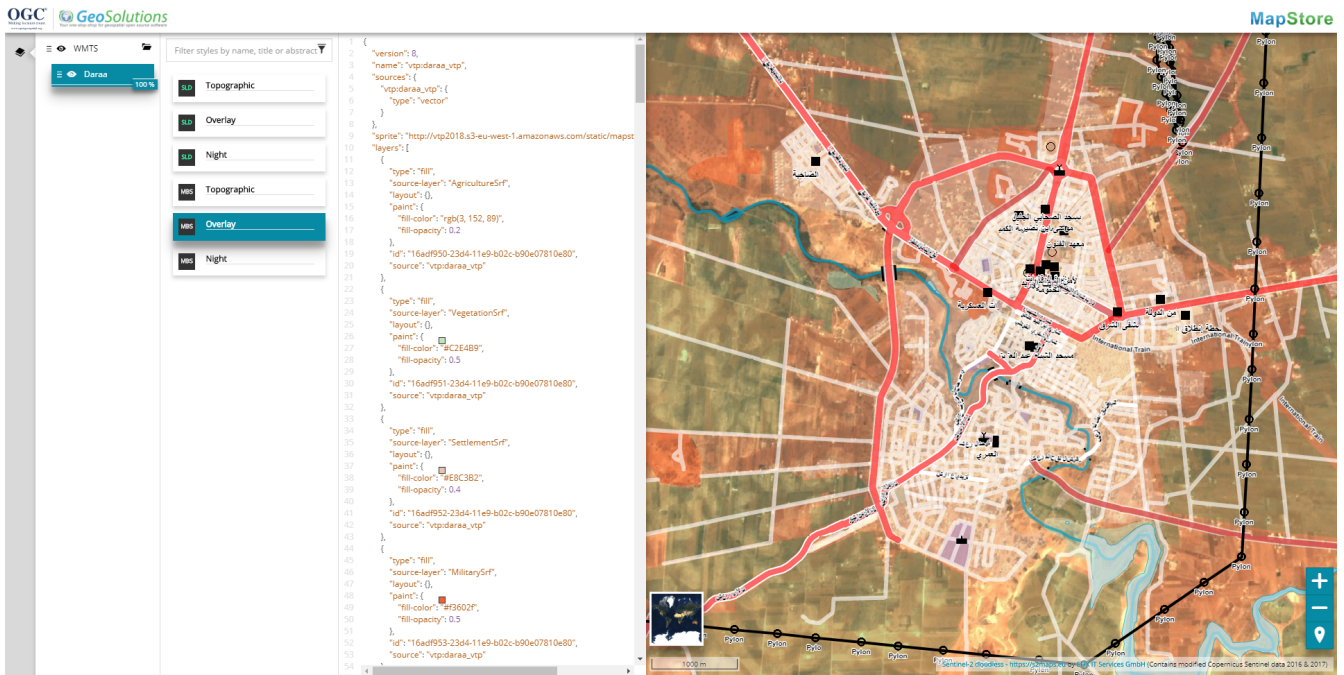


Figure 23. MapStore Client is Rendering Overlay Style in Mapbox Style Language with WMTS Vector Tiles

CompuSult WMTS Client

After binding with WMTS services, the CompuSult GOMobile client requests tiled feature data based on a services WMTS Capabilities Document. KVP and RESTful GetTile operations are supported along with tile formats of `application/vnd.mapbox-vector-tile`, `application/vnd.geo+json`, and other accepted aliases. If the WMTS service supports the GetStyles KVP operation or has style type Resource Templates the WMTS client will request a client specified style and format to render the vector content. Both SLD and Mapbox Style documents are supported, and can be applied to either GeoJSON or MVT tiles. The CubeWerx static WMTS services use default styling to render its MVT content and the standard WMTS 1.3.0 protocol to access pre-rendered tiled feature data in the PNG format.

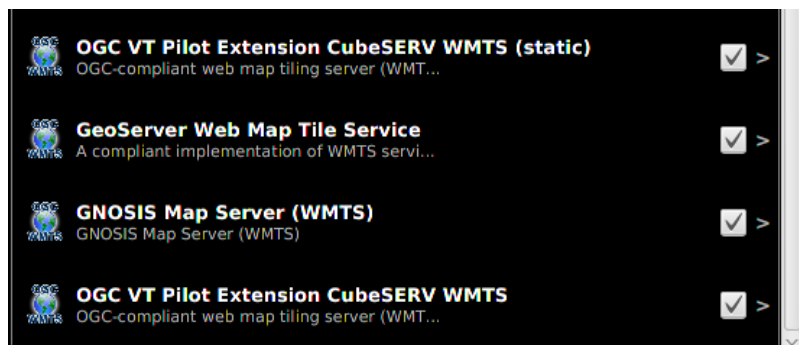


Figure 24. CompuSult GOMobile WMTS Services

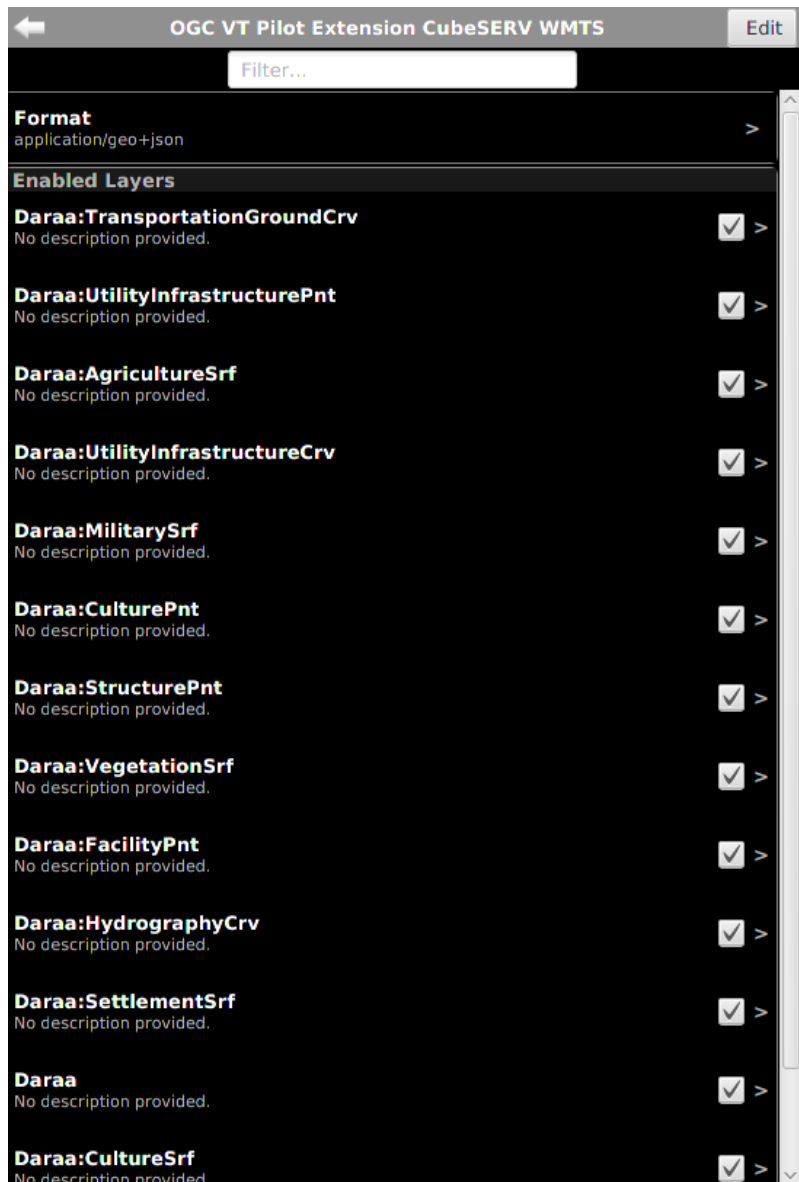


Figure 25. Compusult GOMobile WMTS Layer Selection

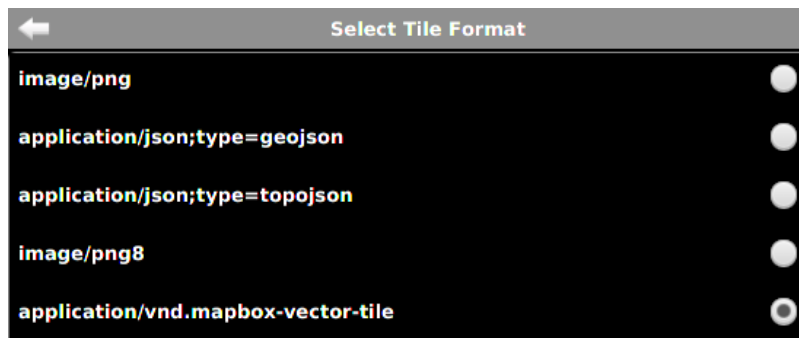


Figure 26. Compusult GOMobile WMTS Tile Format Selection



Figure 27. Compusult GOMobile WMTS Style Selection

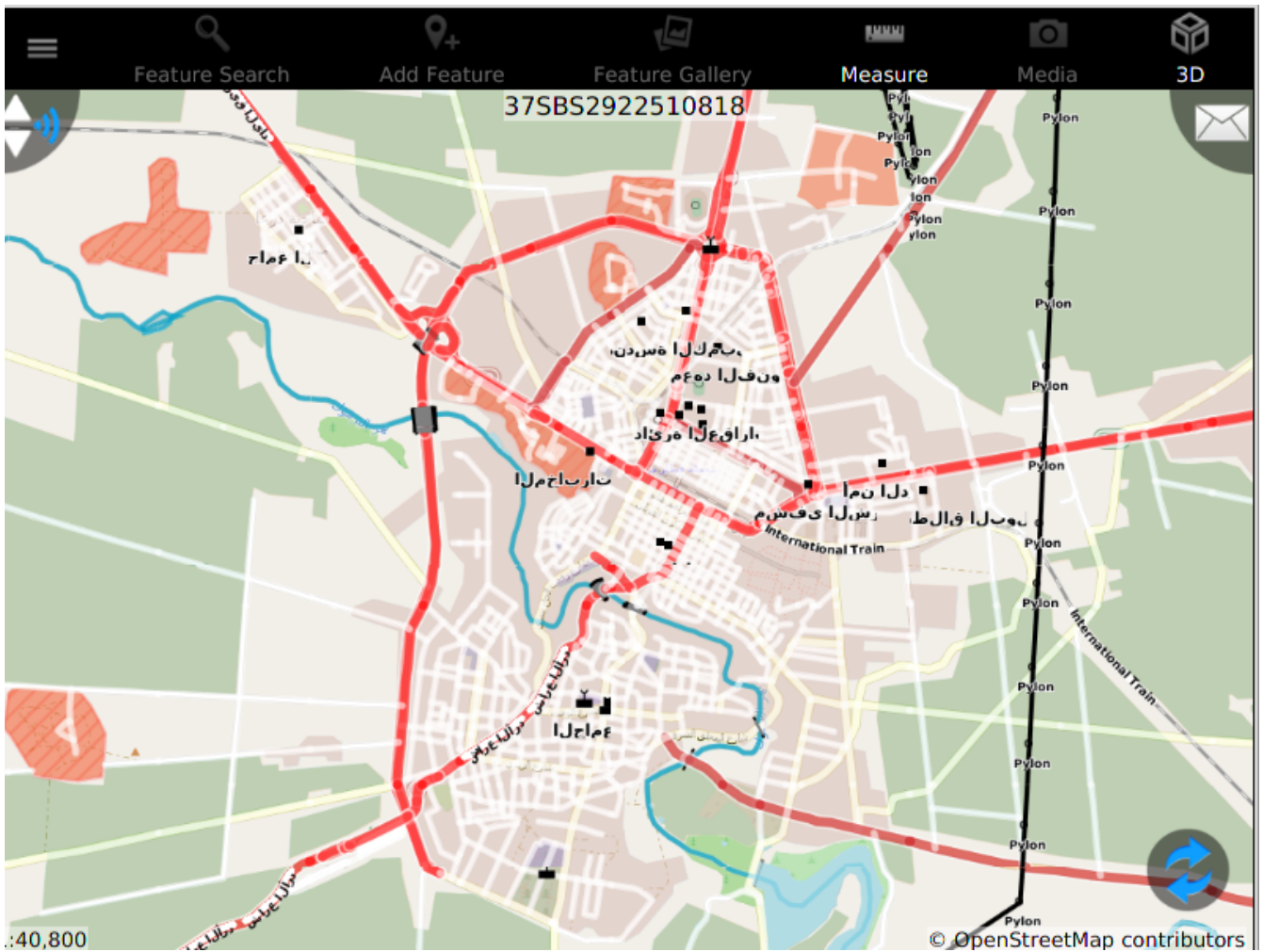


Figure 28. Compusult GOMobile Ecere WMTS (Overlay Style)

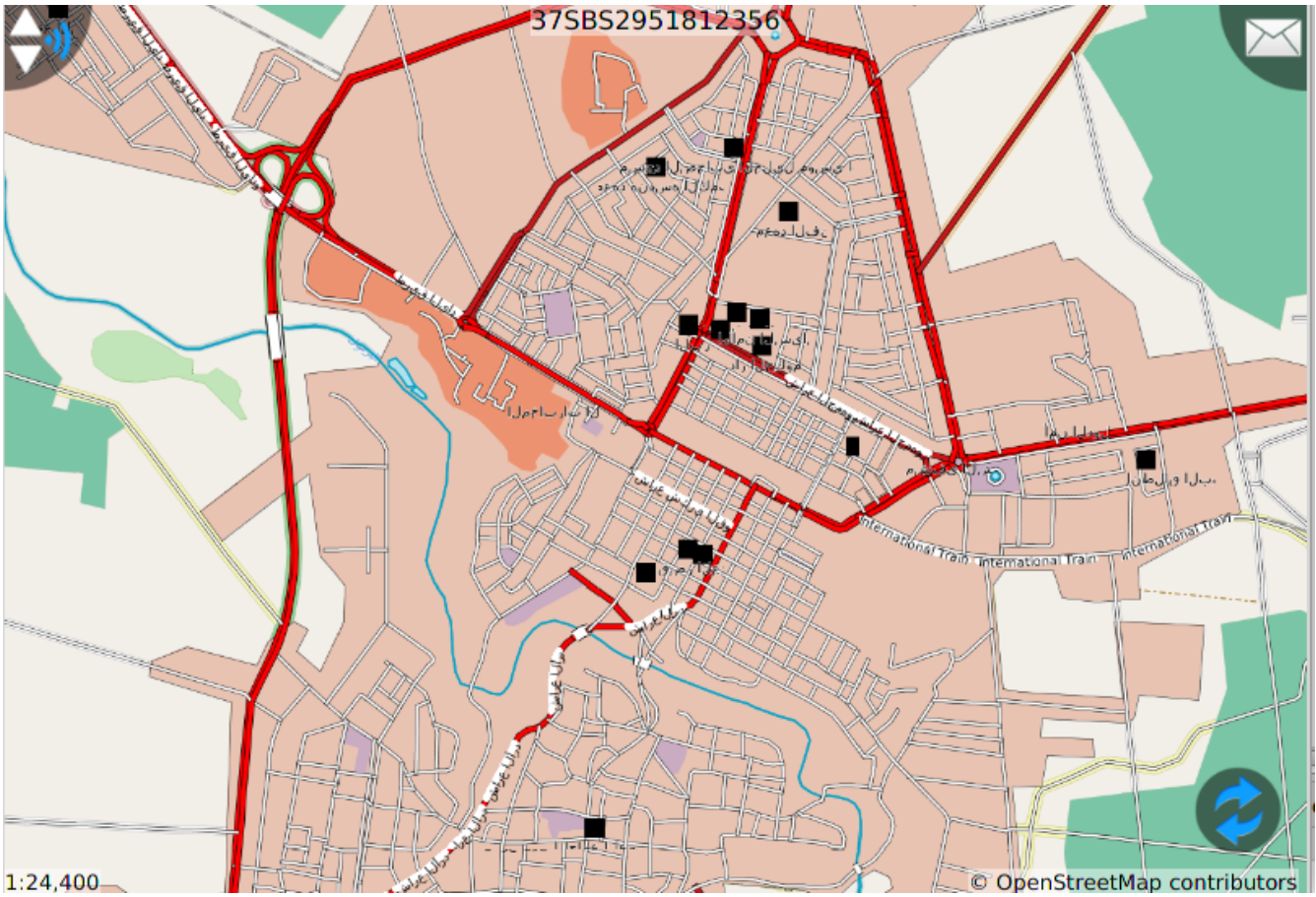


Figure 29. Compusult GOMobile CubeWerx WMTS GeoJSON (Topographic Style)

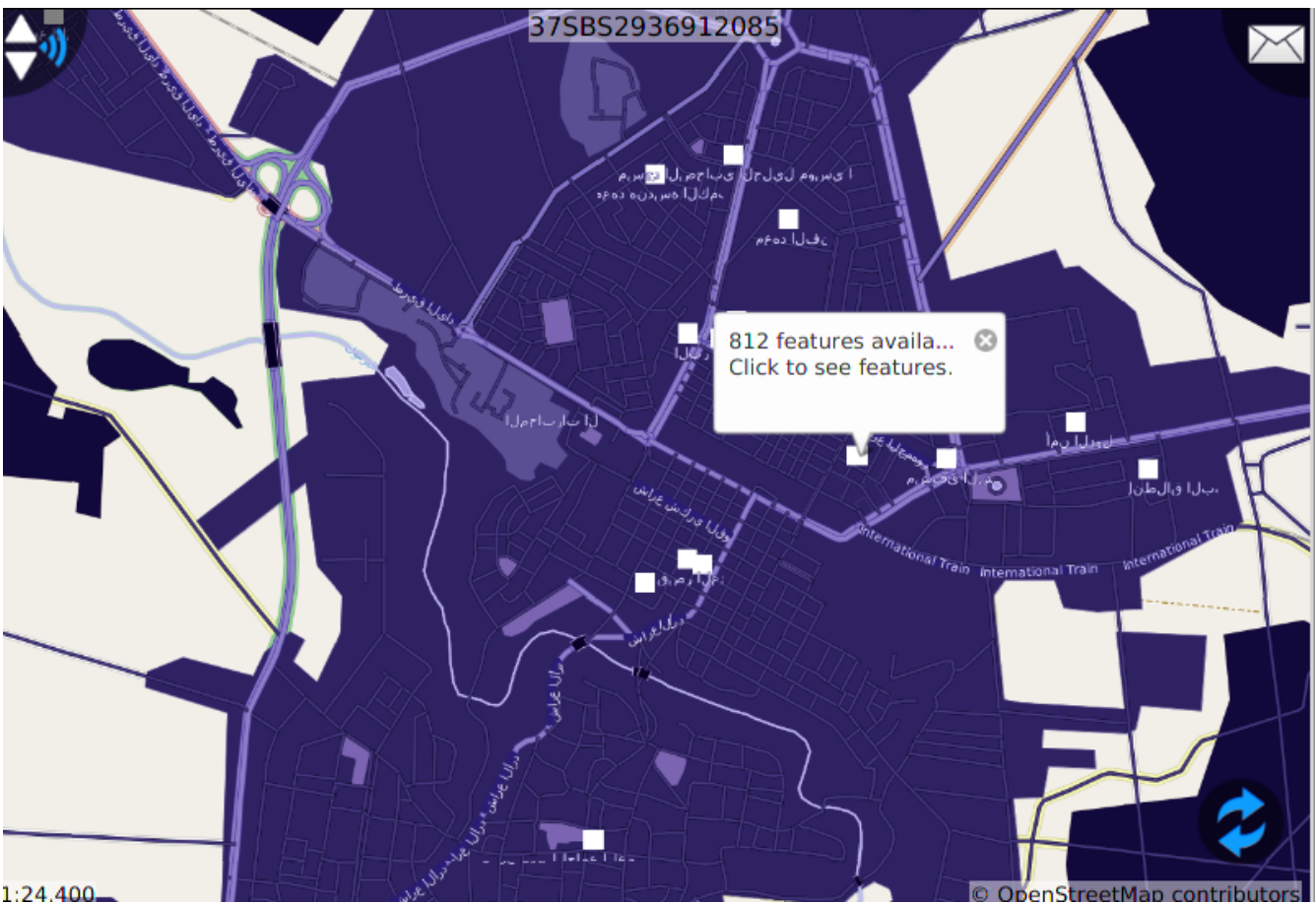


Figure 30. Compusult GOMobile CubeWerx WMTS MVT (Night Style)

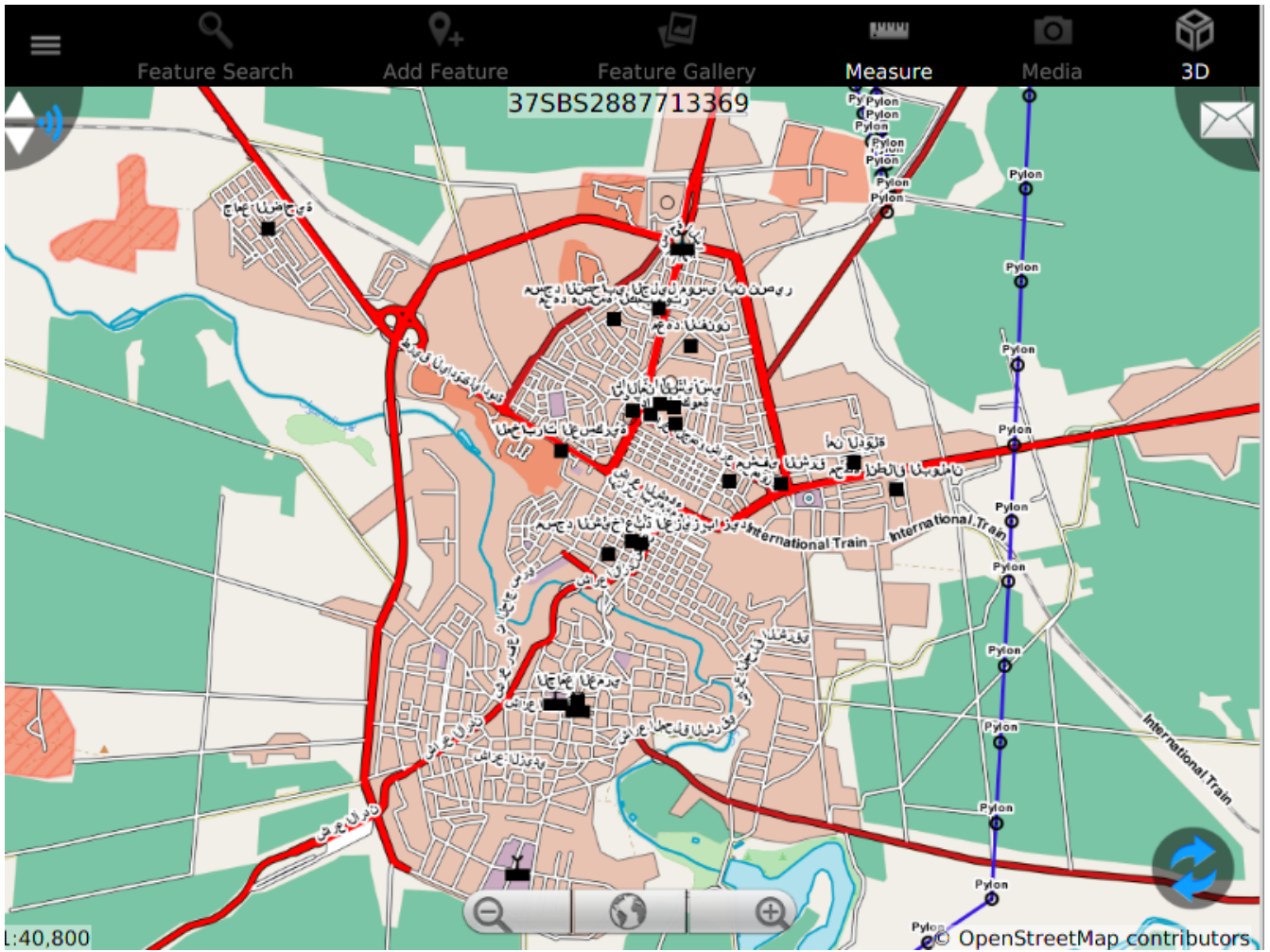


Figure 31. Compusult GOMobile CubeWerx WMTS Static (PNG)

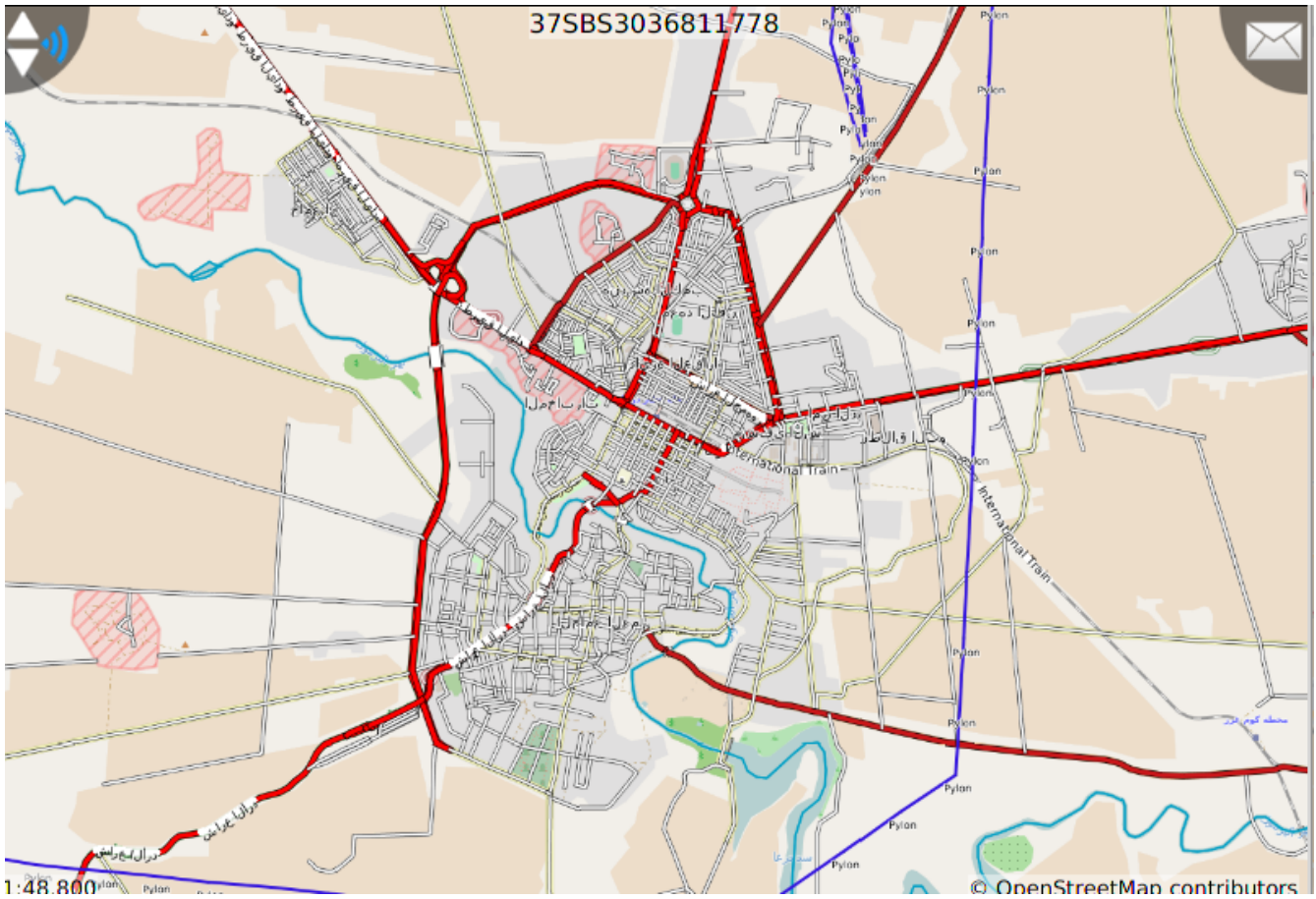


Figure 32. Compusult GOMobile GeoSolution WMTS (Topographic Style Mapbox)

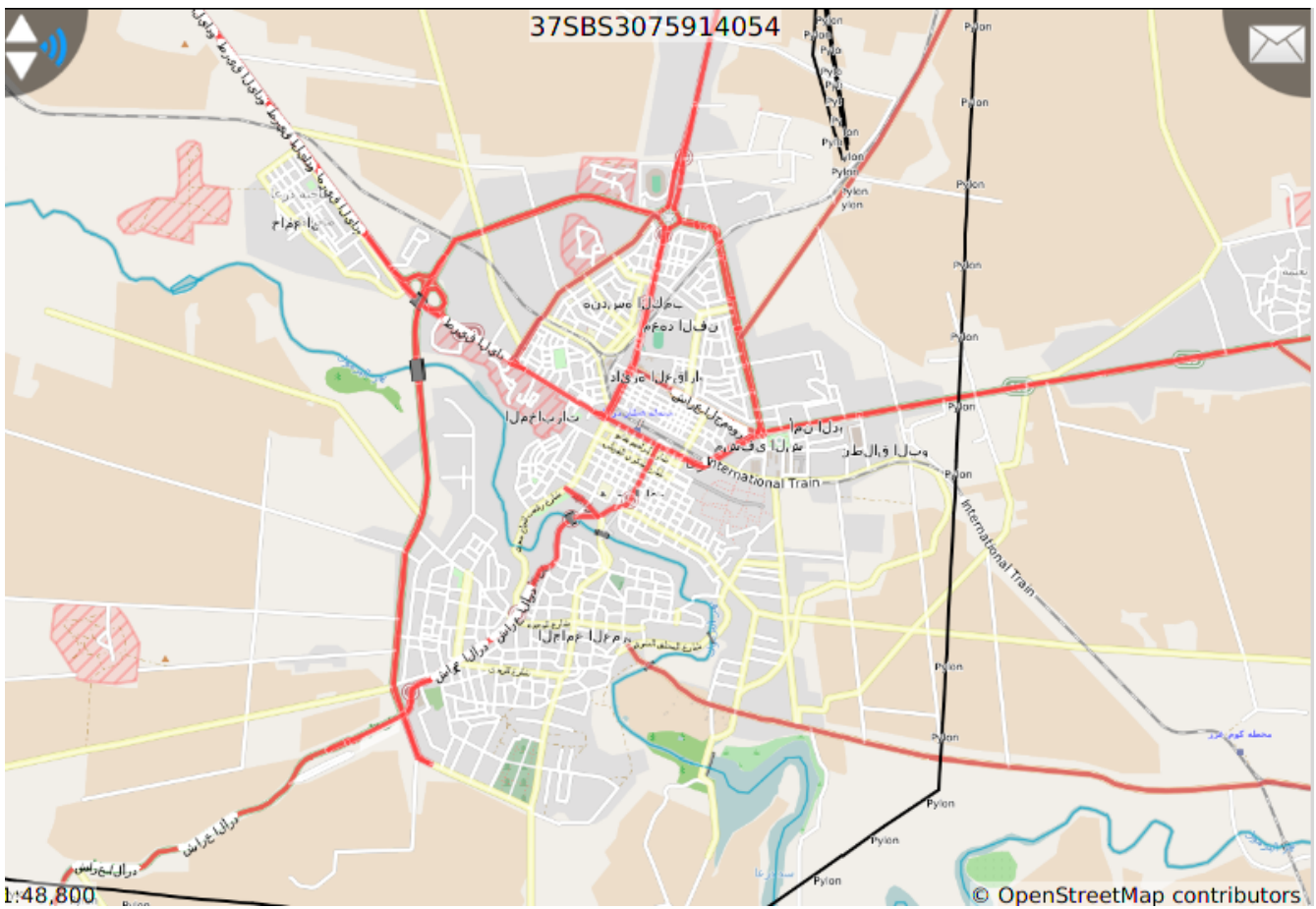


Figure 33. Compusult GOMobile GeoSolution WMTS (Overlay Style SLD)

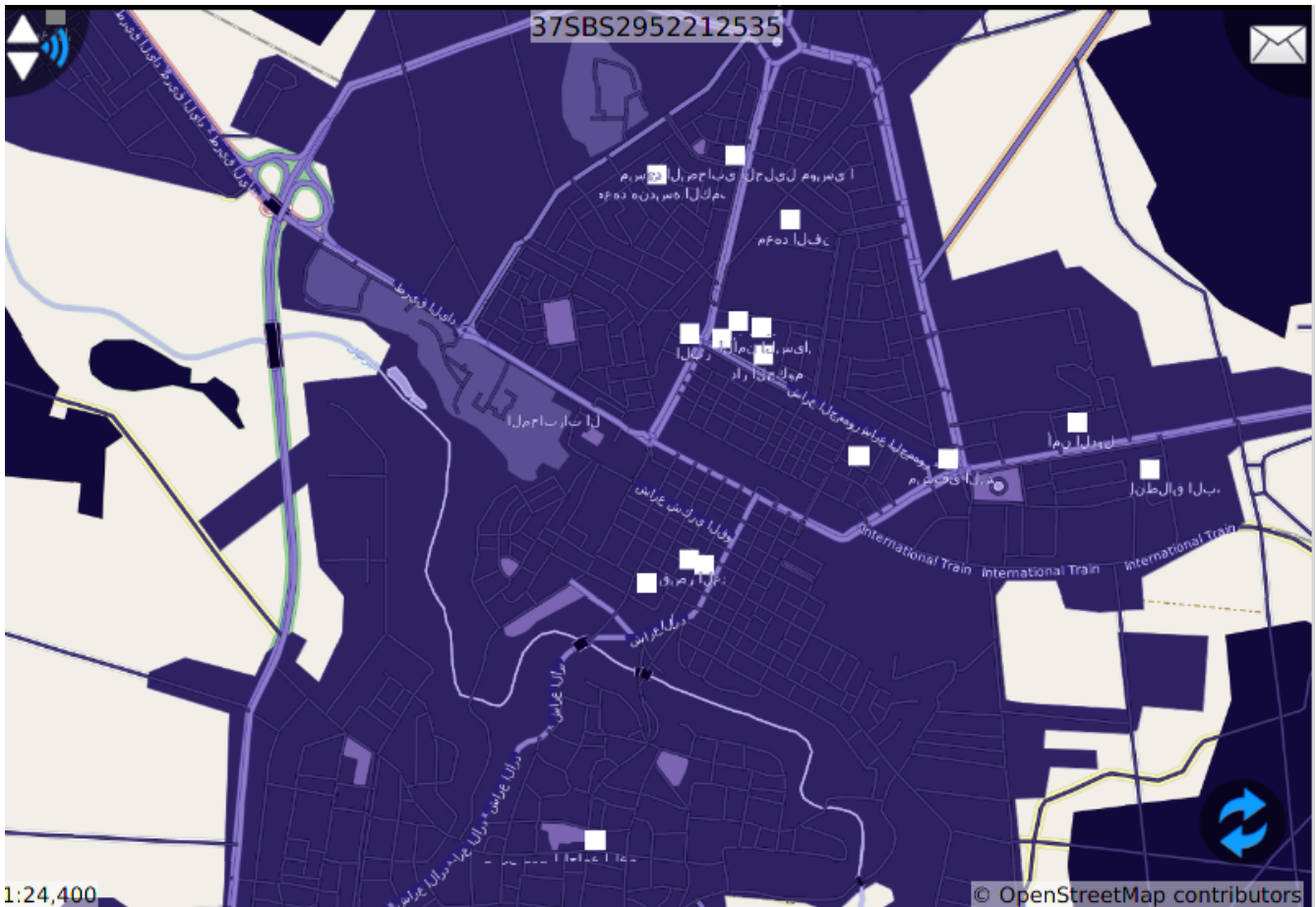


Figure 34. Compusult GOMobile GeoSolution WMTS (Night Style SLD)

Ecere WMTS Client

Ecere was able to reconfirm the interoperability with the GeoSolutions WMTS that was demonstrated during the VTP. The Ecere GNOSIS WMTS client functionality has been tested successfully within GNOSIS Cartographer with both the Cubewerx and Ecere WMTS services. Unfortunately, Ecere was unable to test the styles API from the GeoSolutions or CubeWerx static WMTS because the Ecere client currently only support KVP operations, not the resources API, and those services did not expose a GetStyles KVP operation. Neither the CubeWerx nor the Ecere WMTS service could be styled properly for attributes-specific styles because the Ecere client is currently very dependent on pre-defined attributes schemas. As a result, the need for a mechanism to retrieve attributes schemas through WMTS was discussed, but was not implemented by any participant. Support for listing sub-layers (feature types) inside meta layers, as well as retrieving schemas will likely be implemented in both the Ecere WMTS service and client in the future.

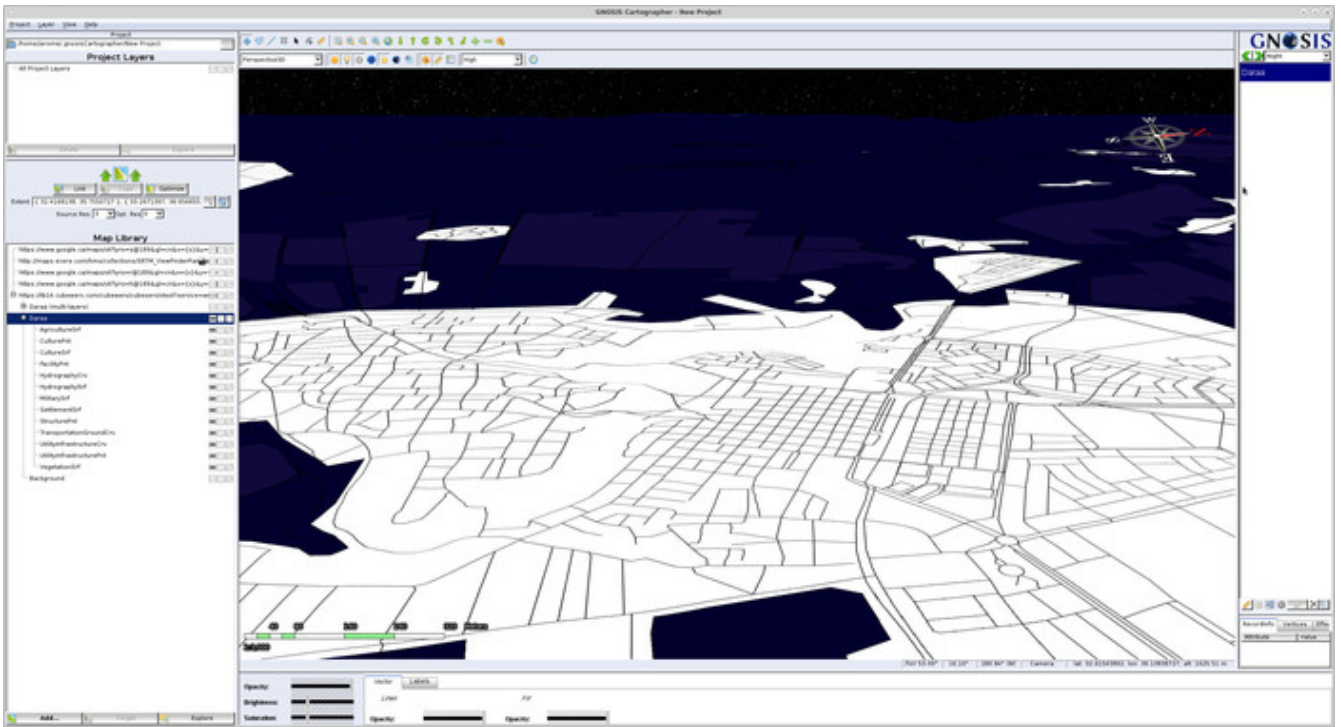


Figure 35. Ecere client visualizing CubeWerx WMTS Service using the Night style

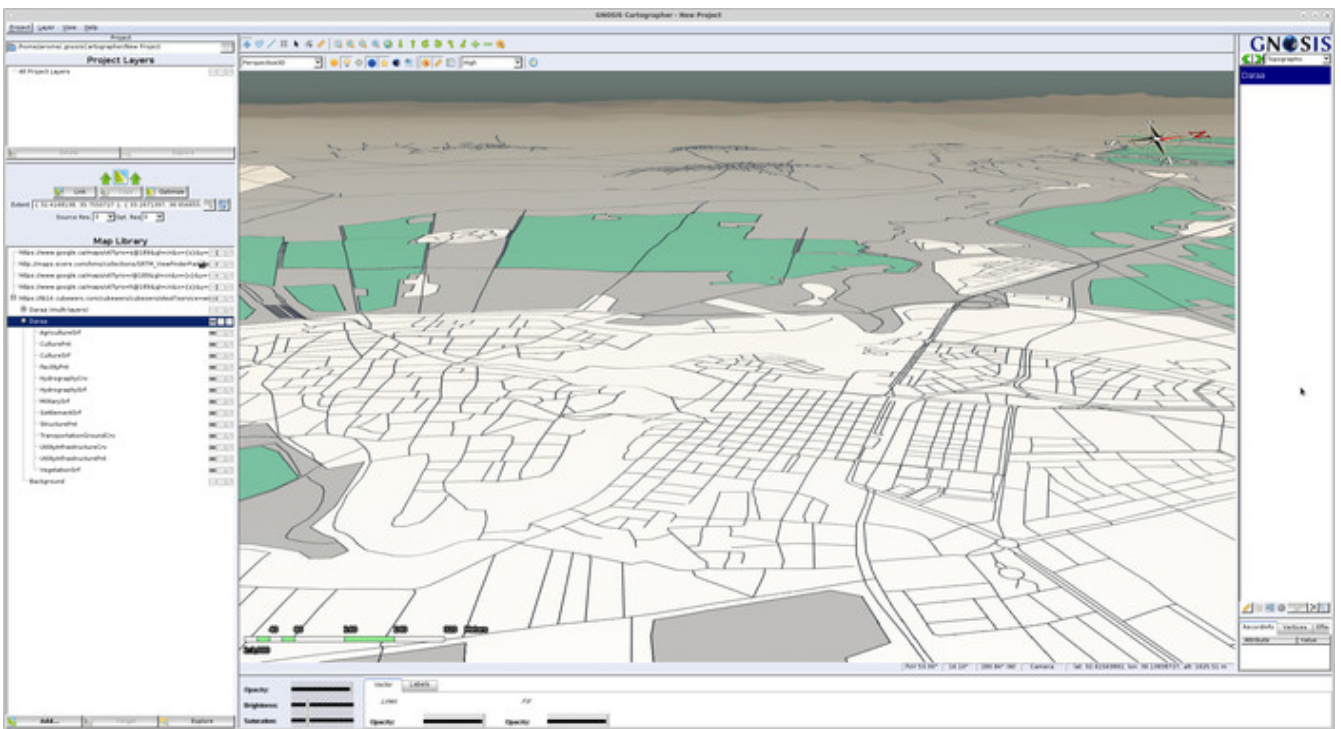


Figure 36. Ecere client visualizing CubeWerx WMTS Service using the Topographic style

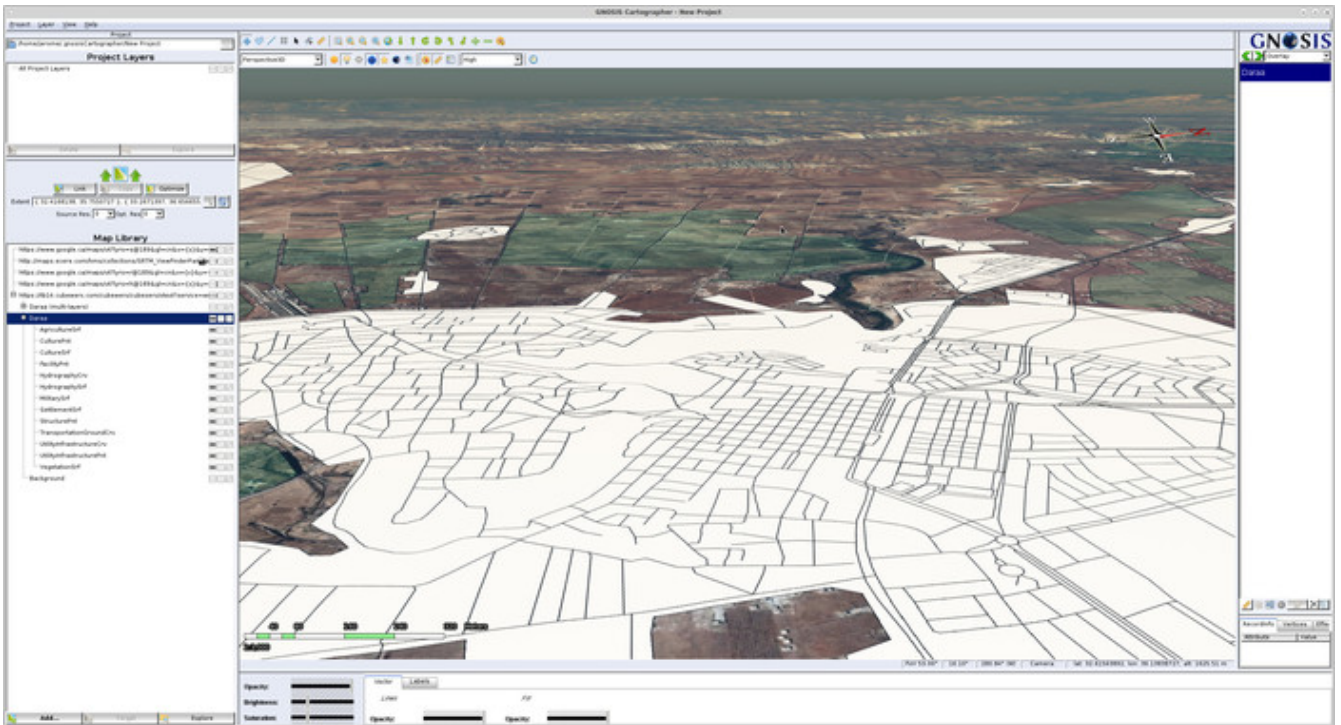


Figure 37. Ecere client visualizing CubeWerx WMTS Service using the Overlay style (Google Maps data underneath)

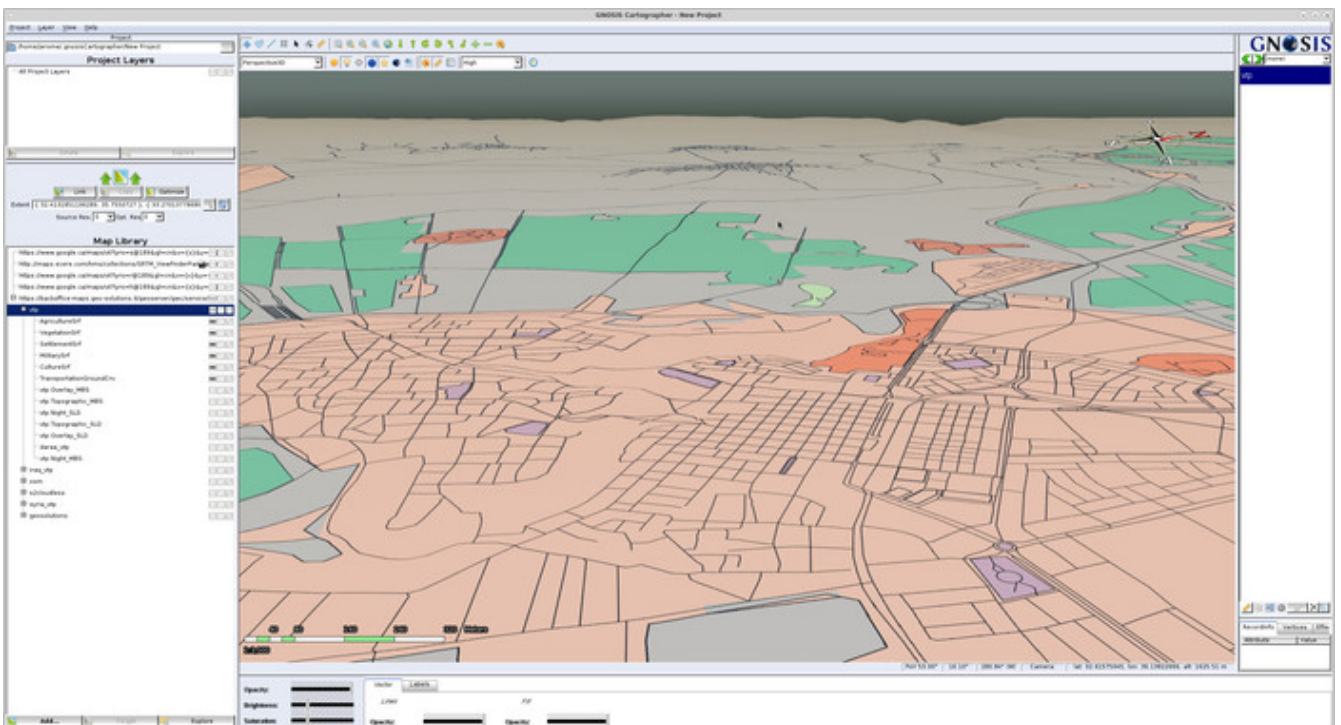


Figure 38. Ecere client visualizing GeoSolutions WMTS Service (styles not yet working due to no GetStyles operations)

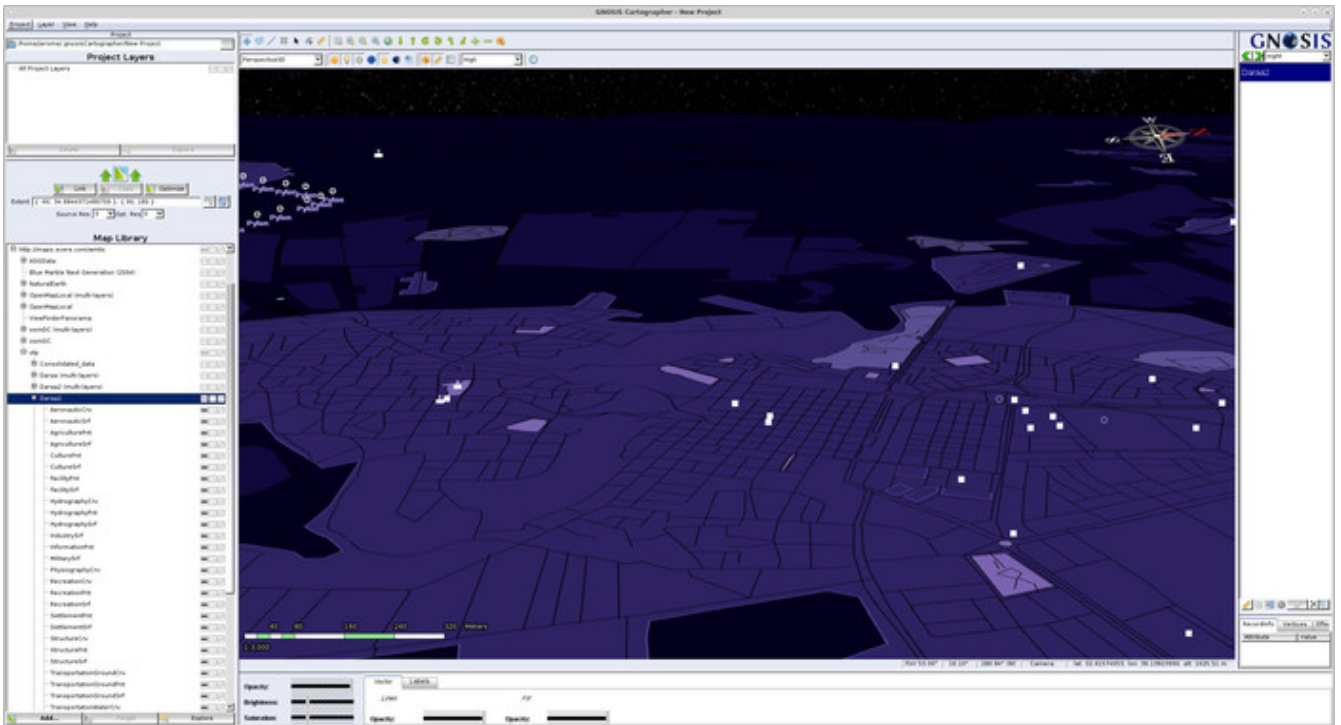


Figure 39. Ecere client visualizing Ecere WMTS Service using the Night style

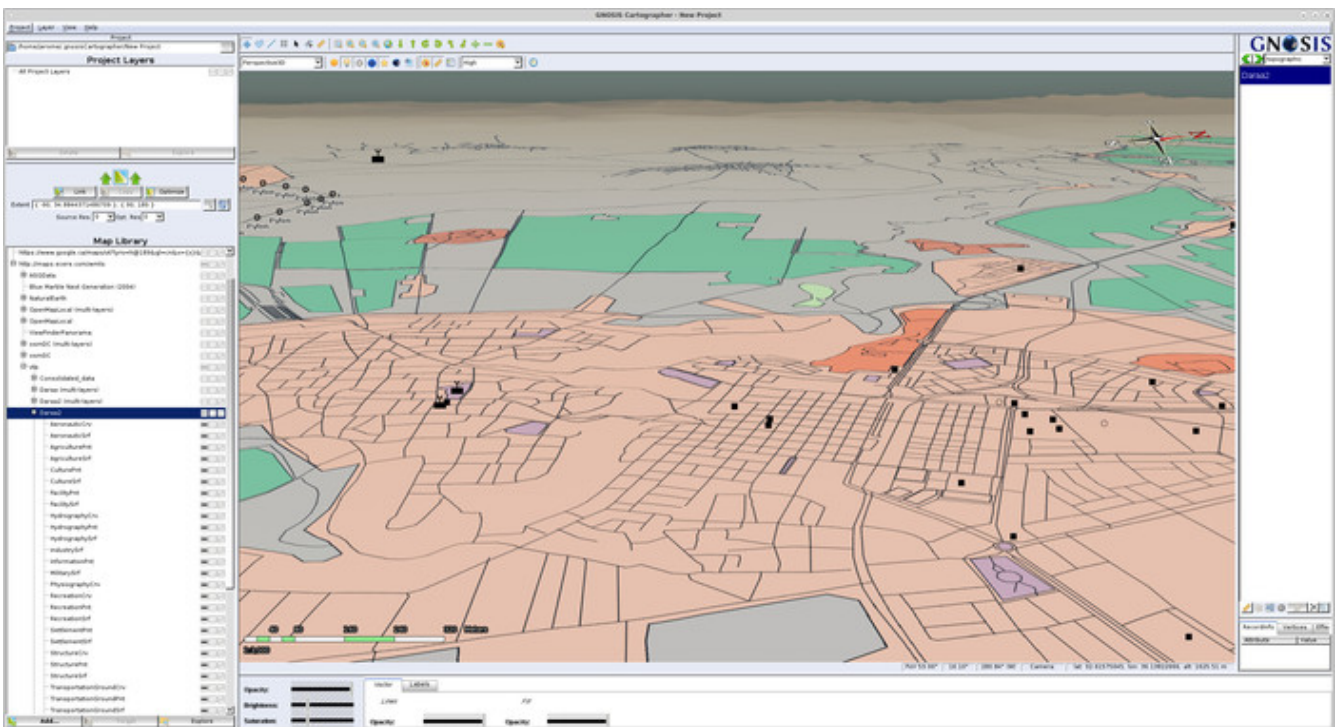


Figure 40. Ecere client visualizing Ecere WMTS Service using the Topographic style

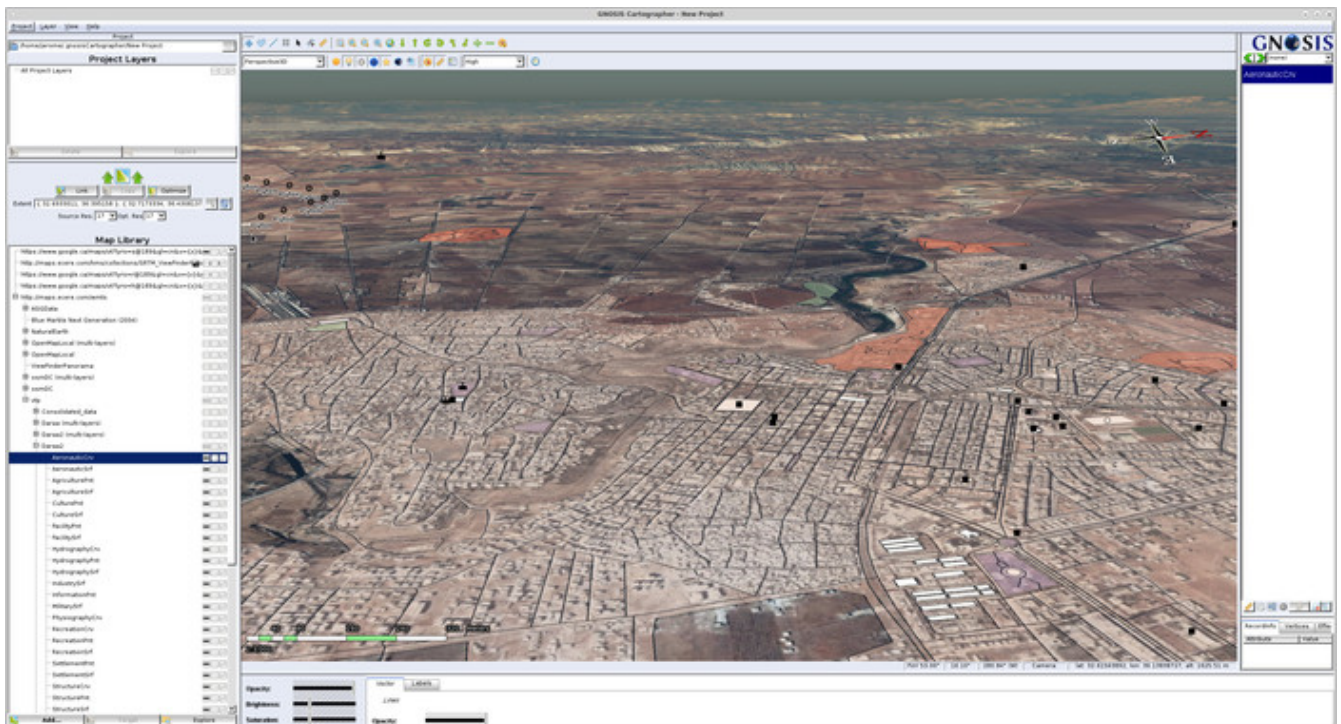


Figure 41. Ecere client visualizing Ecere WMTS Service using the Overlay style (Google Maps data underneath)

7.4.3. Requesting Tiled Feature Data from a WFS

The Feature Tiles API (documented in the Vector Tiles Pilot WFS 3.0 Engineering Report) and [The OpenAPI Styles API](#), both extending the API specified by WFS 3.0 Core, allow for selecting vector tiles and their associated styles as shown in [Figure 42](#).

The design goal for the APIs is to support two approaches of how clients can use the server. The two approaches are complementary, but not exclusive.

1. One option is to study the API using its API definition (at `/api`) and then develop client applications based on the API documentation, the examples and typically an interactive HTML client derived from the API definition (e.g., using Swagger UI). Familiarity with OpenAPI is expected, but no previous knowledge of WFS 3.0 or any other OGC standard is required. OpenAPI also supports code-generation based on the API definition.
2. The other option is to navigate the API based on the API responses and the knowledge about the resource definitions. They will typically start at the landing page, analyze the information, follow links to other resources, etc. The OpenAPI definition may be used to determine details e.g. on filter parameters, but this may not be necessary depending on the application. A client may navigate from the landing page to the `/tiles` and `/tiles/{tilingSchemeId}` resources (for tiles that include all features) or alternatively to the `/collections/{collectionId}/tiles` and `/collections/{collectionId}/tiles/{tilingSchemeId}` resources (for layer/collection-specific tiles) to determine the available tiling schemes and the URI template for accessing the individual tiles. The client may also fetch the `/styles` and `/styles/{styleId}` resources to determine the available styles for the data. The client can then use this information to render the tiled feature data using the available styles.

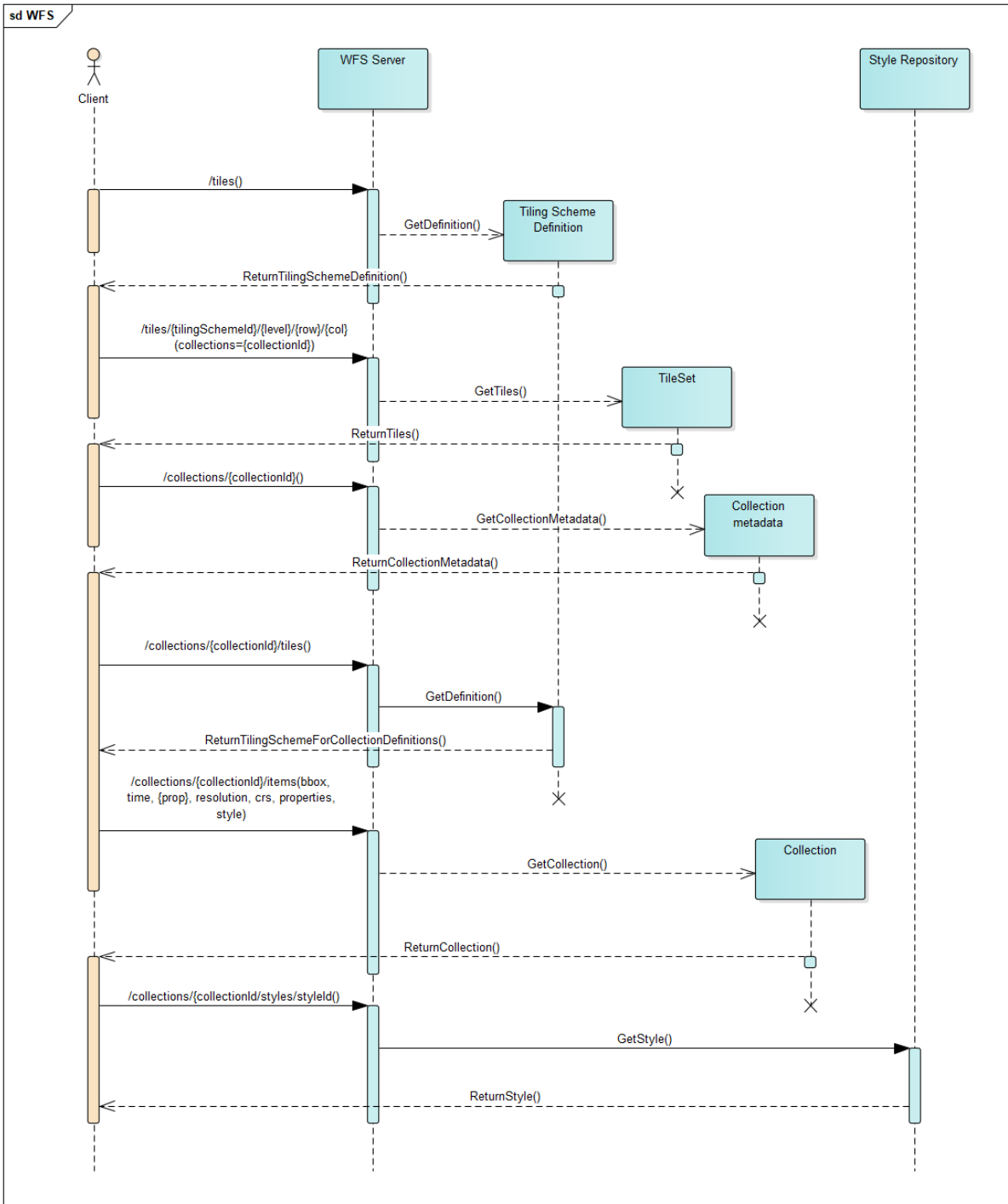


Figure 42. WFS GET Style Sequence Diagram

NOTE

Since the VTPExt focused on styling, most participants had no new content on this topic in this ER. Please see the WFS 3.0 Vector Tiles Extension Engineering Report (OGC 18-078) for more details.

Ecere WFS Client

The Ecere GNOSIS WFS client functionality has been tested successfully within GNOSIS Cartographer with both the interactive instruments, GeoSolutions, and Ecere WFS3 services. However, neither the CubeWerx nor the GeoSolutions WFS service could be styled properly for attributes-specific styles because the Ecere client is currently very dependent on pre-defined attributes schemas. The Ecere service currently provides an attributes schema as an XML Schema

Document using a /schema end-point. (This mechanism is also currently supported by the CubeWerx WFS3 service, but this was not tested during the VTPExt.) The need for a standard mechanism to retrieve attributes schemas through WFS was discussed towards the end of the pilot. The Ecere WFS service and client will be adjusted to support the resulting standard approach, once decided upon.

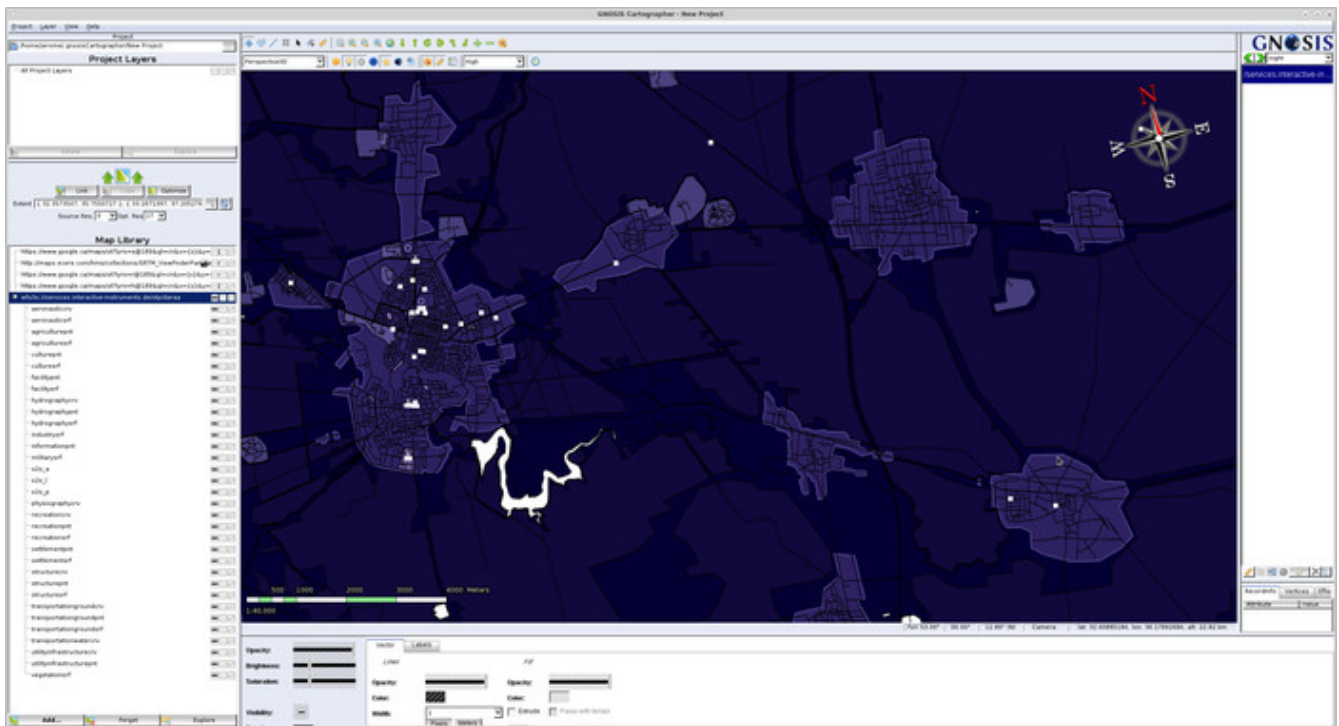


Figure 43. Ecere client visualizing interactive instruments WFS Service using the Night style

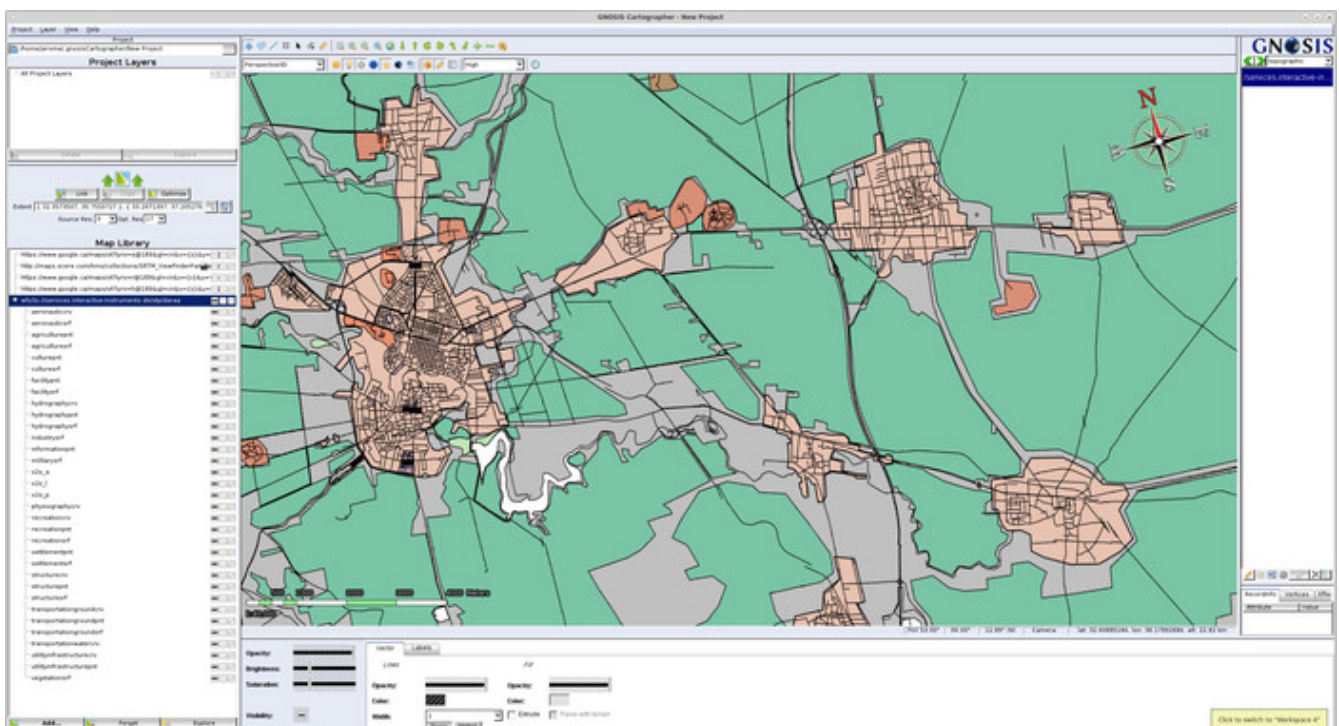


Figure 44. Ecere client visualizing interactive instruments WFS Service using the Topographic style

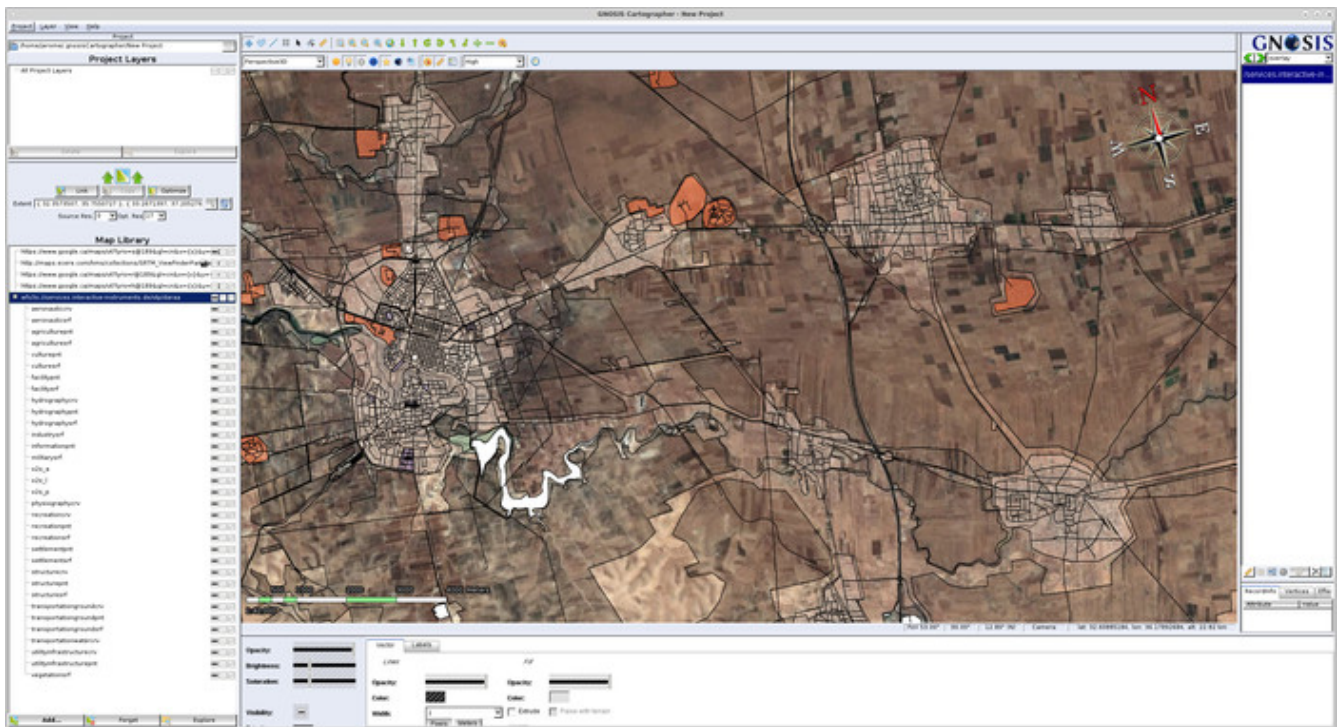


Figure 45. Ecere client visualizing Interactive Instruments WFS Service using the Overlay style (Google Maps data underneath)

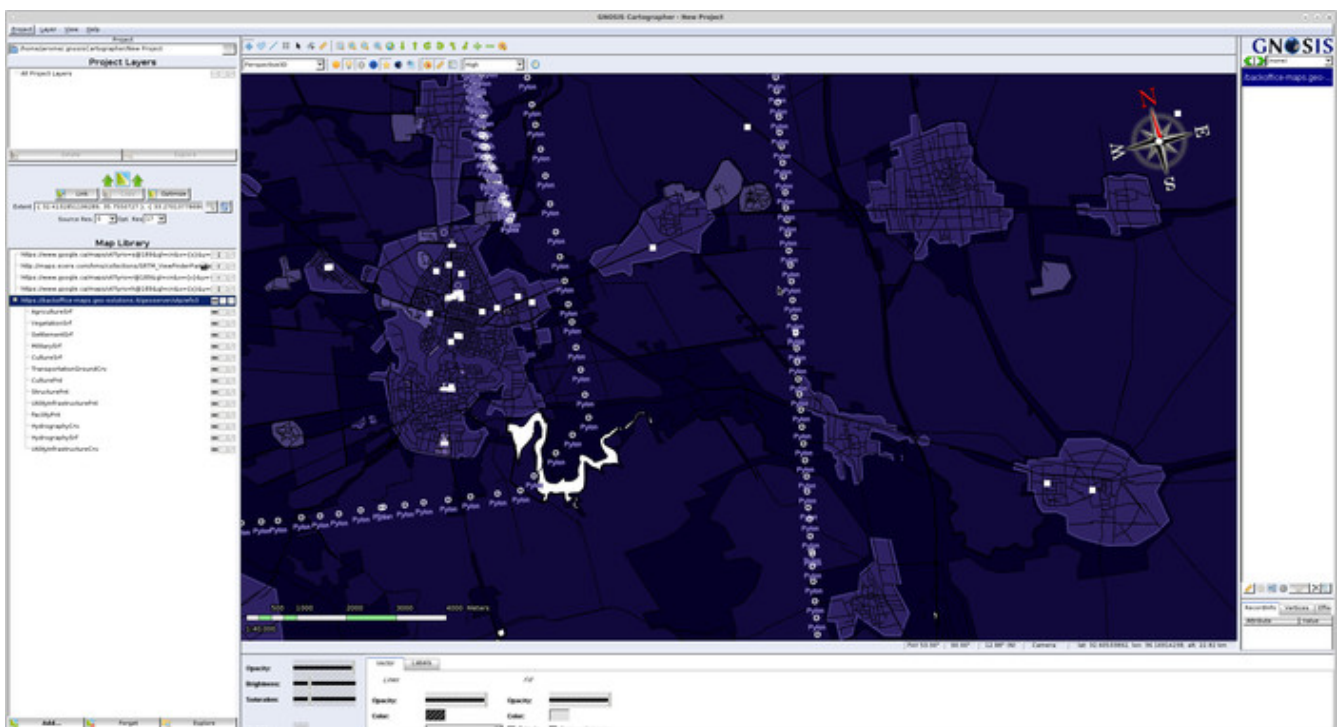


Figure 46. Ecere client visualizing GeoSolutions WFS Service using the Night style

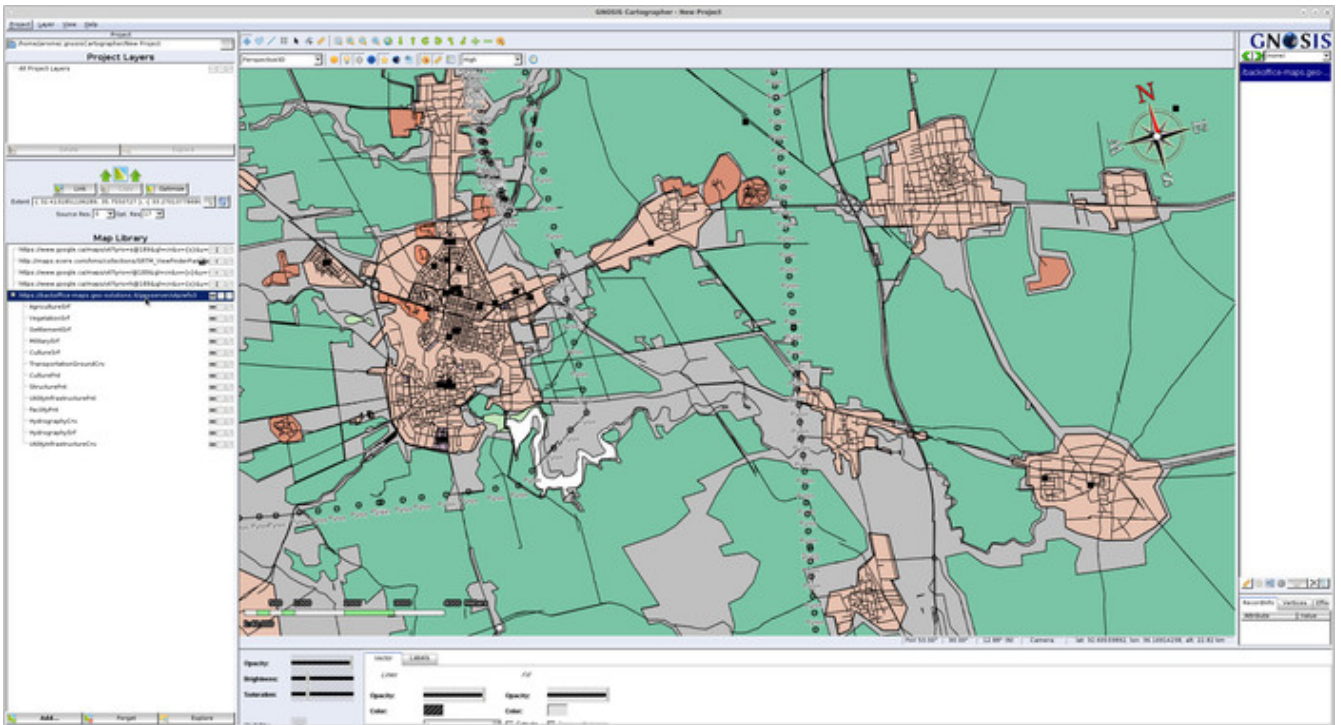


Figure 47. Ecere client visualizing GeoSolutions WFS Service using the Topographic style

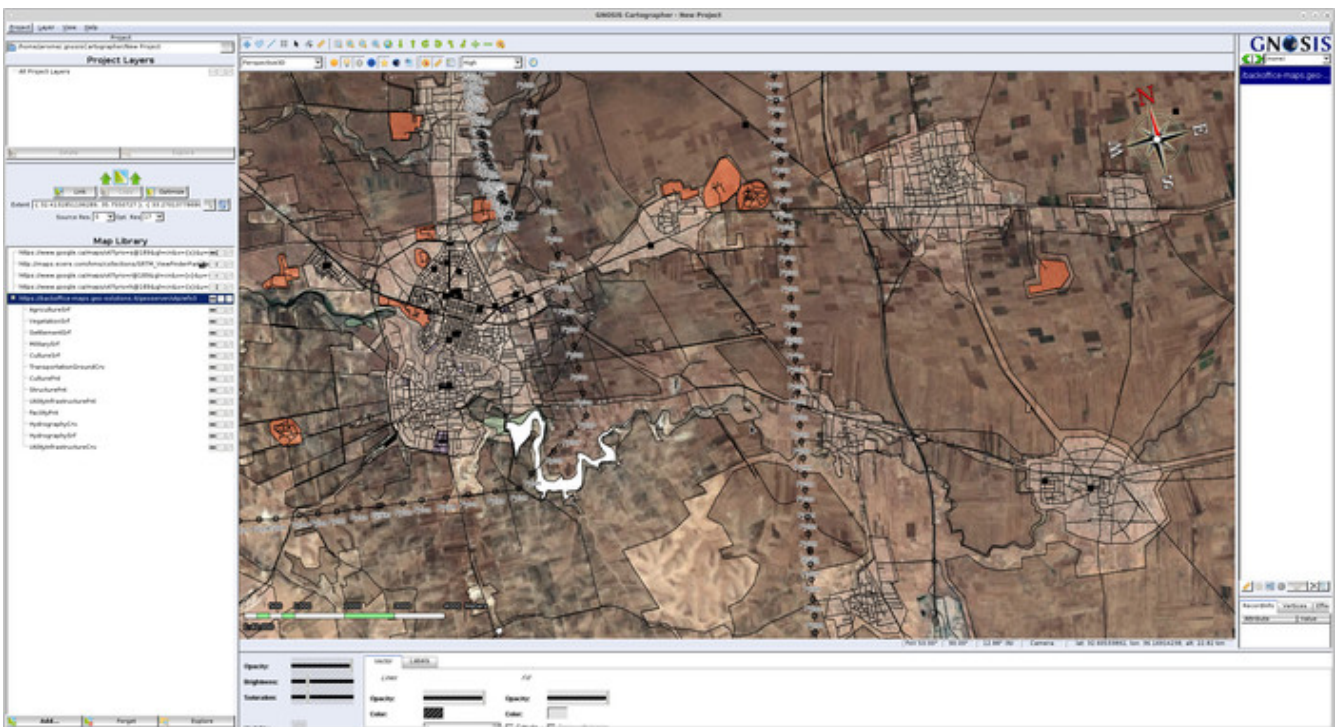


Figure 48. Ecere client visualizing GeoSolutions WFS Service using the Overlay style (Google Maps data underneath)

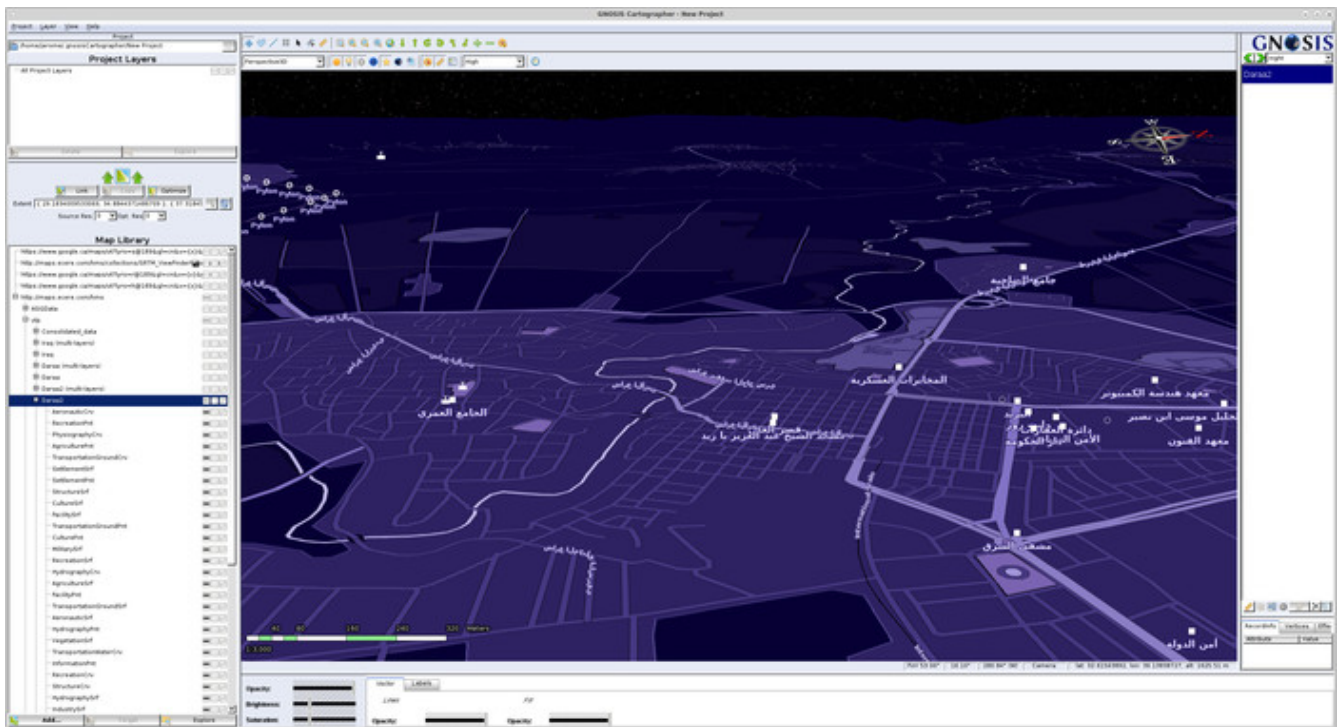


Figure 49. Ecere client visualizing Ecere WFS Service using the Night style

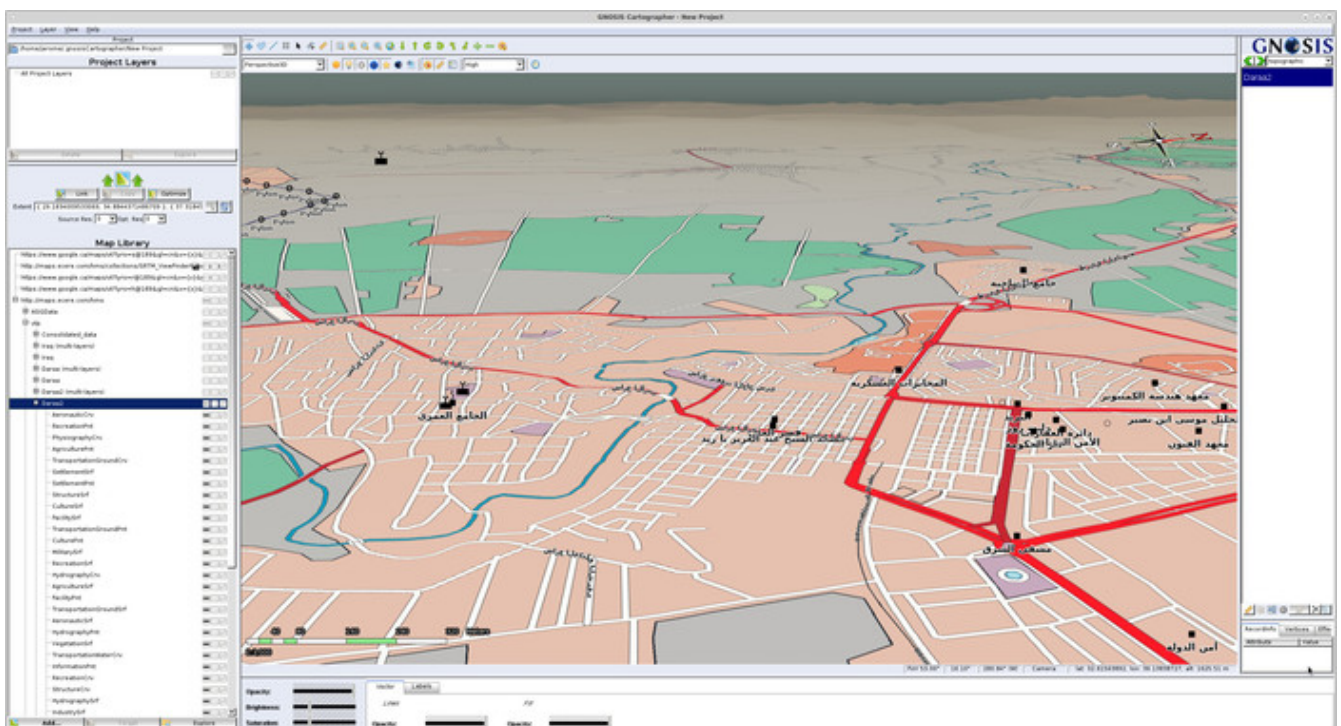


Figure 50. Ecere client visualizing Ecere WFS Service using the Topographic style

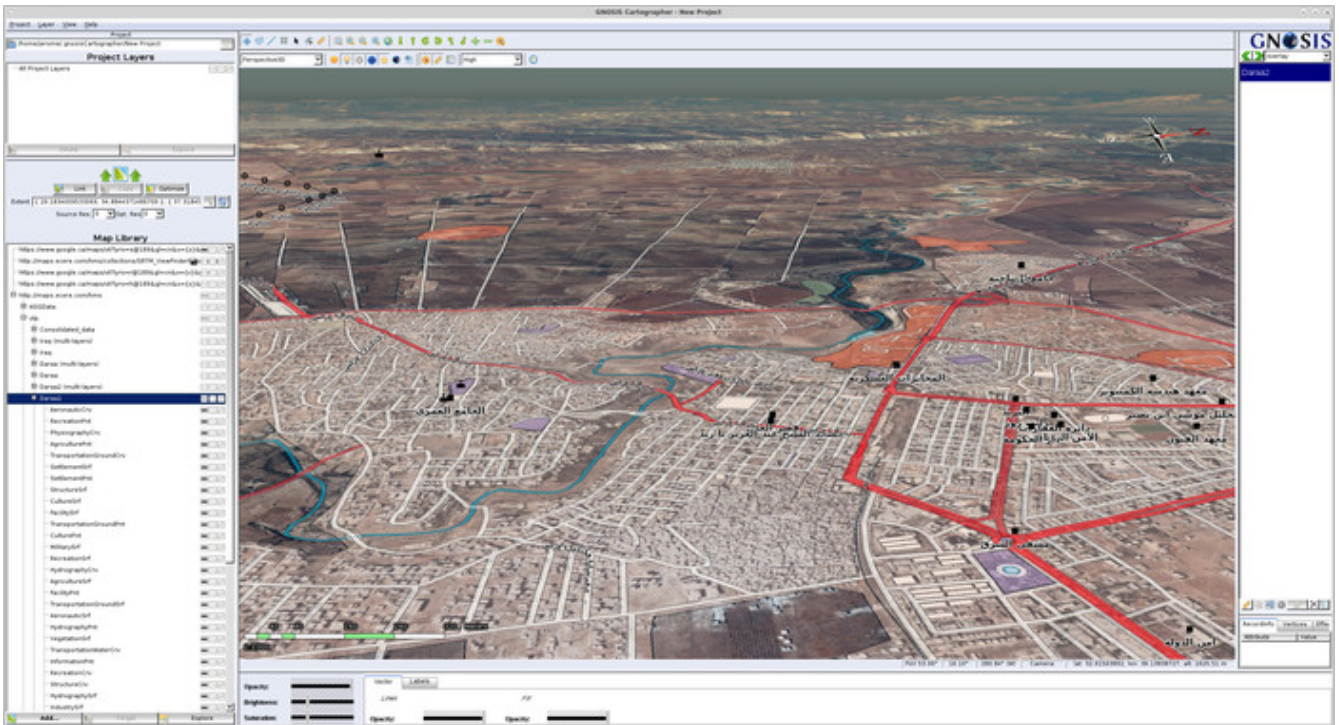


Figure 51. Ecere client visualizing Ecere WFS Service using the Overlay style (Google Maps data underneath)

7.4.4. Modifying Style Sheets

The WFS Feature Tile extension also allows for editing a WFS style as shown in (Figure 52) below.

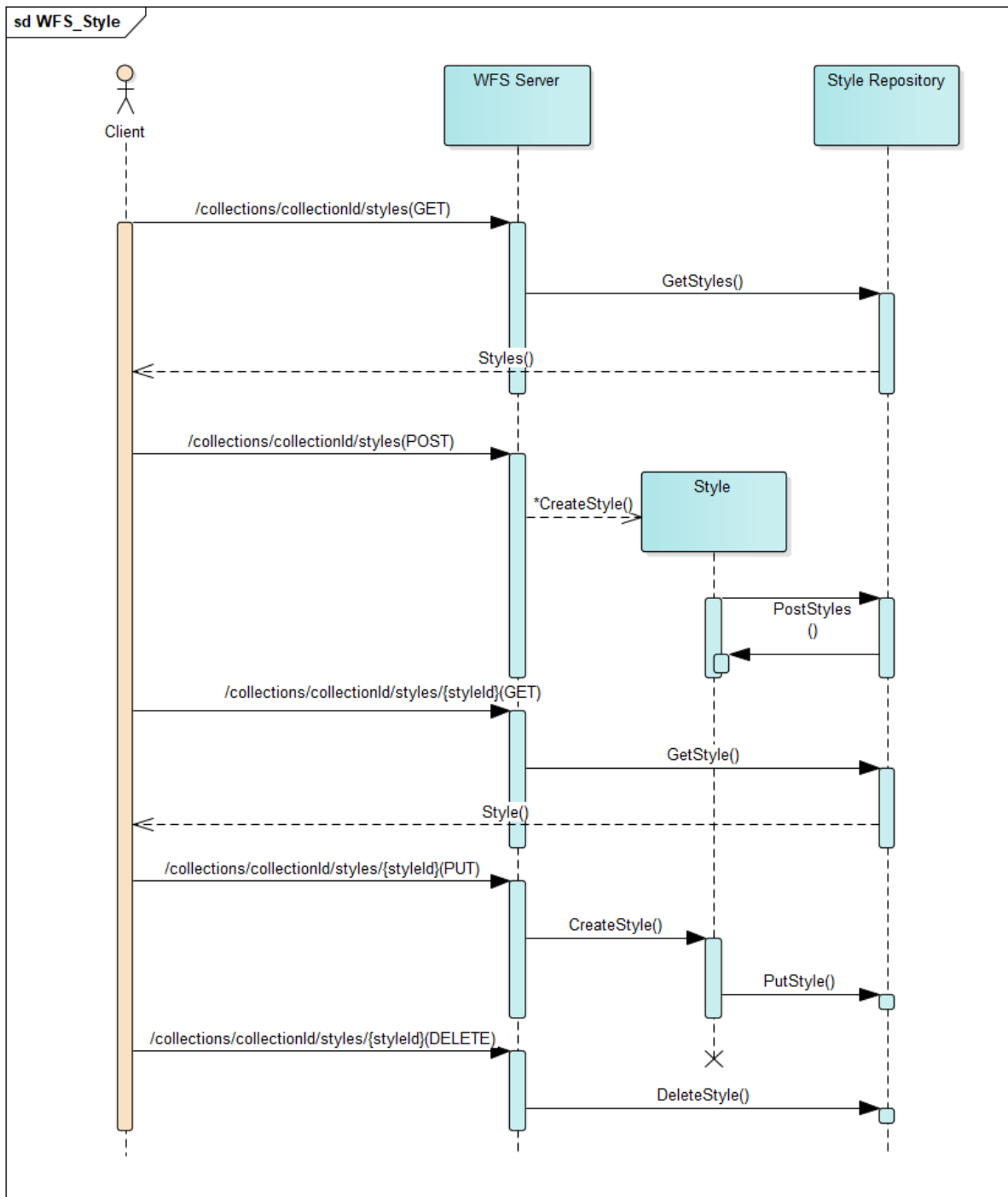


Figure 52. WFS Edit Style Sequence Diagram

GeoSolutions

WFS3 Client

This client shows a complete workflow to get styles using a [GeoServer](http://geoserver.org/) [http://geoserver.org/], Interactive Instruments, or Ecere WFS3 service and editing of styles client side.

The demo application has been build with [MapStore](https://mapstore.geo-solutions.it/mapstore/#/) [https://mapstore.geo-solutions.it/mapstore/#/] framework and it uses [OpenLayers](https://openlayers.org/) [https://openlayers.org/] as map renderer.

As represented in [Figure 53](#), the application performs following steps:

1. get all layers from a WFS3 collection

2. get all dataset styles not assigned to layers from WFS3 styles
3. for each style it requests the style body
4. the client splits all style bodies and verifies if the name inside the NamedLayer (SLD) or 'source-layer' (MBStyle) matches with a layer in the collection
5. for each match in the collection the client will render a separate layer and apply the portion of styles previously matched.
6. after every editing the style is recomposed and the client can request to the server to update or delete the current style (PUT or DELETE)

Link to demo repository: <https://github.com/geosolutions-it/ogc-vector-tiles-vtp/tree/master/MapStoreStyle>

Link to live demo: GeoServer WFS3 Service: <http://vtp2018.s3-eu-west-1.amazonaws.com/vtpext-wfs.html/>

TIP

Interactive instruments WFS3 Service: http://vtp2018.s3-eu-west-1.amazonaws.com/vtpext-wfs.html/?wfs3=interactive_instruments

Ecere WFS3 Service: <http://vtp2018.s3-eu-west-1.amazonaws.com/vtpext-wfs.html/?wfs3=ecere>

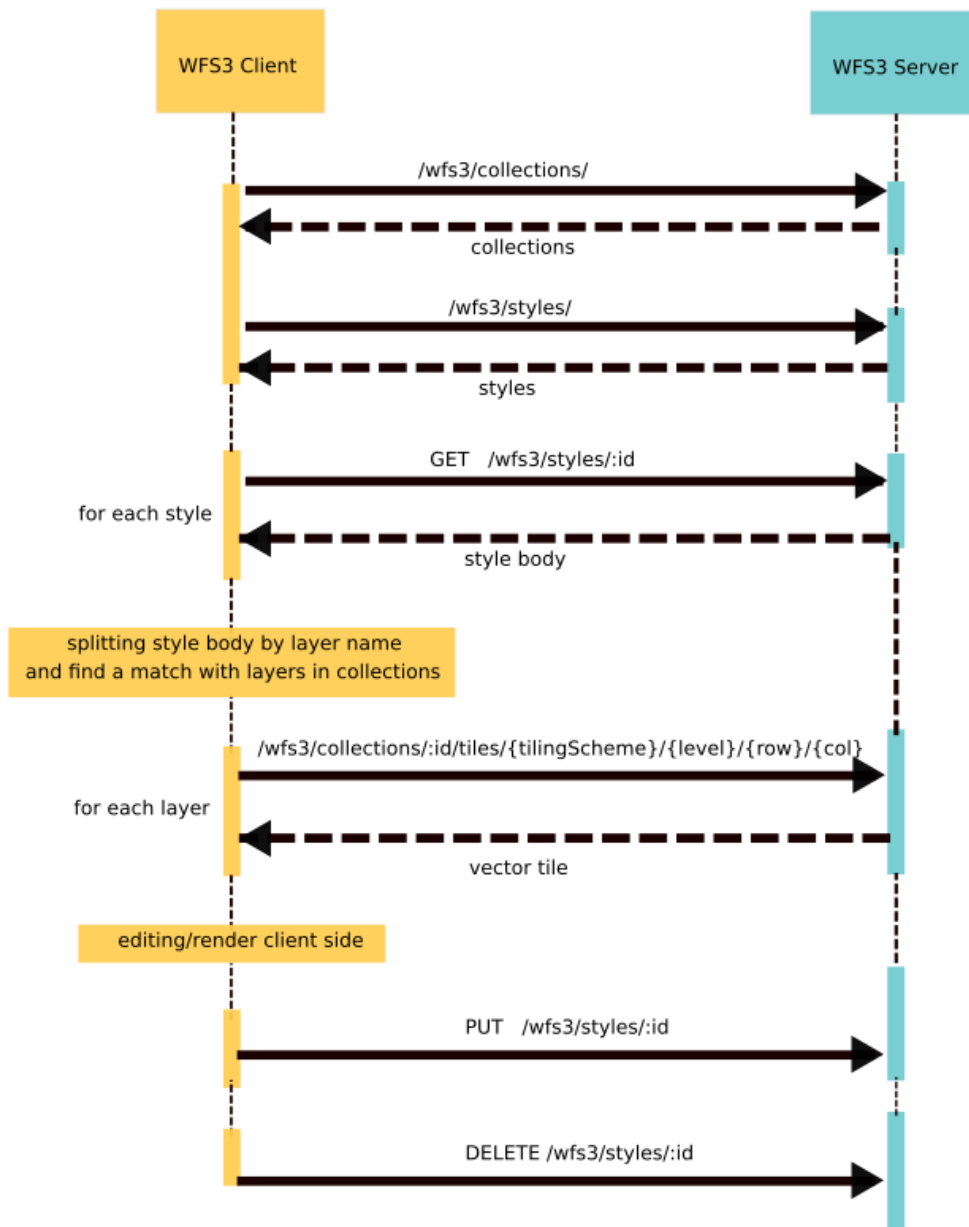


Figure 53. WFS3 - Client Sequence Diagram

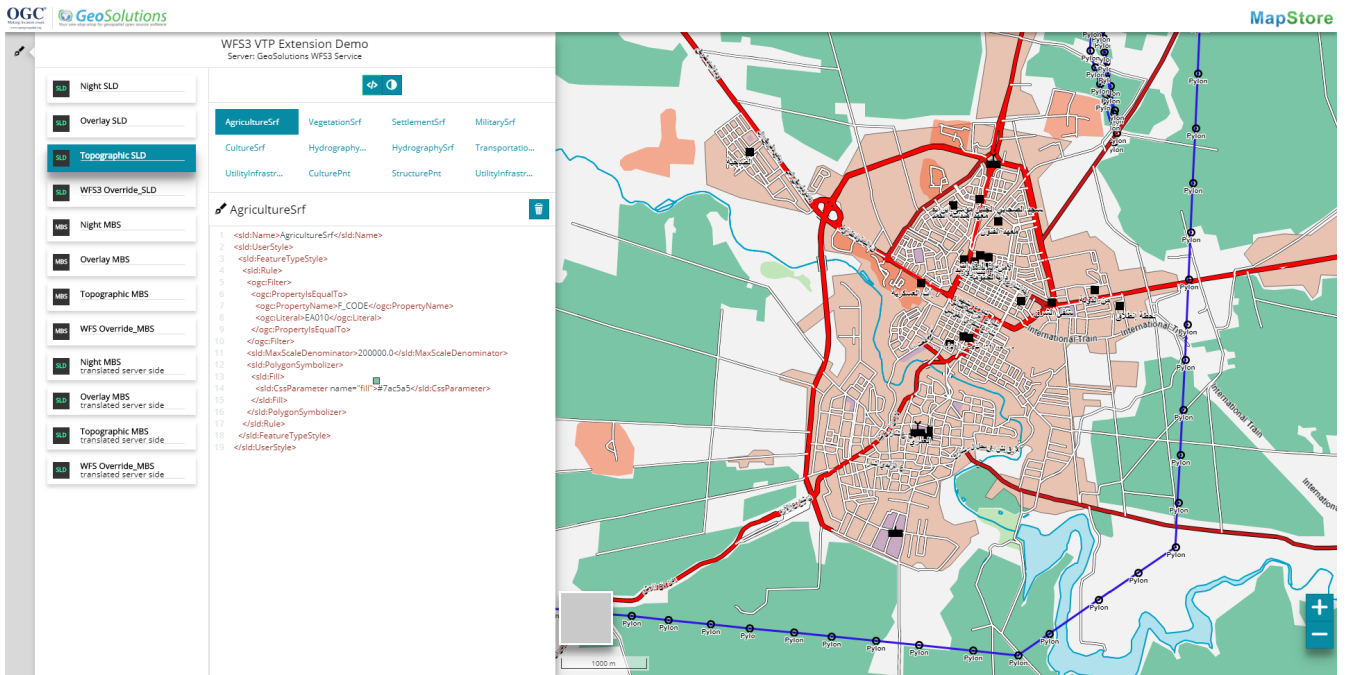


Figure 54. MapStore Client Splits Styles in Named Layer in SLD of Topographic Style with WFS3 Vector Tiles

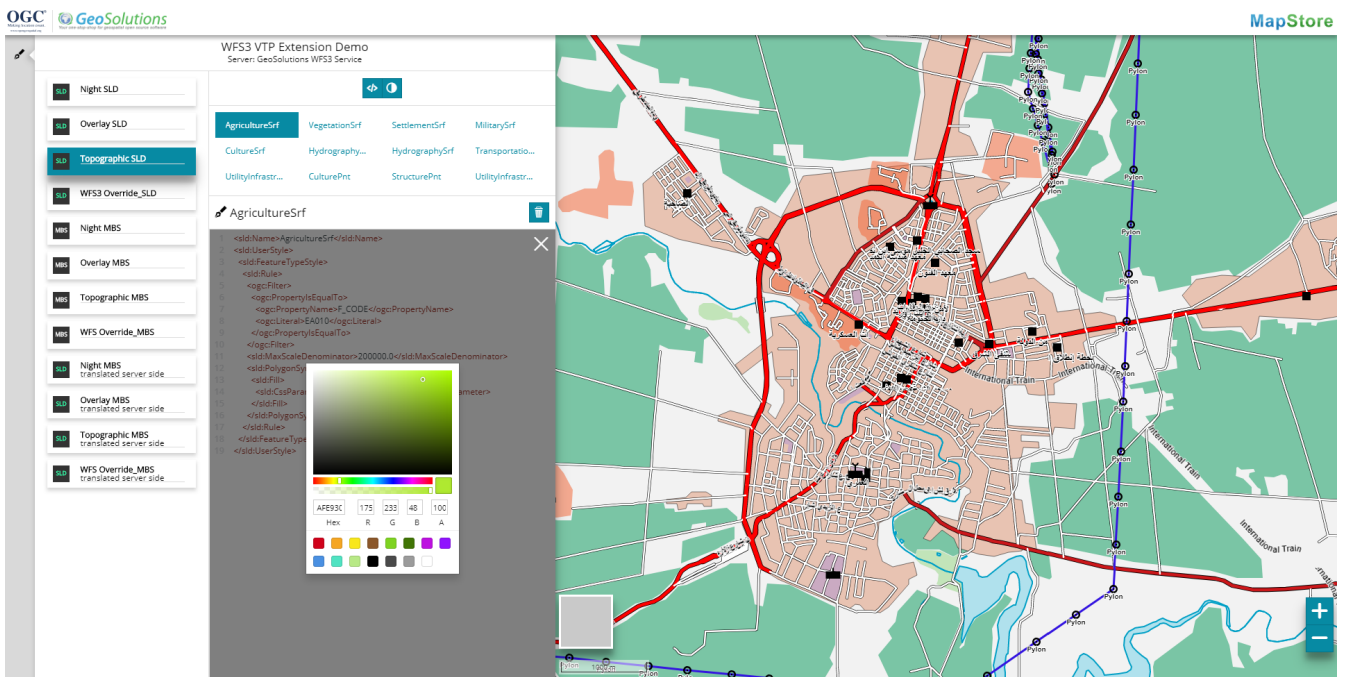


Figure 55. MapStore Client Editing Color of Agriculture Surface of Topographic Style with WFS3 Vector Tiles

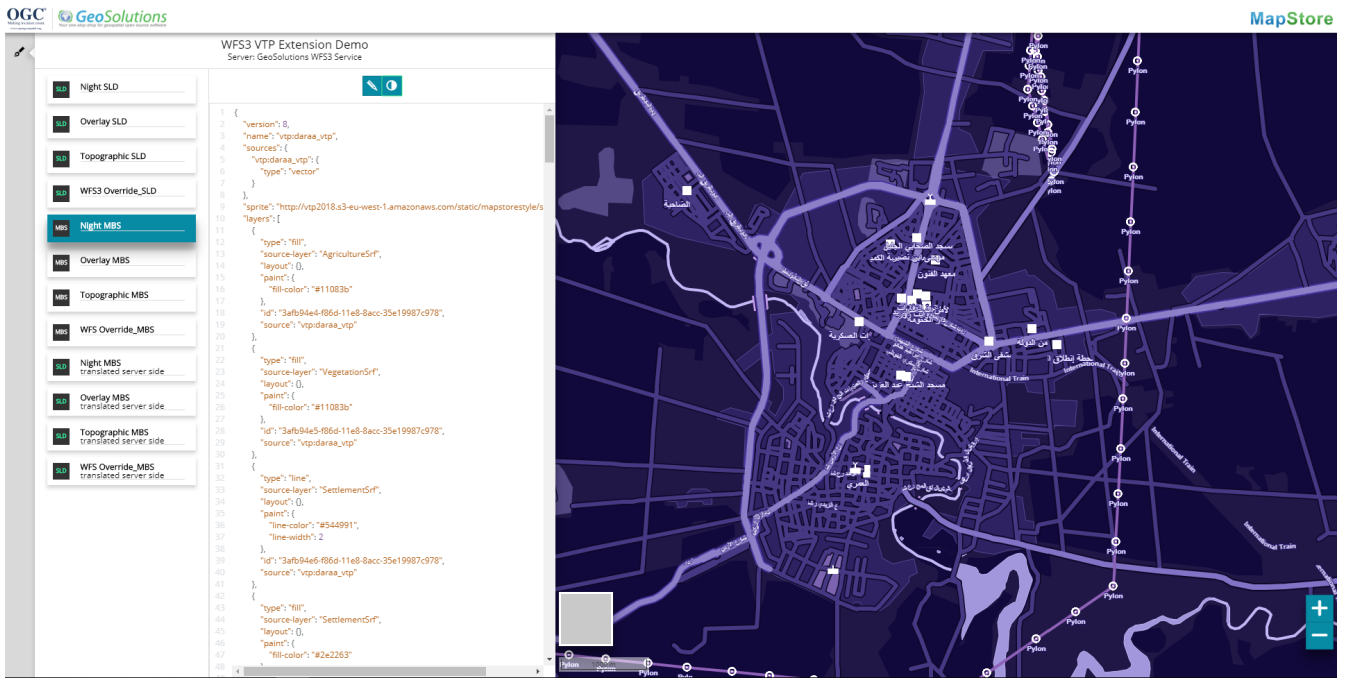


Figure 56. MapStore Client Shows the Complete Night Mapbox Style with GeoServer WFS3 Service

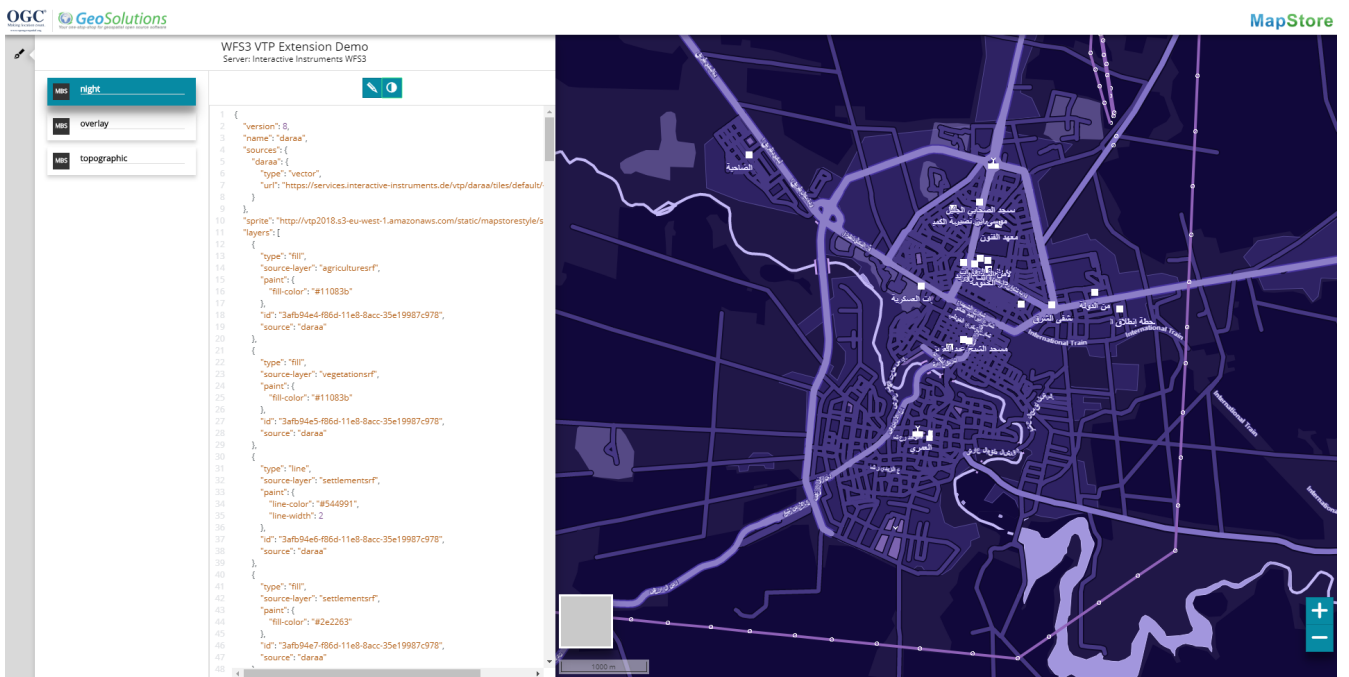


Figure 57. MapStore Client Shows the Complete Night Mapbox Style with Interactive Instrument WFS3 Service

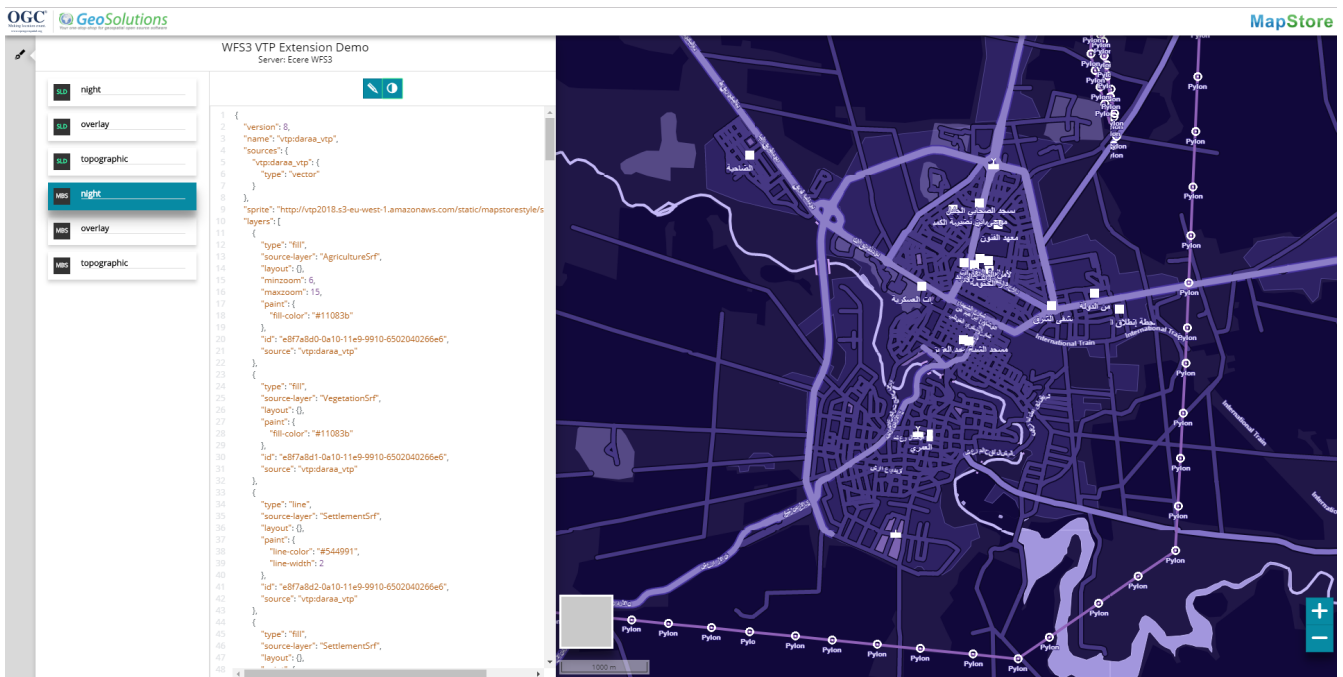


Figure 58. MapStore Client Shows the Complete Night Mapbox Style with Ecere WFS3 Service

7.4.5. Displaying Tiled Feature Data

Analysis

On a mobile client, performance and scalability issues potentially have a greater impact. This particularly applies to drawing to the screen; the less drawing the better. One participant found that creating a single image out of all of the layers, rather than one image for each layer, was significantly better for performance. With this in mind, it is more convenient to have all of the layers in a single tile set. If the layers are kept separate, it would be better for the client to produce one aggregate image than to create one image for each layer.

GeoSolutions

GeoSolutions produced a WMTS Client. This demo shows an application built with [MapStore](https://mapstore.geo-solutions.it/mapstore/#/) [https://mapstore.geo-solutions.it/mapstore/#/] framework that lets the user switch the rendering between server and client side using a WMS Service from [GeoServer](http://geoserver.org/) [http://geoserver.org/]. GeoServer exposes WMS tiles in different formats, including vector ones such as MVT, so in a GetMap request it is possible to set the `format` parameter to 'application/vnd.mapbox-vector-tile'.

All the styles are retrieved from the GeoServer REST administration API and are listed in the client in three formats: SLD, MBStyle or GeoCSS. When the application is rendering client-side, SLD and MBStyle are directly converted to OpenLayers Style while GeoCSS format is requested as SLD from GeoServer. The GeoServer REST administration API exposes all configuration elements, including styles, stores, and layers, and allows other applications to programmatically retrieve and alter the GeoServer setup (admin level authentication required).

TIP

Link to demo repository: <https://github.com/geosolutions-it/ogc-vector-tiles-vtp/tree/master/MapStoreStyle>

link to live demo: <http://vtp2018.s3-eu-west-1.amazonaws.com/vtpevt-wms.html/#>

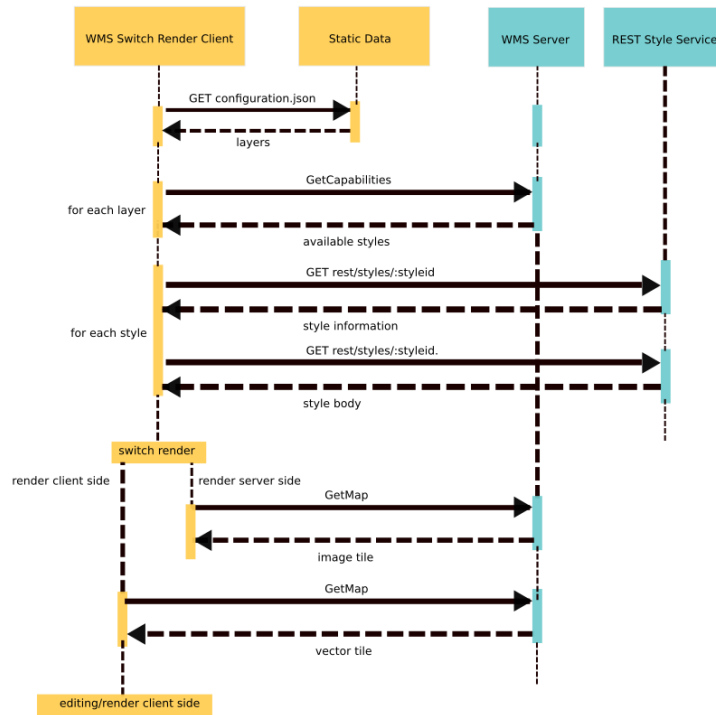


Figure 59. WMS - Client Sequence Diagram

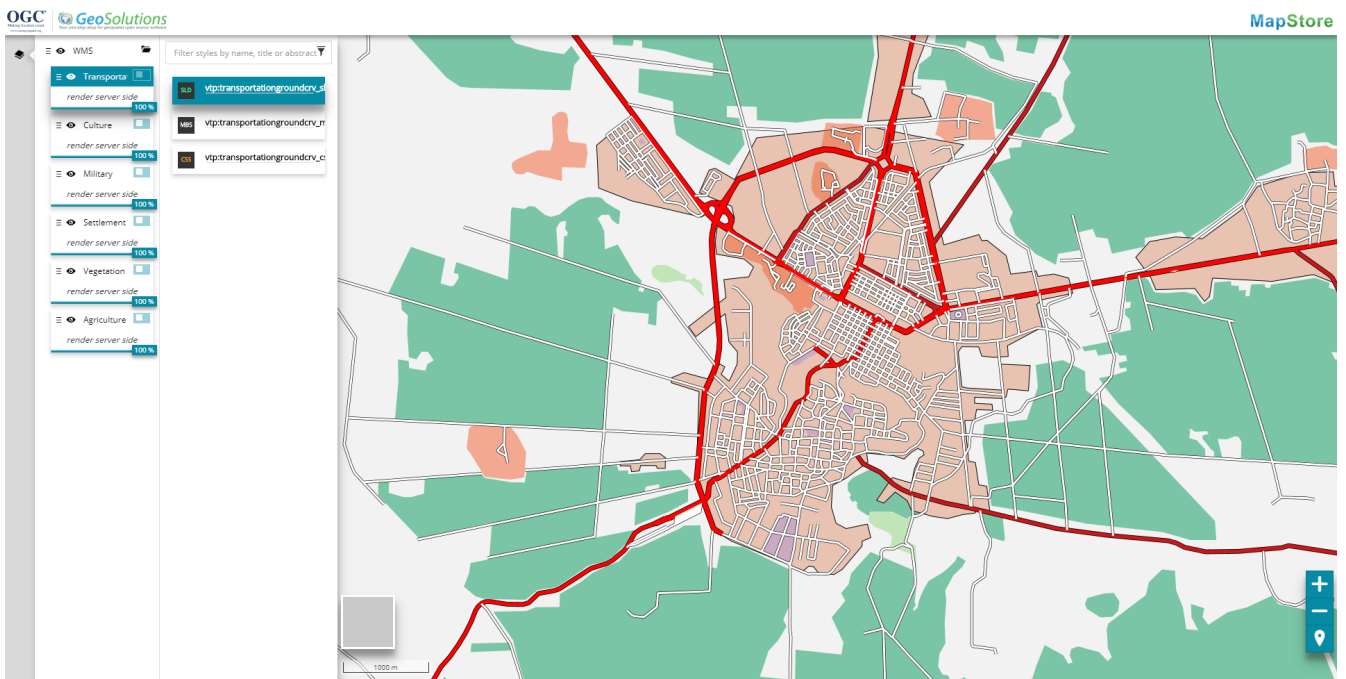


Figure 60. MapStore Client Shows Tiles Rendered Server Side of Transportation Ground Layer with WMS Tiles

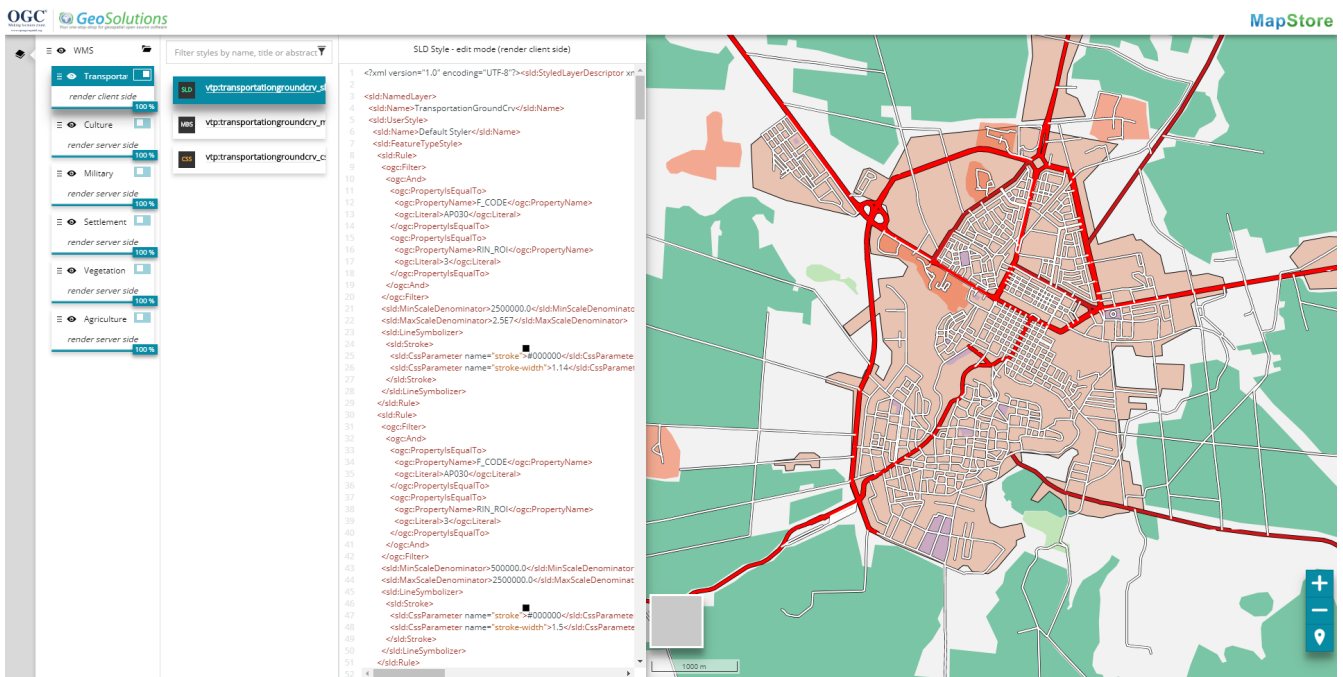


Figure 61. MapStore Client Render Tiles Client Side of Transportation Ground Layer with WMS Vector Tiles

Ecere

Ecere’s GNOSIS Cartographer was used as client to display tiled feature data from either its native GNOSIS data store, WMTS, WFS3, or GeoPackages supporting the extensions developed for the VTP/VTPExt. The tool is capable of displaying the feature data in either 3D (optionally draped onto 3D terrain) or cartographic projections, and it can dynamically apply styles in real-time. Support for importing styles in the Mapbox GL styles definition format was developed during the VTPEExt. The Ecere client can now import styles from Mapbox GL styles, SLD/SE, and its GNOSIS Cartographic Map Style Sheets. In addition to the WMTS and WFS interoperability experiments described above, GeoPackages produced by Compusult, CubeWerx, and Ecere were successfully visualized, using the different styles.

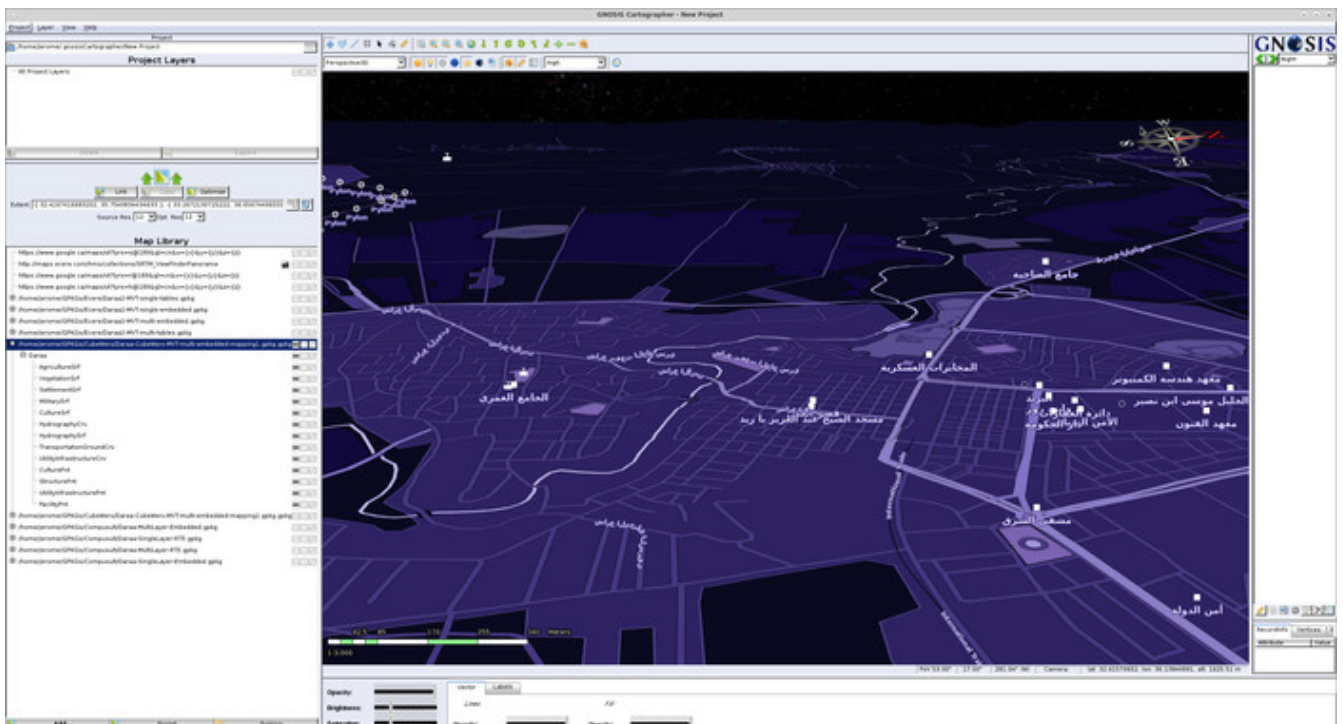


Figure 62. Ecere client visualizing CubeWerx GeoPackage using the Night style

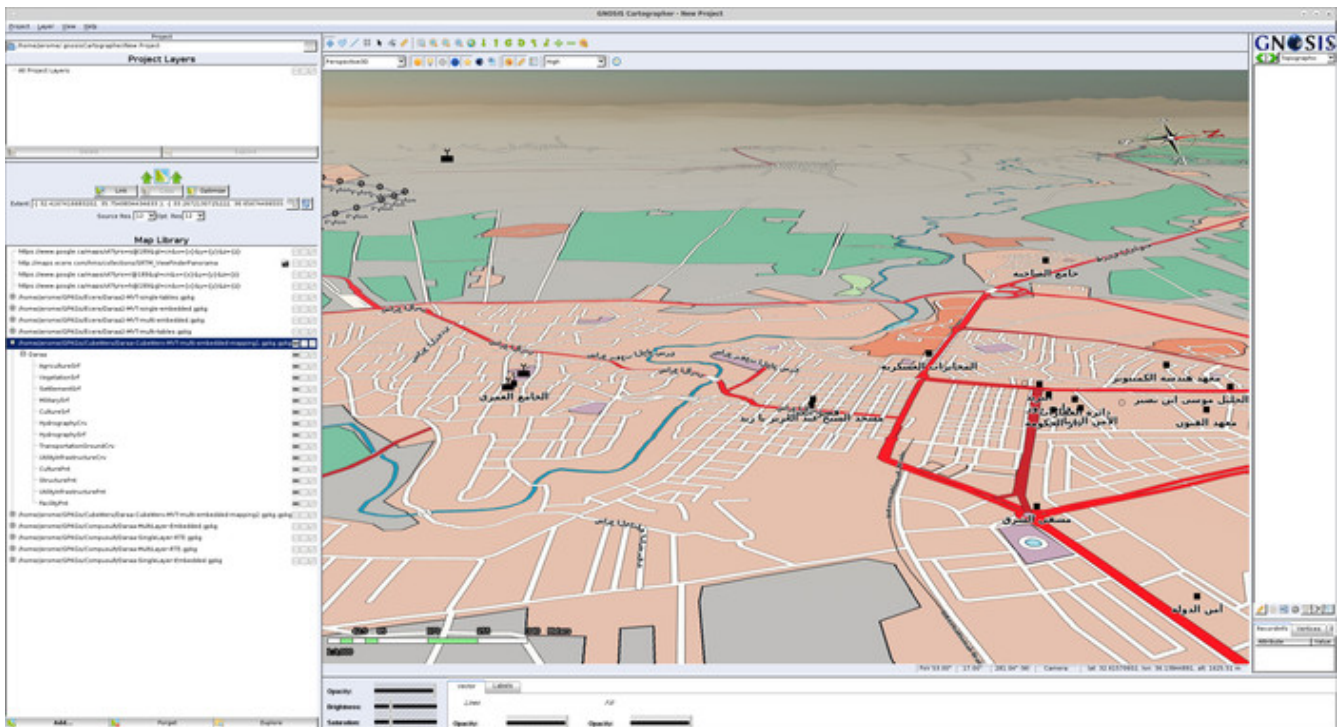


Figure 63. Ecere client visualizing CubeWerx GeoPackage using the Topographic style

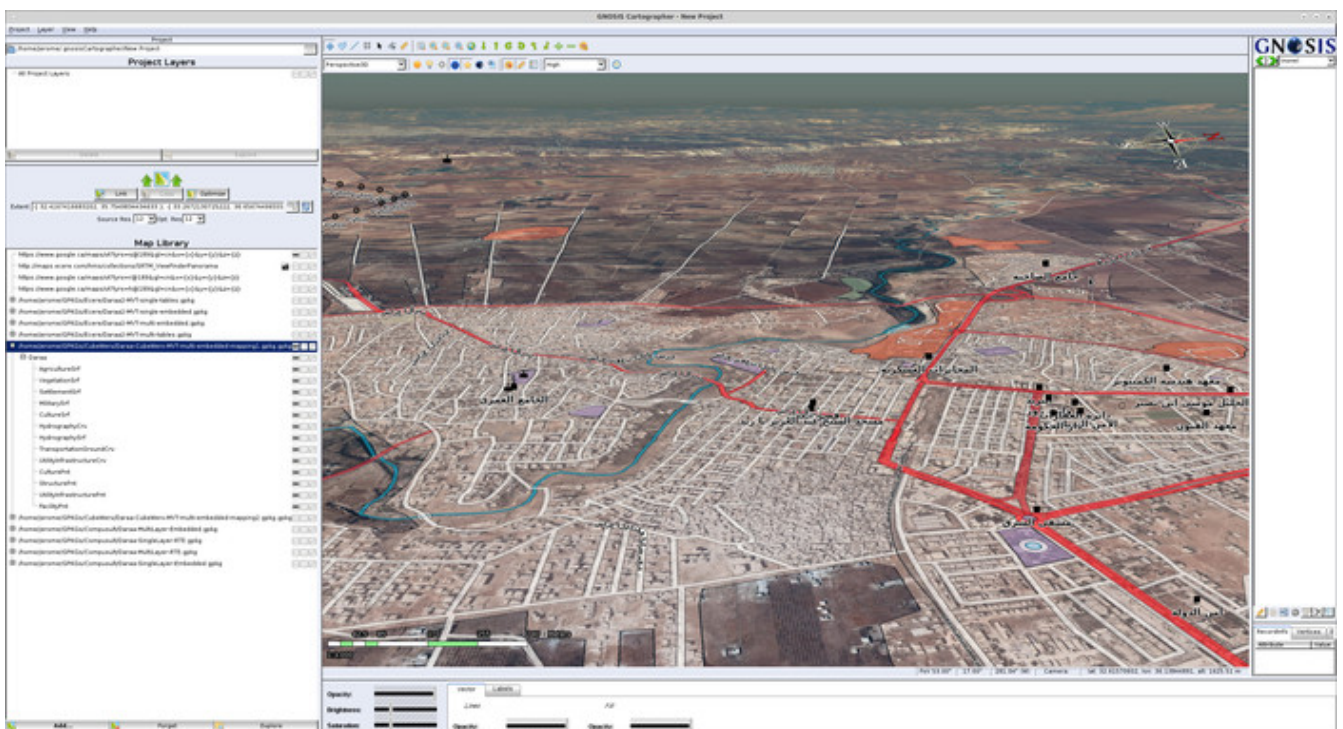


Figure 64. Ecere client visualizing CubeWerx GeoPackage using the Overlay style (Google Maps data underneath)

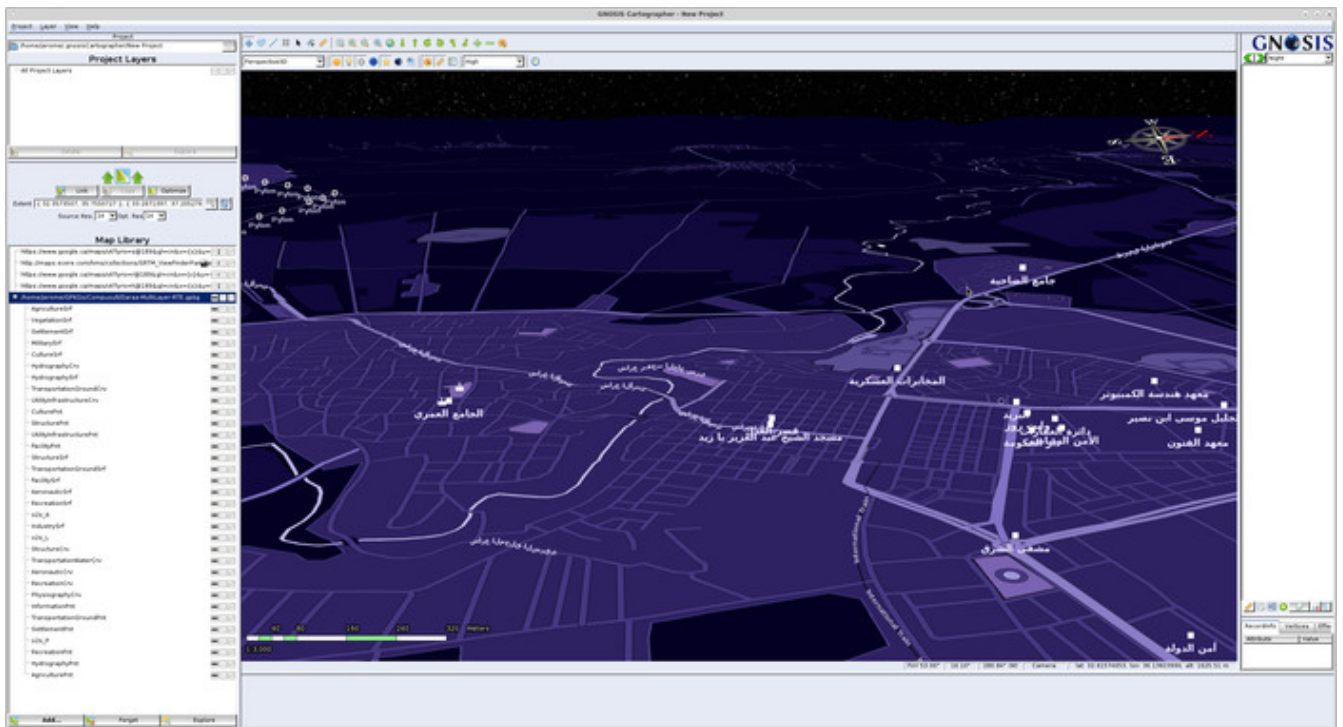


Figure 65. Ecere client visualizing Compusult GeoPackage using the Night style

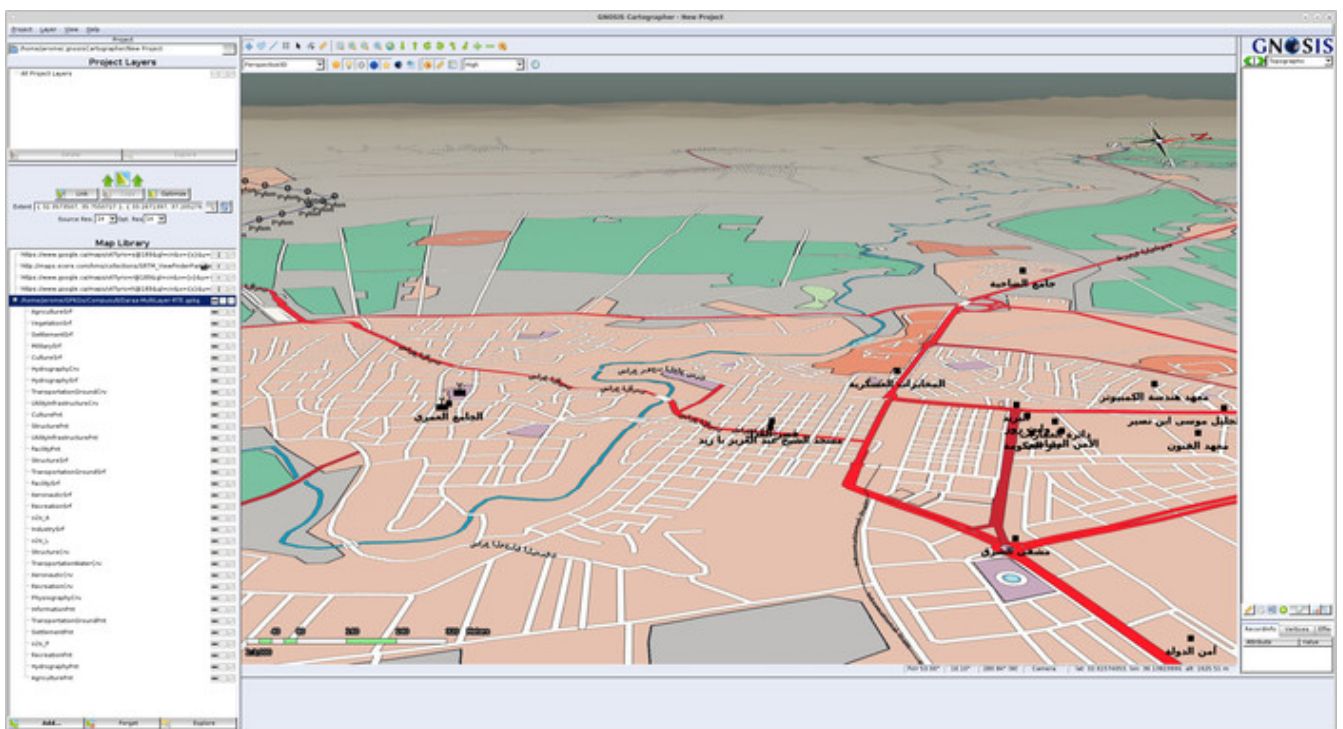


Figure 66. Ecere client visualizing Compusult GeoPackage using the Topographic style

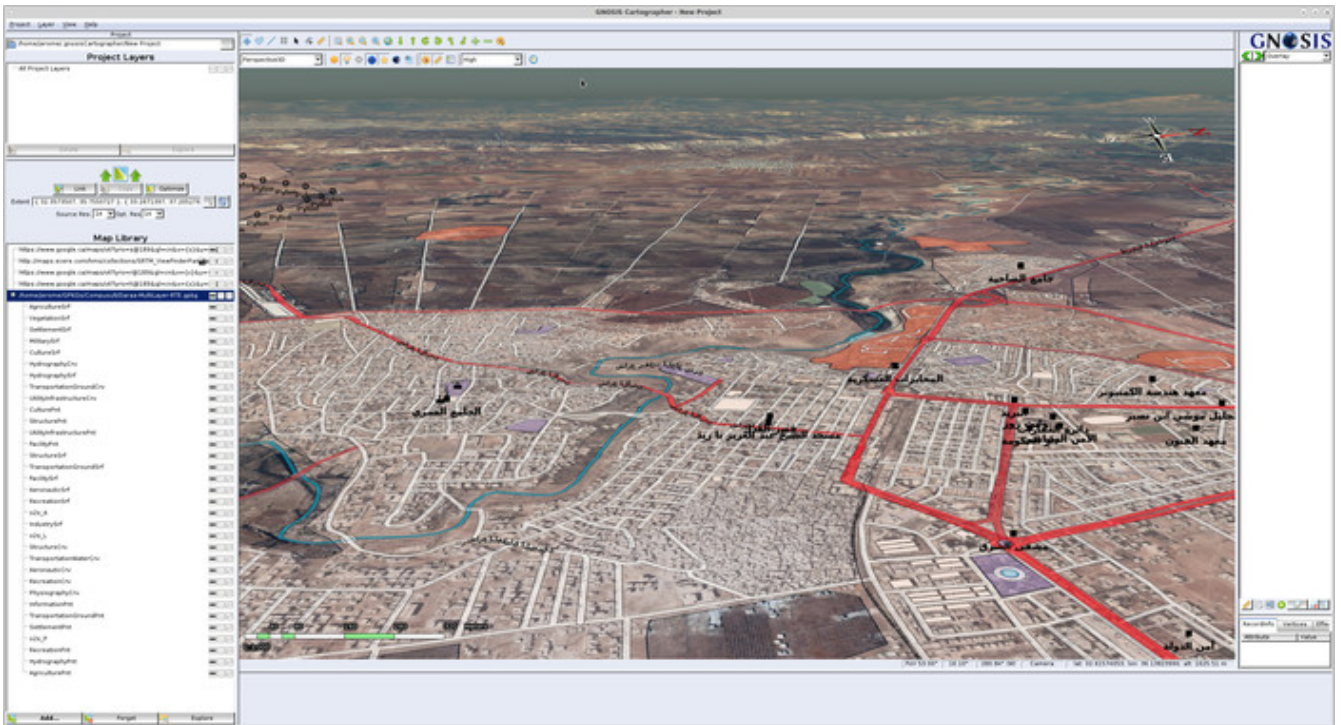


Figure 67. Ecere client visualizing *Compusult GeoPackage* using the *Overlay* style (Google Maps data underneath)

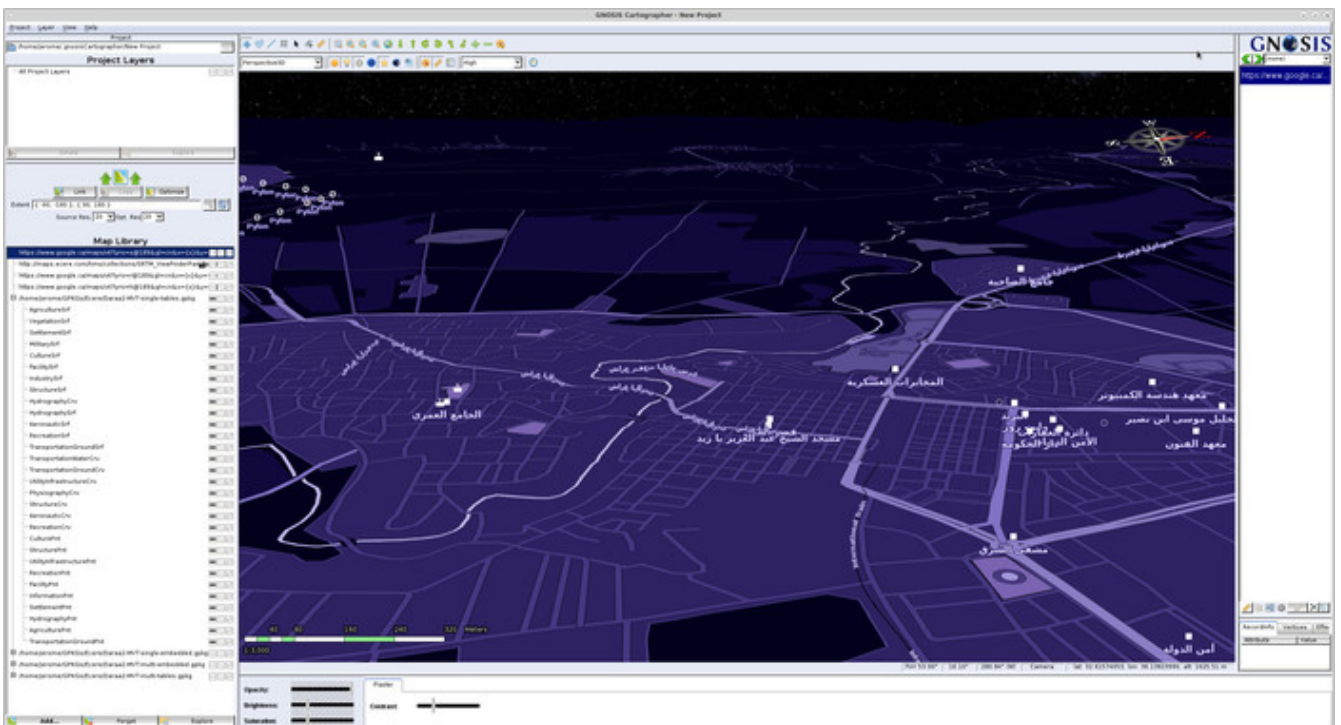


Figure 68. Ecere client visualizing *Ecere GeoPackage* using the *Night* style

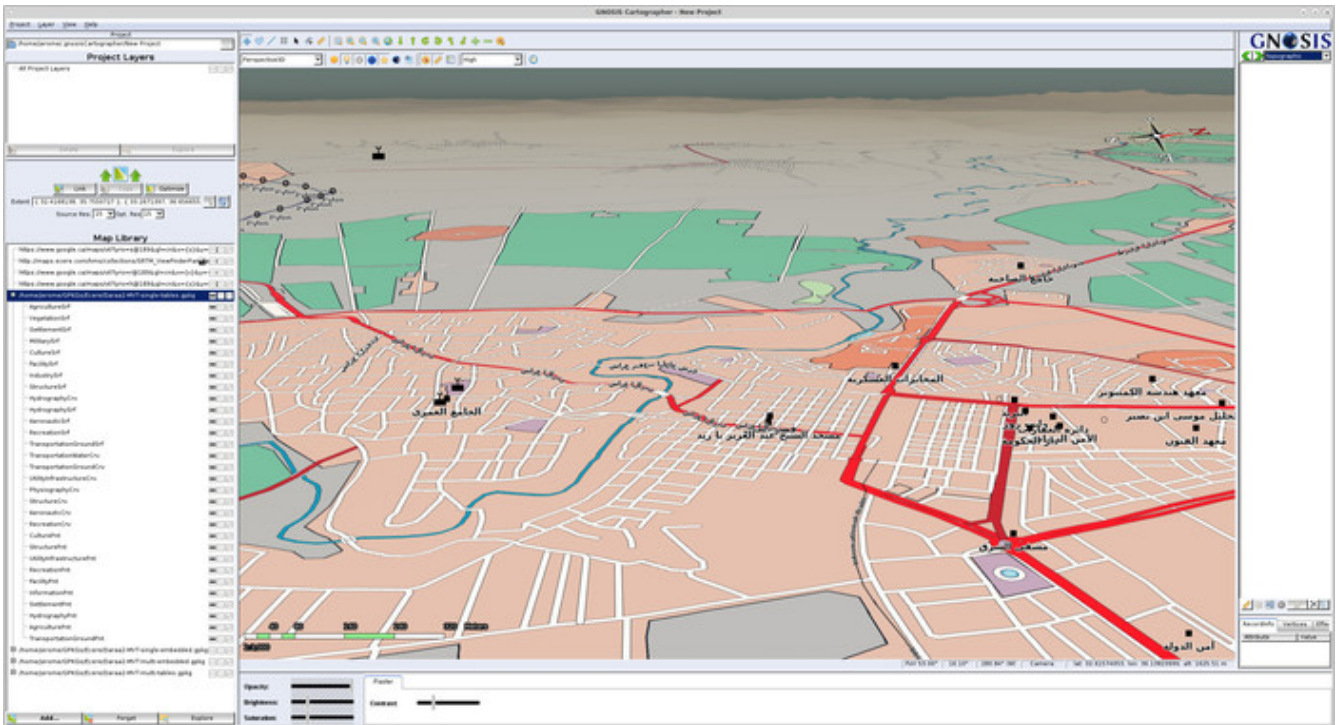


Figure 69. Ecere client visualizing Ecere GeoPackage using the Topographic style

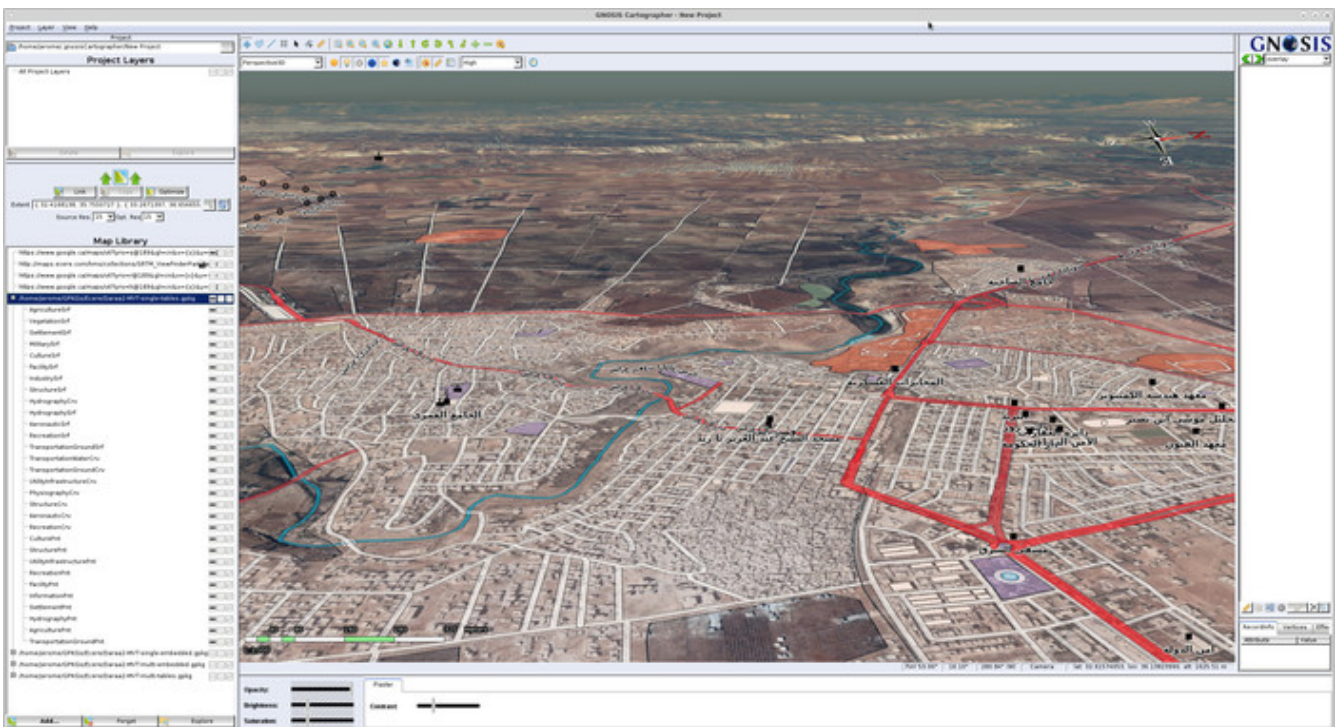


Figure 70. Ecere client visualizing Ecere GeoPackage using the Overlay style (Google Maps data underneath)

Image Matters

Mobile Client

The Experience Reality mobile client, among other functions, can render and display GeoPackages on a map view. This client is capable of rendering GeoPackages with conventional feature tables styled with CartoCSS and tiled feature data styled with the Mapbox style specification. There are trade-offs to each approach, with respect to use on a mobile device. To render and display features from a GeoPackage with enumerated feature tables, the client reads the geometry from the table,

reads the CartoCSS styling information corresponding to that table, and draws the features directly on the map in the correct location. Each layer can be enabled and disabled individually. This is very quick and responsive. However, this keeps the rendered features in memory, of which a mobile device has precious little. Drawing too many features simultaneously will severely slow down the mobile client, and possibly crash the application, due to running out of memory. This is a common occurrence when there are a few dozen feature layers in the same viewport.

To address this issue, the mobile client also supports tiled feature data, which is handled differently. To render and display this data, each layer has its styling information parsed beforehand, and both are aggregated into an object called a Tile Provider. While enabled, the Tile Provider will render and display tiles that are within the viewport of the map view. The tile is rendered into an image, and stored into a cache on the local disk, so it is only rendered once. The cached image is displayed on the map as it is requested. This significantly improves the performance of the mobile client while all of the features are enabled, as there is only one image to display per tile, and it is cached on the local disk. However, this means the ability to enable and disable each layer individually is lost. To allow that feature, there would need to be a separate Tile Provider for each individual layer, which would reintroduce the memory issue, as there would be one image per layer enabled per tile.

To summarize, a non-Vector Tile approach renders more easily, but has performance issues while displaying on a mobile client. Using tiled feature data makes rendering more difficult and virtually prohibits enabling individual layers, but displays far more smoothly.

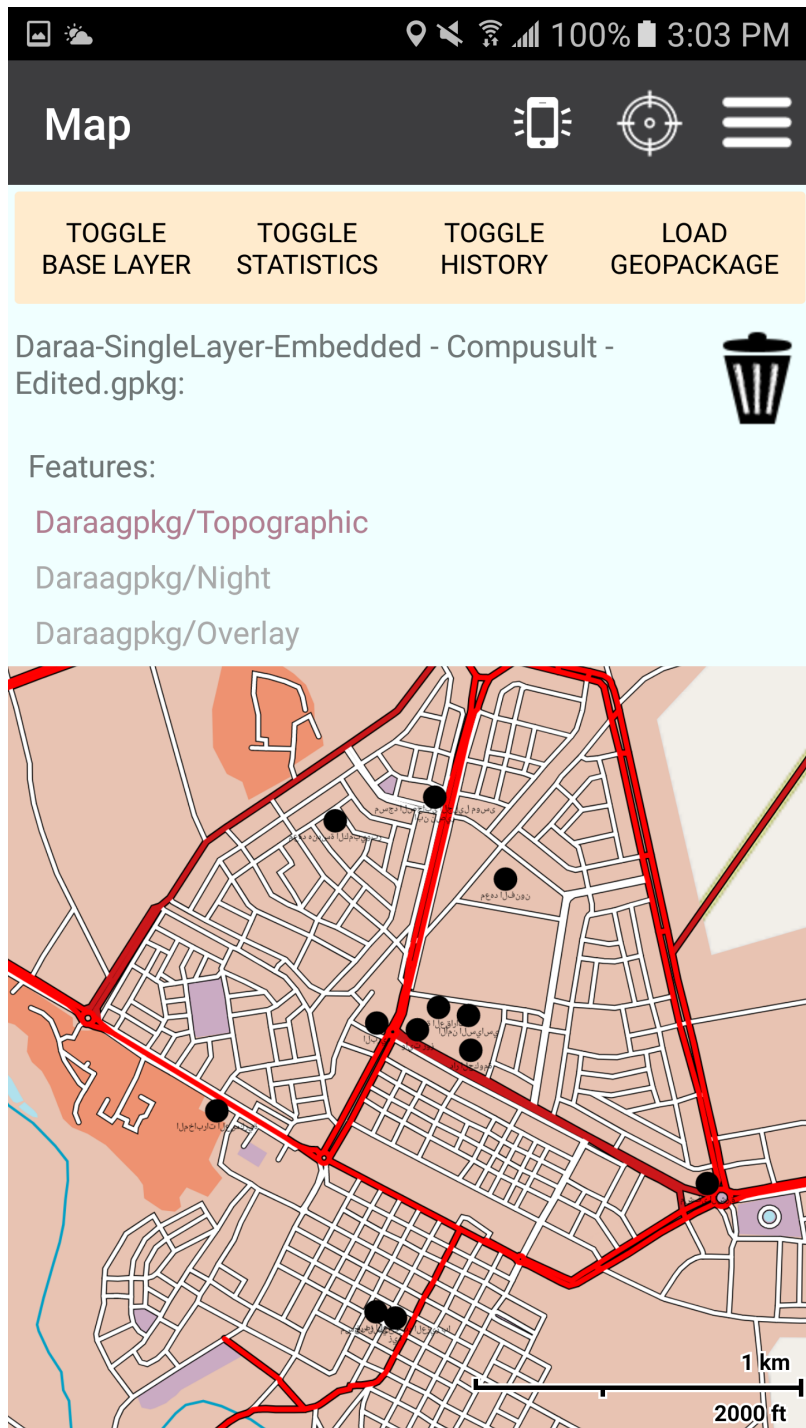


Figure 71. Image Matters Mobile Client rendering Compusult GeoPackage with Topographic style

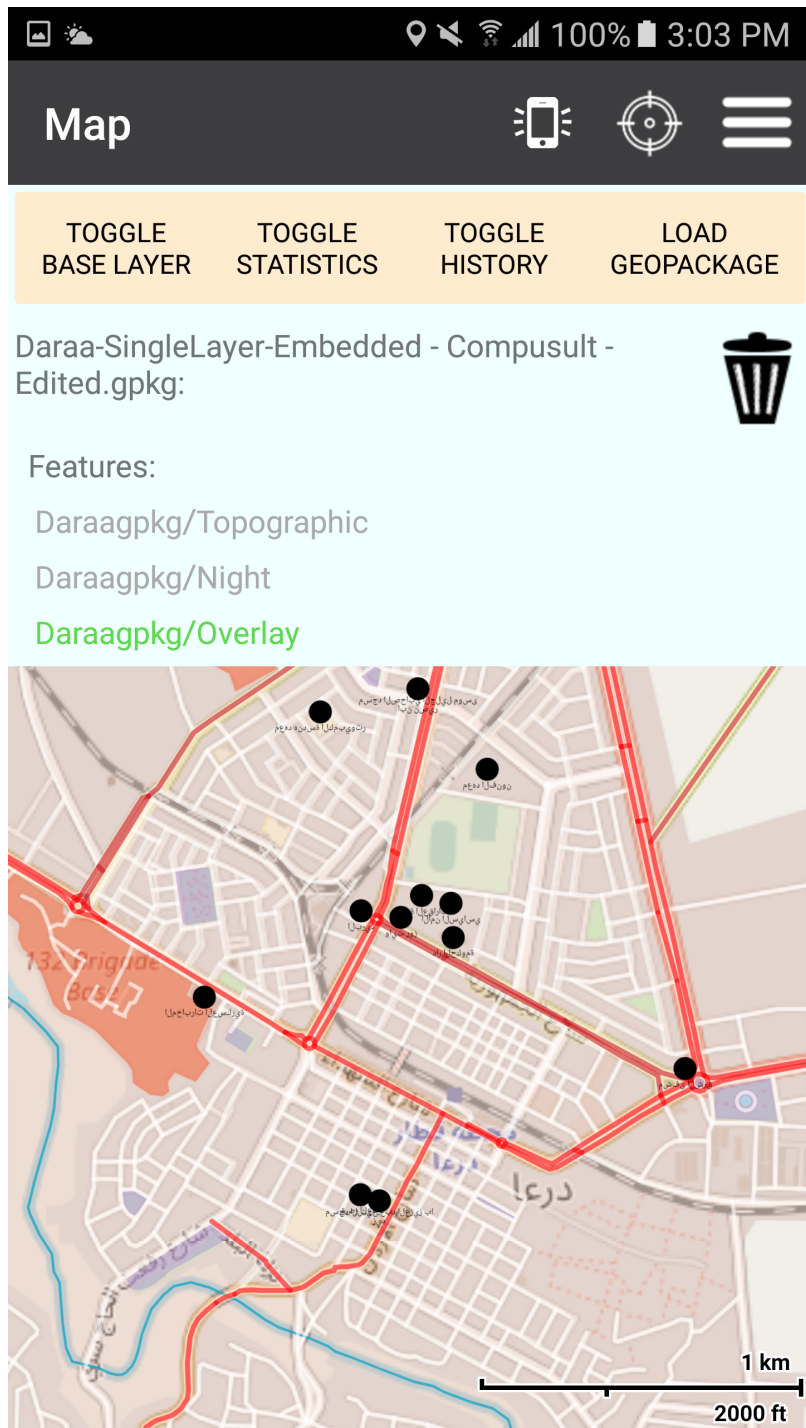


Figure 72. Image Matters Mobile Client rendering Compusult GeoPackage with Overlay style

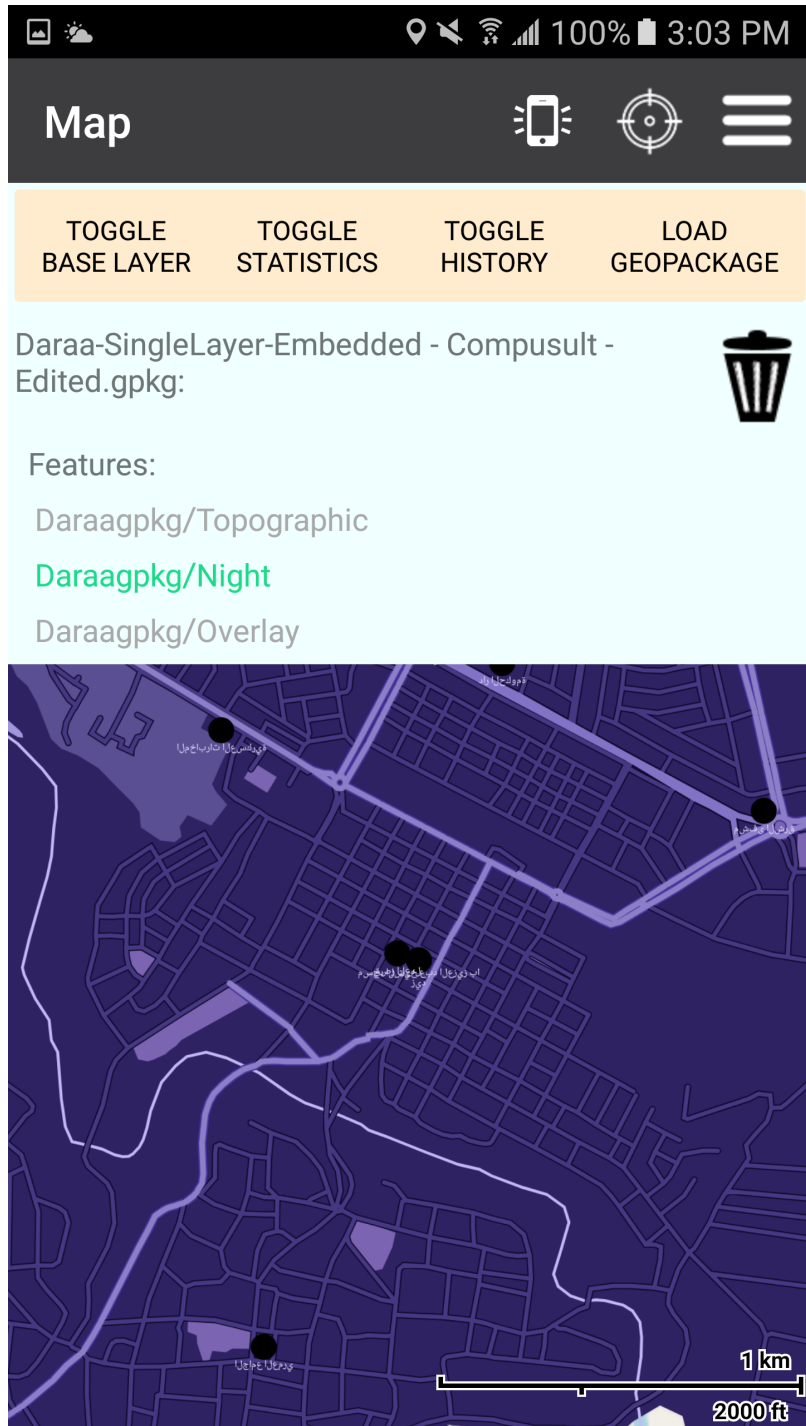


Figure 73. Image Matters Mobile Client rendering Compusult GeoPackage with Night style

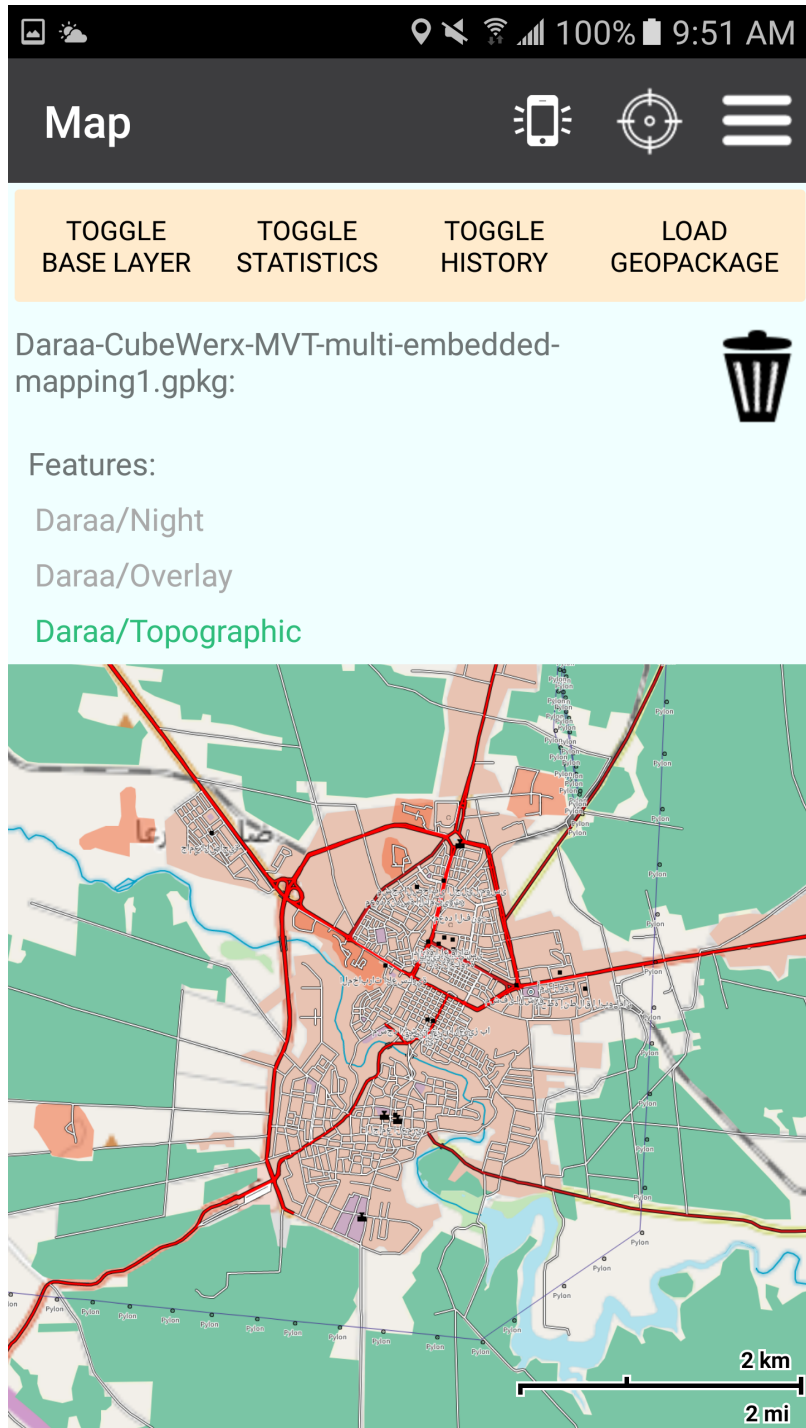


Figure 74. Image Matters Mobile Client rendering CubeWerx GeoPackage with Topographic style

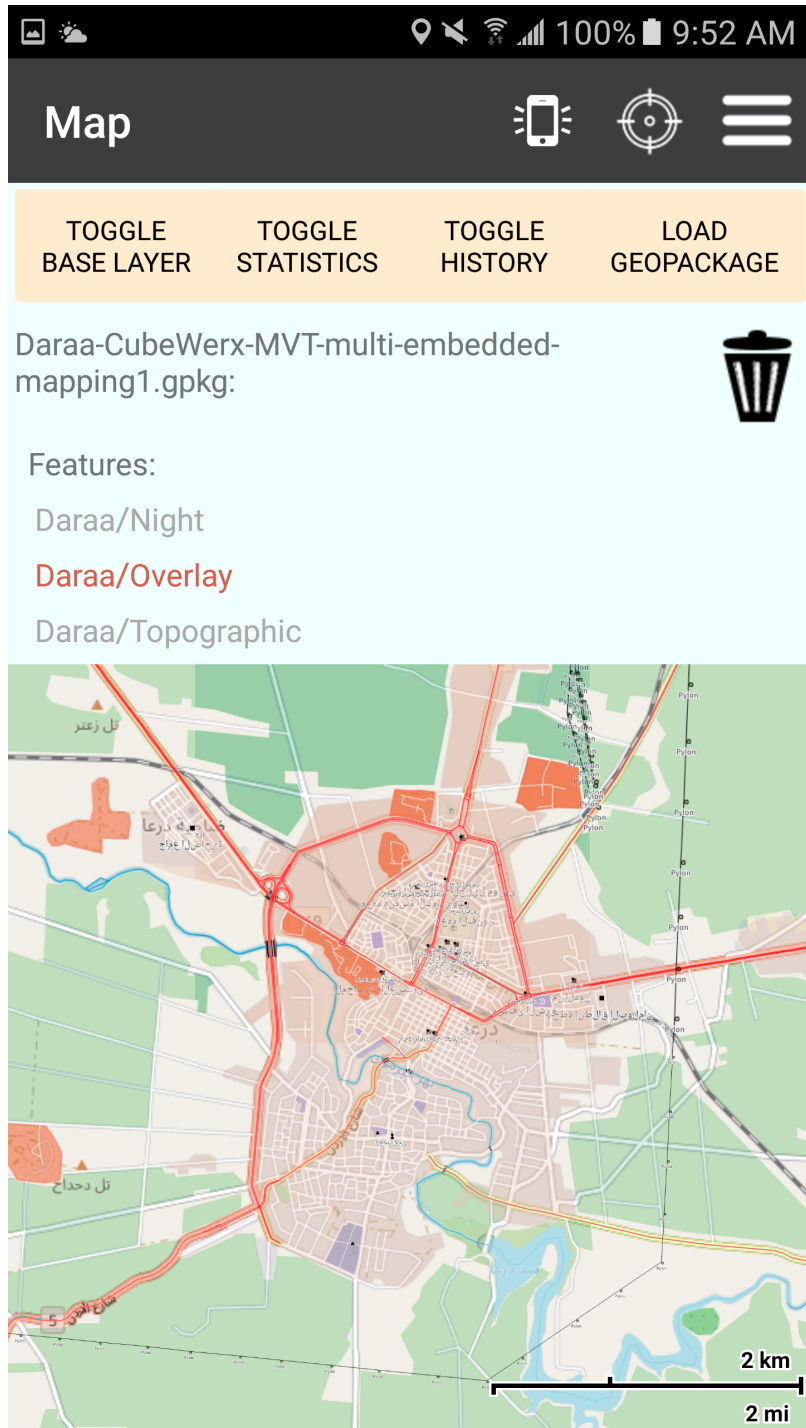


Figure 75. Image Matters Mobile Client rendering CubeWerx GeoPackage with Overlay style

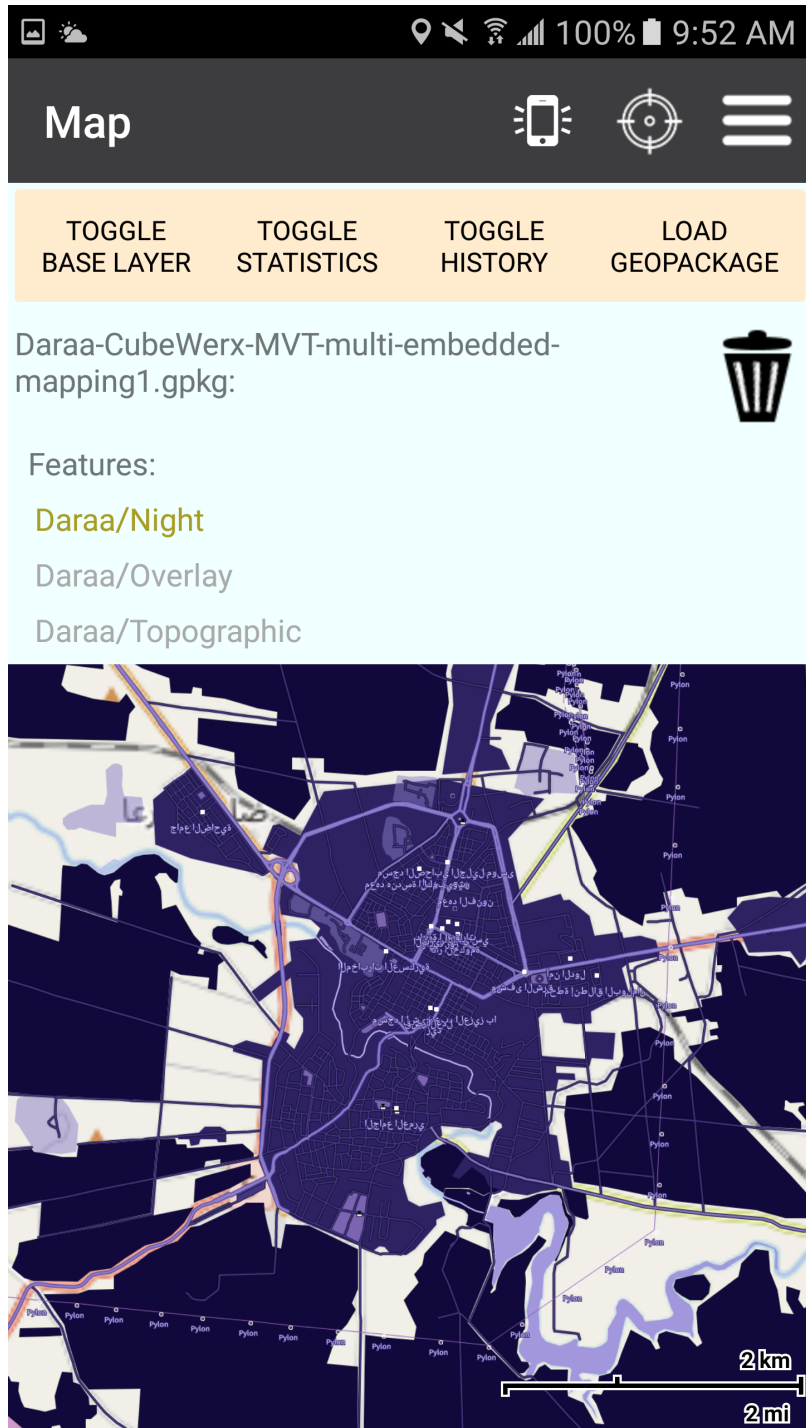


Figure 76. Image Matters Mobile Client rendering CubeWerx GeoPackage with Night style

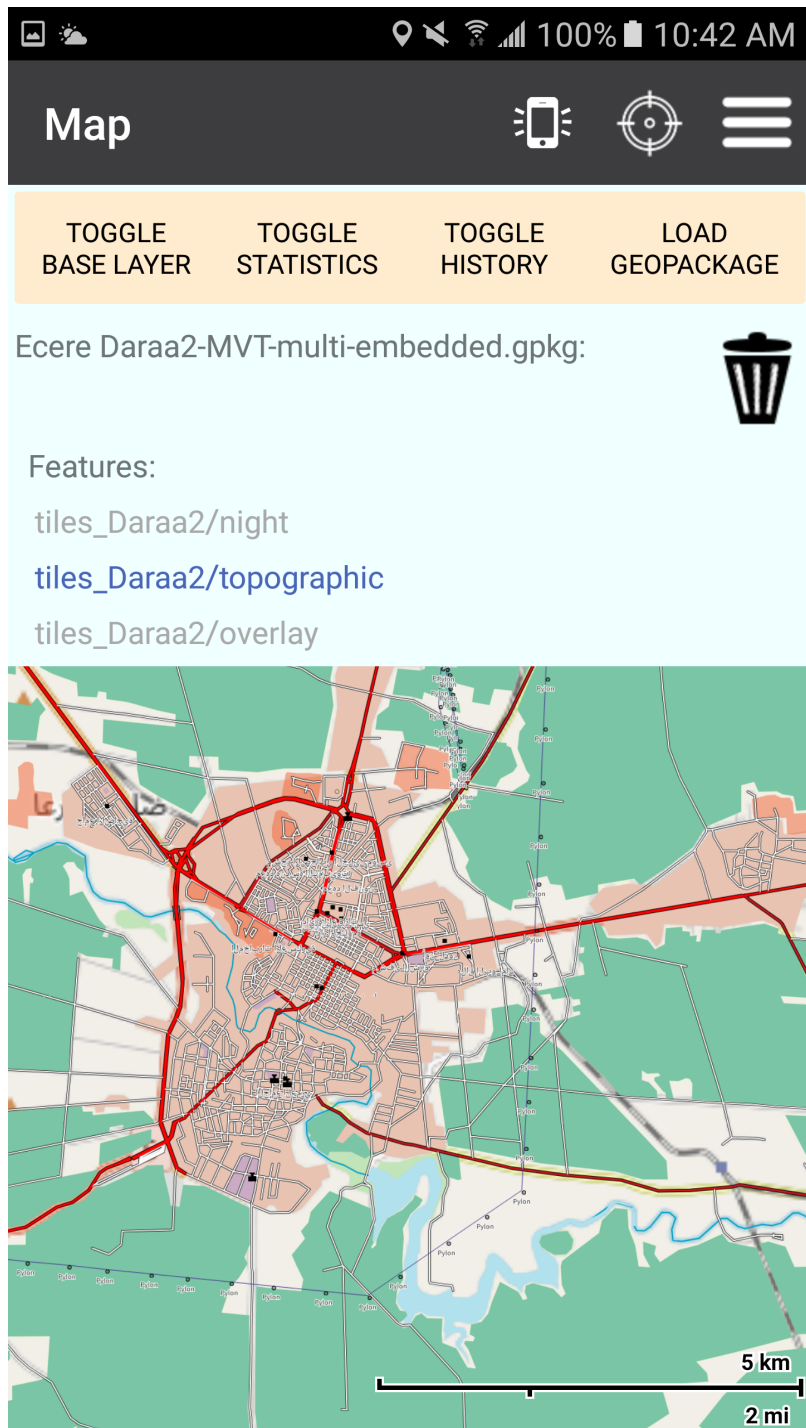


Figure 77. Image Matters Mobile Client rendering Ecere GeoPackage with Topographic style

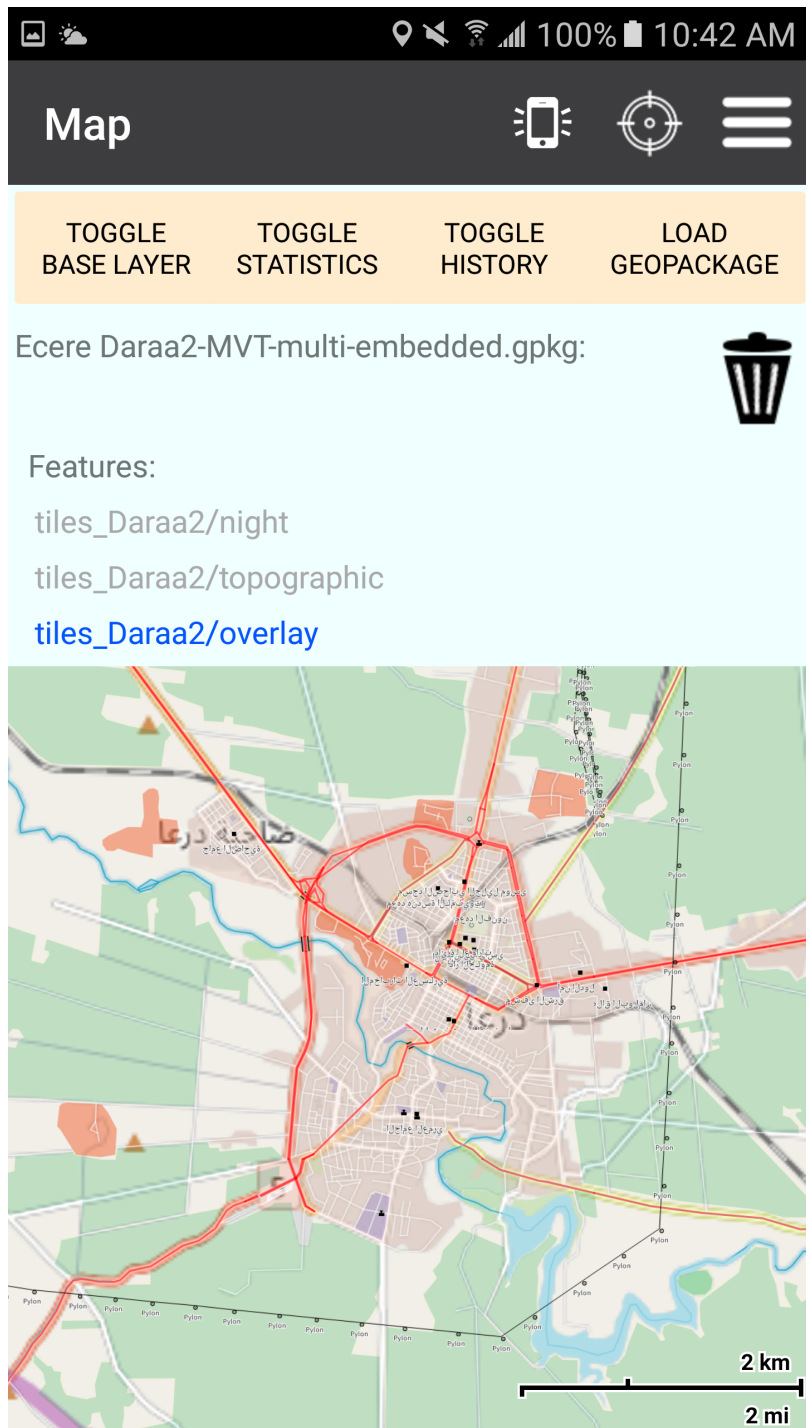


Figure 78. Image Matters Mobile Client rendering Ecere GeoPackage with Overlay style

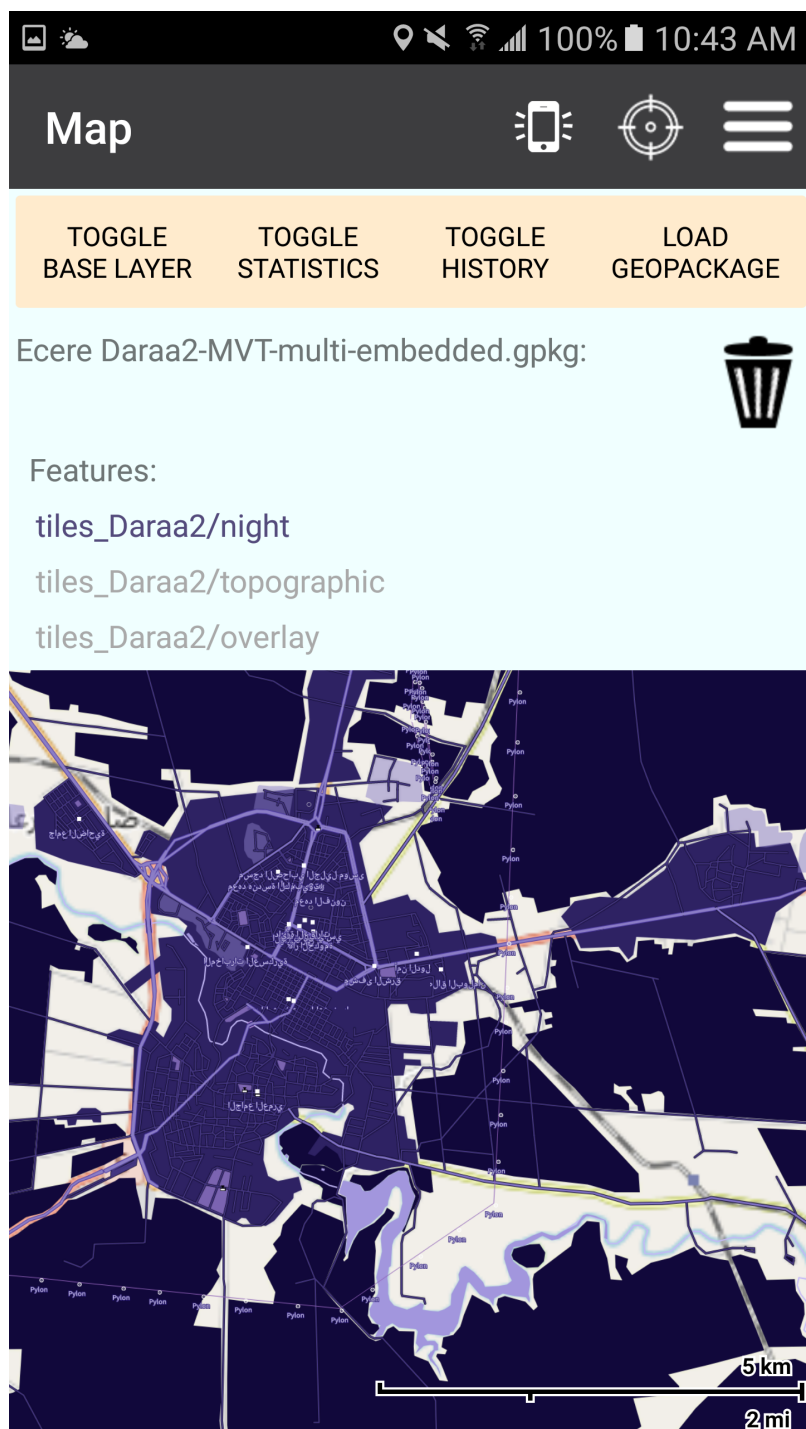


Figure 79. Image Matters Mobile Client rendering Ecere GeoPackage with Night style

Compusult GOMobile Client

The Compusult GOMobile client retrieves GeoPackages, GeoPackage services, and WMTS services from a CSW which has this content published in it. WMTS services are rendered using GetTile/GetStyle operations or using the RESTful equivalent. SLD and Mapbox styles are parsed and converted to an abstract styling model which is used to render vector tile content. GeoPackage style rendering supports both embedded feature attributes and attributes stored using the GeoPackage Vector Tile Attribute extension. Clients are able to specify the tile format and style format for the most flexible solution. Mapbox Vector Tile and GeoJSON tile formats are supported, along with Mapbox Style and SLD style formats. Style documents are cached to ensure documents are not requested multiple times when dealing with a multi-layer GeoPackage which leverages from the same style document. When attributes are stored using the GeoPackage Related Tables Extension, it

allows for simple editing of features, which leads to the ability to dynamically style the tiled feature data.

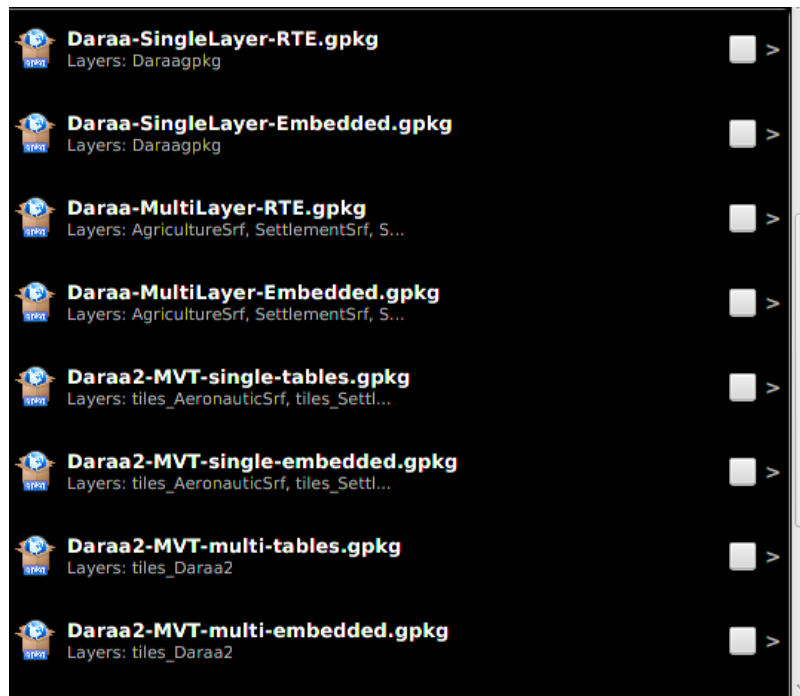


Figure 80. Compusult GOMobile GeoPackage Binding



Figure 81. Compusult GOMobile GeoPackage Style Selection

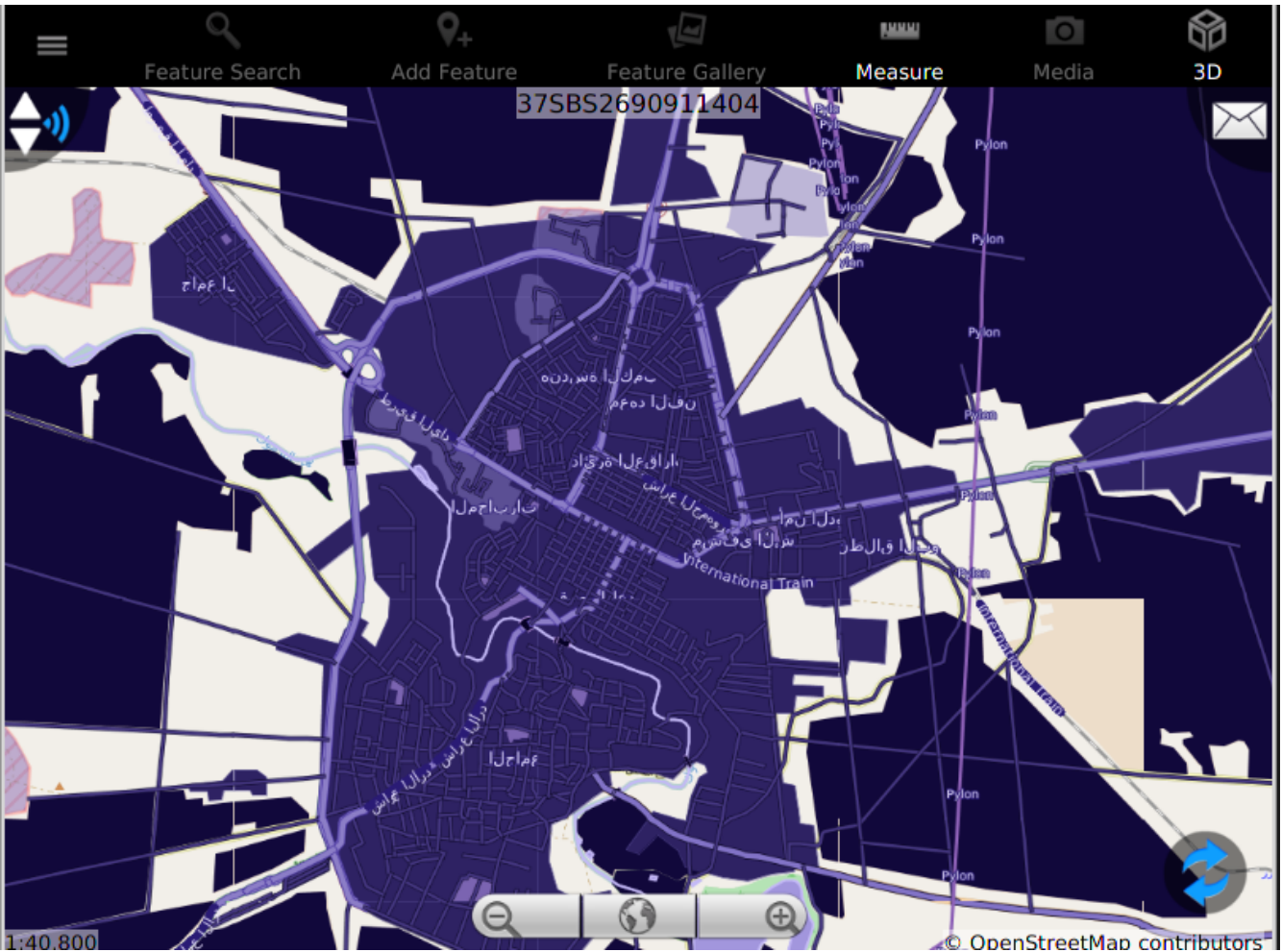


Figure 82. Compusult GOMobile GeoPackage Ecere Single MVT Tileset/Embedded Attributes (Night Style SLD)

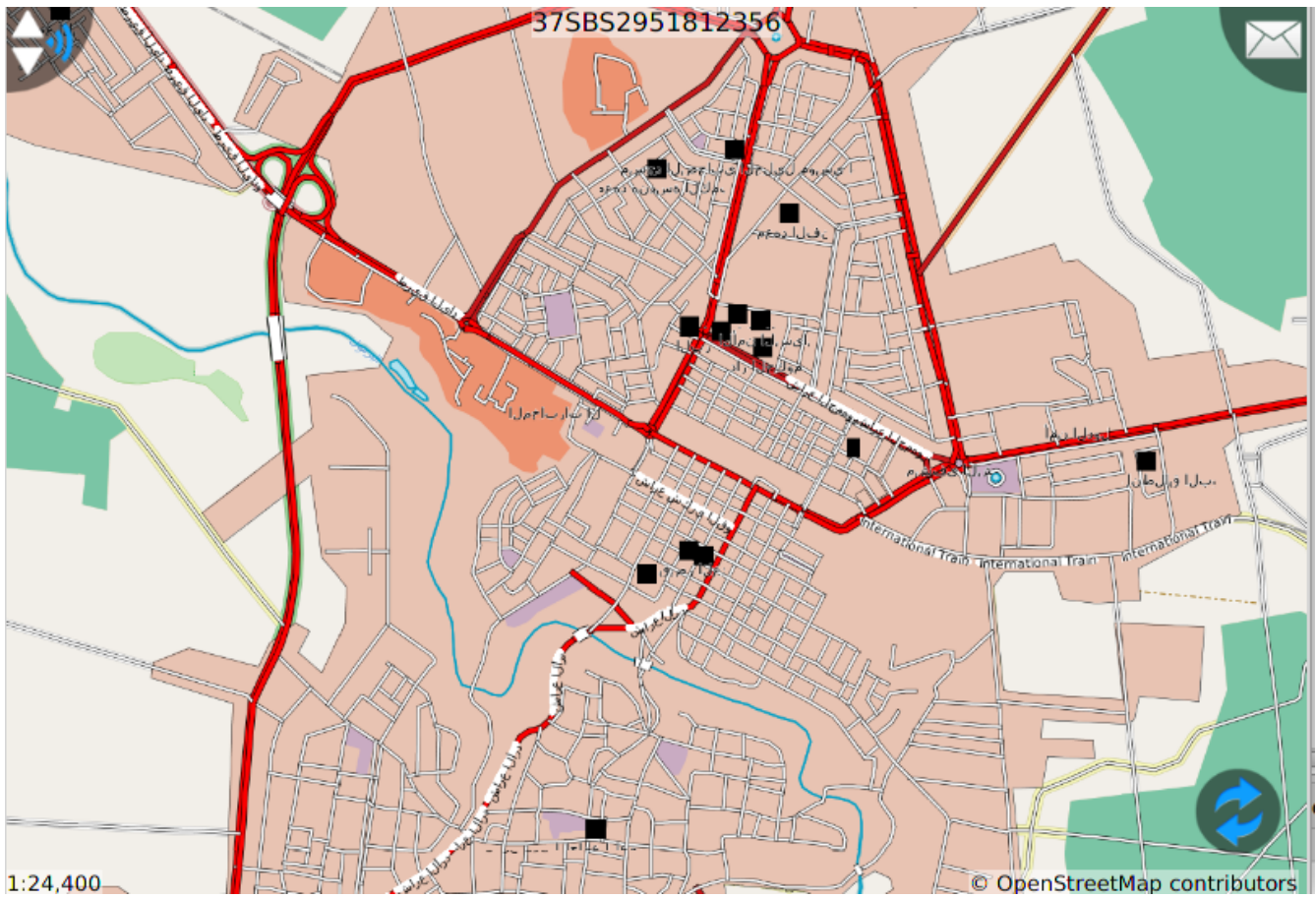


Figure 83. Compusult GOMobile GeoPackage Multi-Tileset MVT/Attributes Extension (Topographic Style Mapbox)

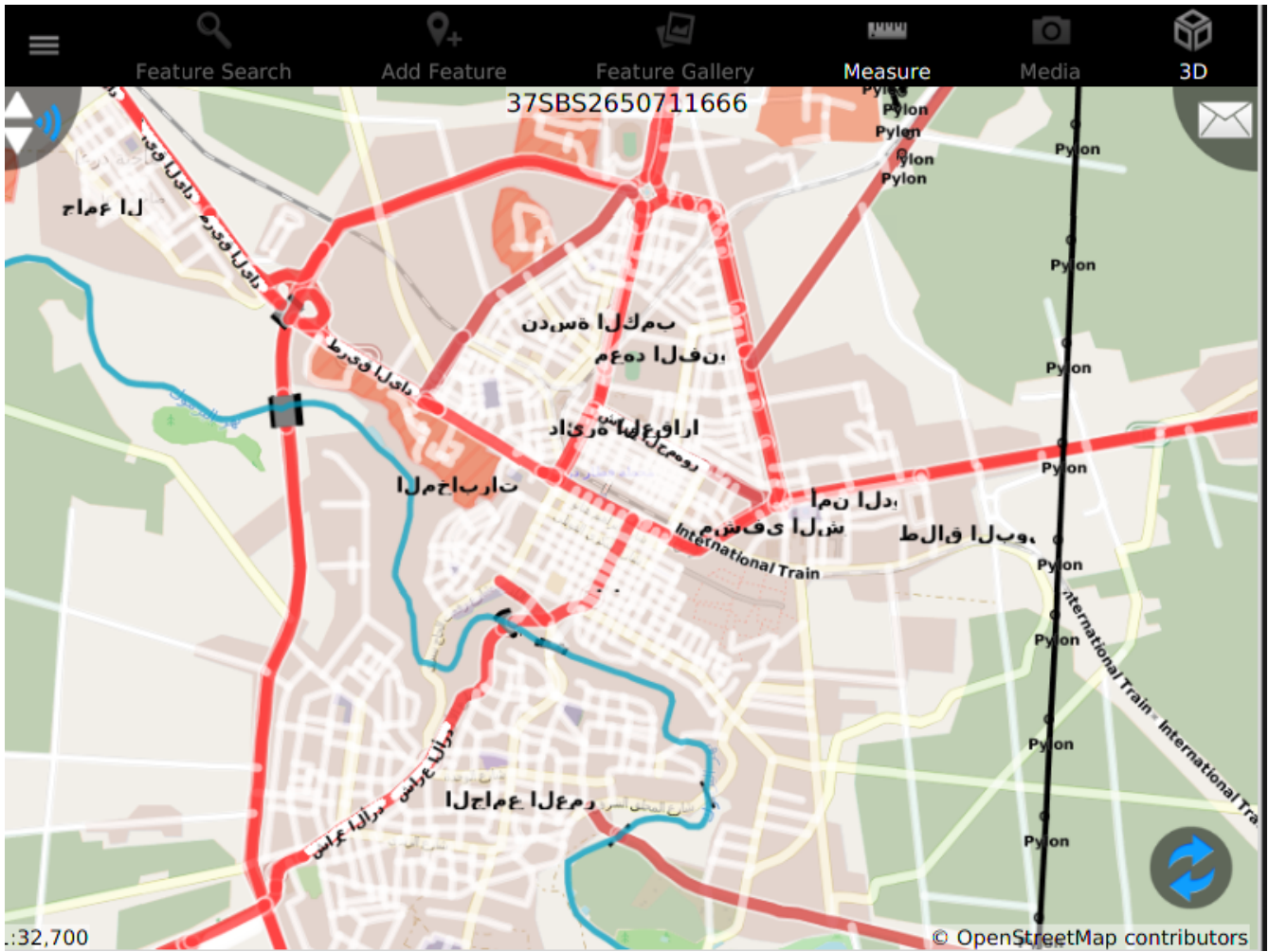


Figure 84. Compusult GOMobile GeoPackage Single Tileset GeoJSON (Overlay Style Mapbox)

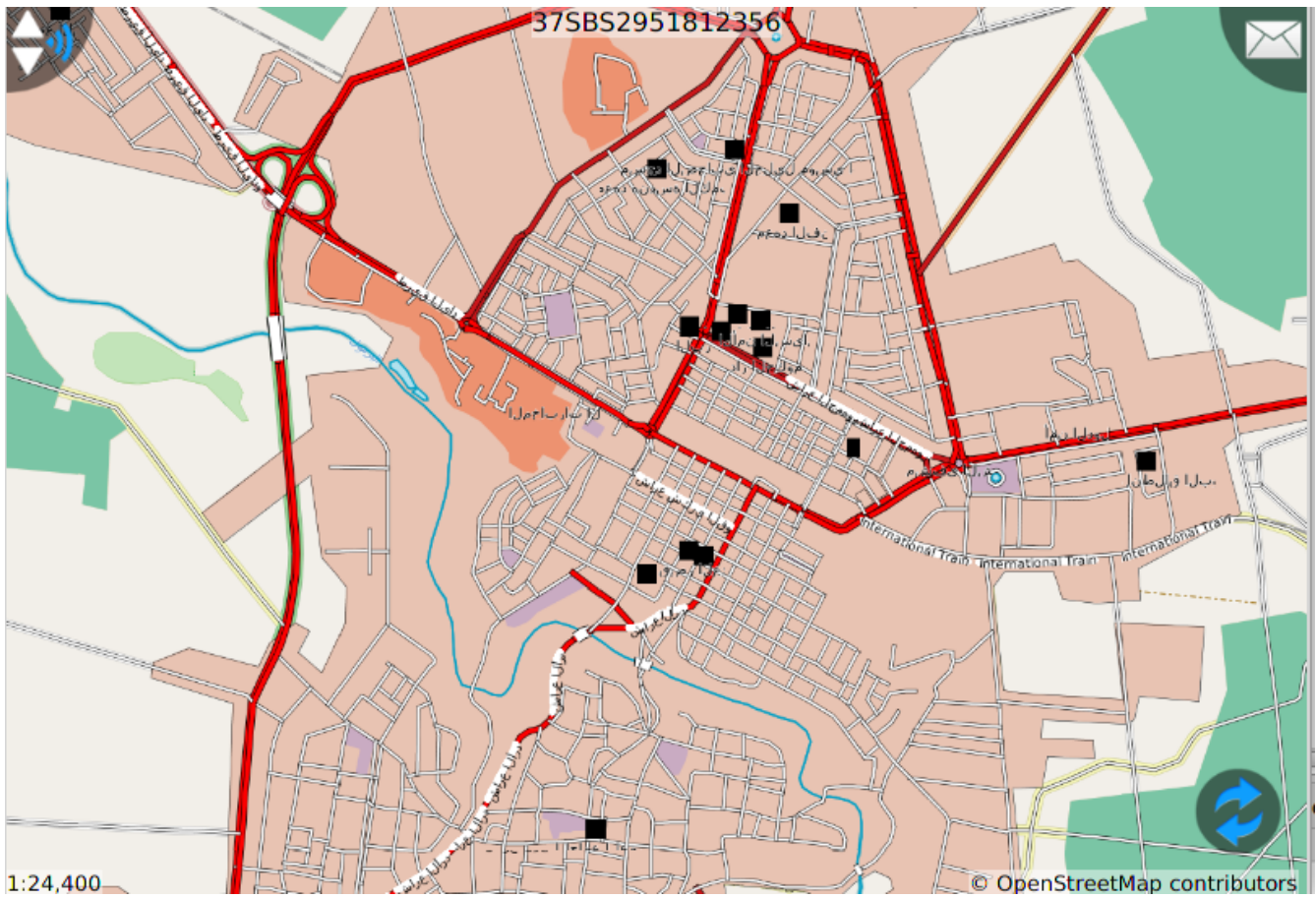


Figure 85. Compusult GOMobile GeoPackage CubeWerx Single Tileset MVT/Embedded Attributes (Topographic Style Mapbox)



Figure 86. CompuSult GOMobile GeoPackage CubeWerX Single Tileset MVT/Embedded Attributes (Overlay Style Mapbox)

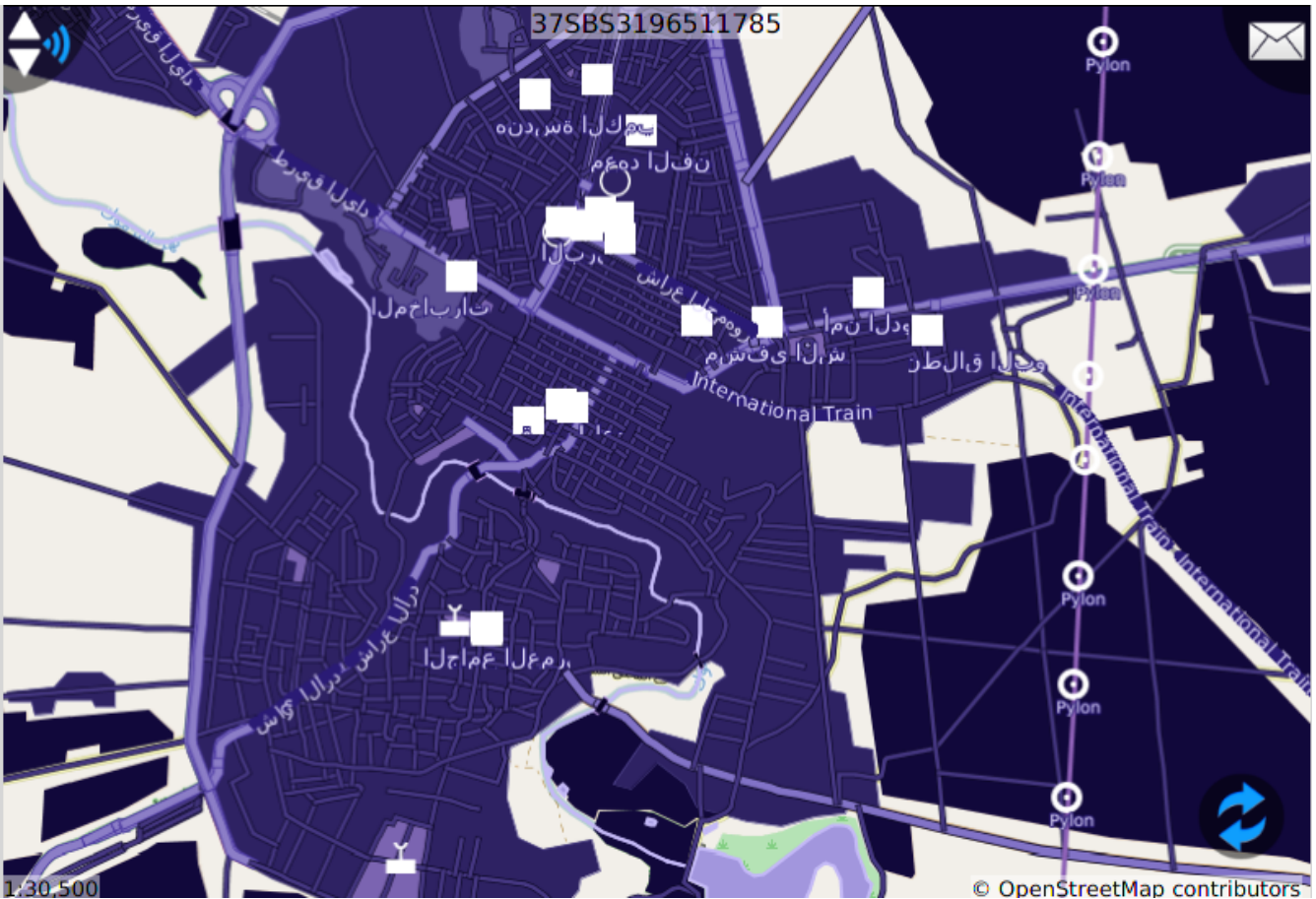


Figure 87. Compusult GOMobile GeoPackage CubeWerx Single Tileset MVT/Embedded Attributes (Night Style SLD)

7.4.6. Querying Tiled Feature Data

GeoSolutions

The GeoSolutions WMTS Simple client described above provides a tool to request feature information directly from the vector tiles. It is enabled by clicking on the marker button in the bottom right corner.

TIP | Link to live demo: <http://vtp2018.s3-eu-west-1.amazonaws.com/vtpevt-wmts.html/>

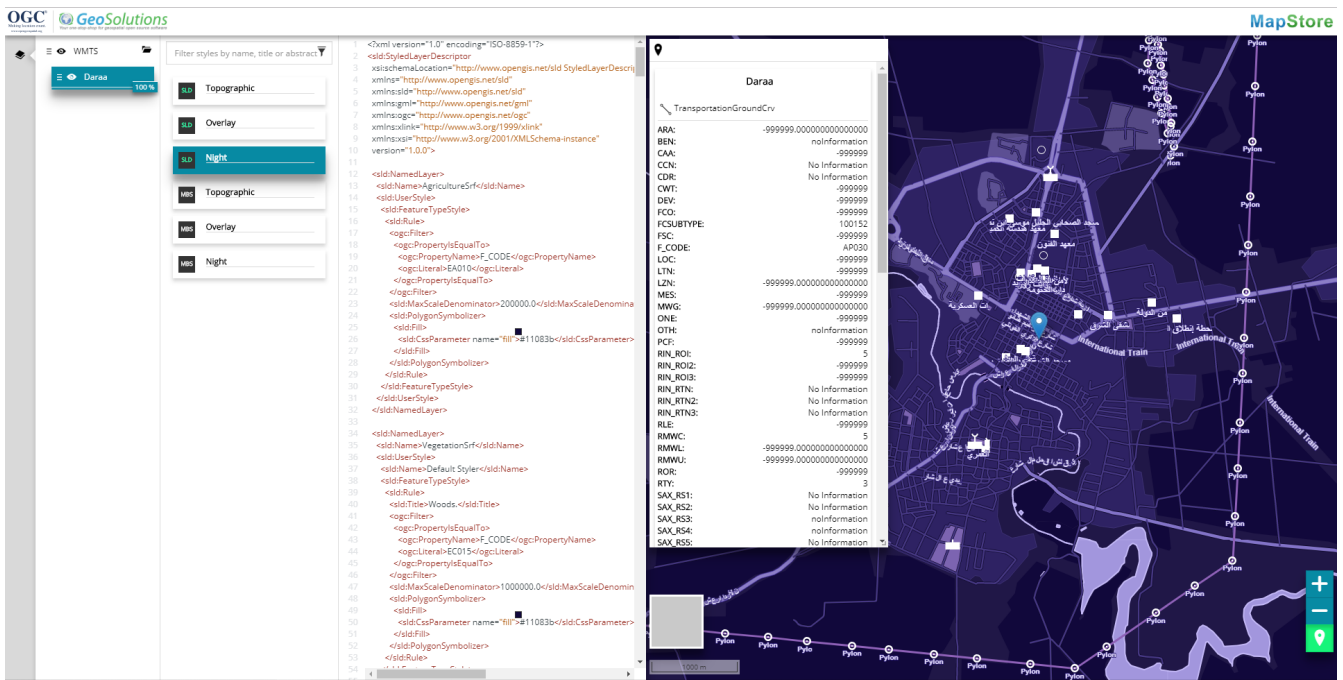


Figure 88. MapStore Client Get Feature Info from WMTS Vector Tiles

Compusult

The Compusult GOMobile client allows the user to click on the map and detect any feature information available in the rendered services. Both WMTS and GeoPackages which serve vector tile content can be queried and displayed to the client. GeoPackages with both embedded attributes and related tables attributes are supported.

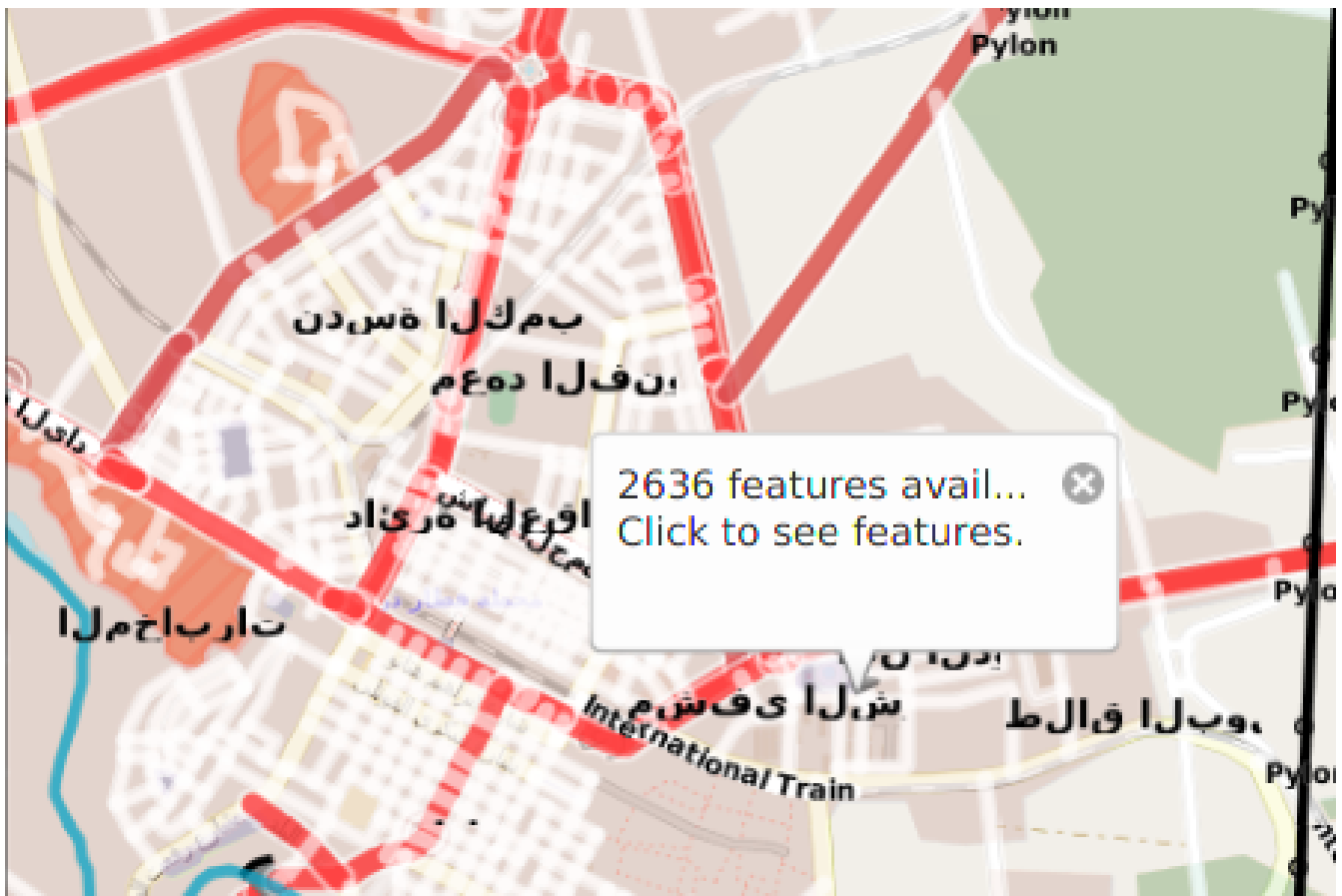


Figure 89. Compusult GOMobile Feature Selection (GeoPackage)

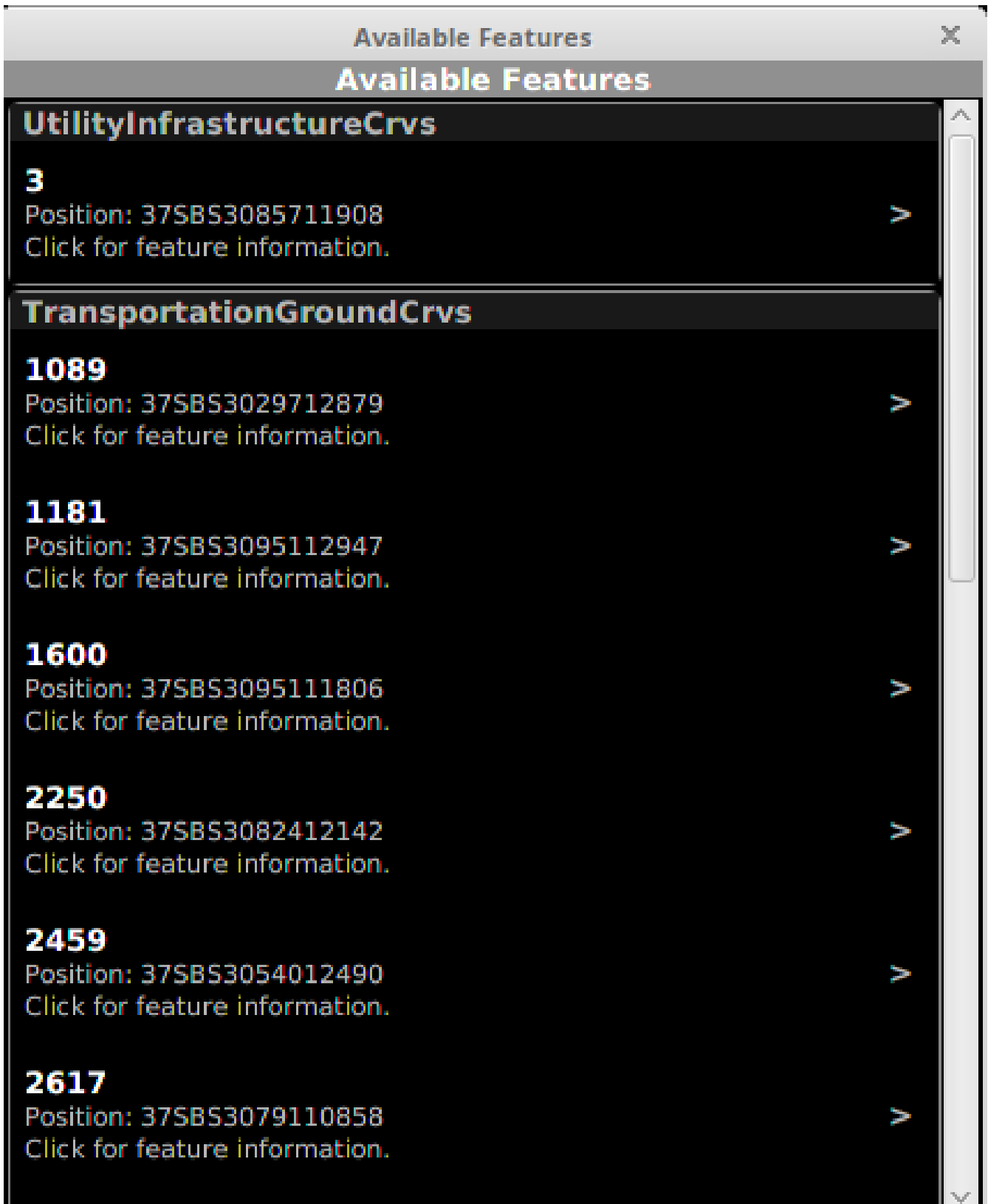


Figure 90. Compusult GOMobile Multiple Features (WMTS)

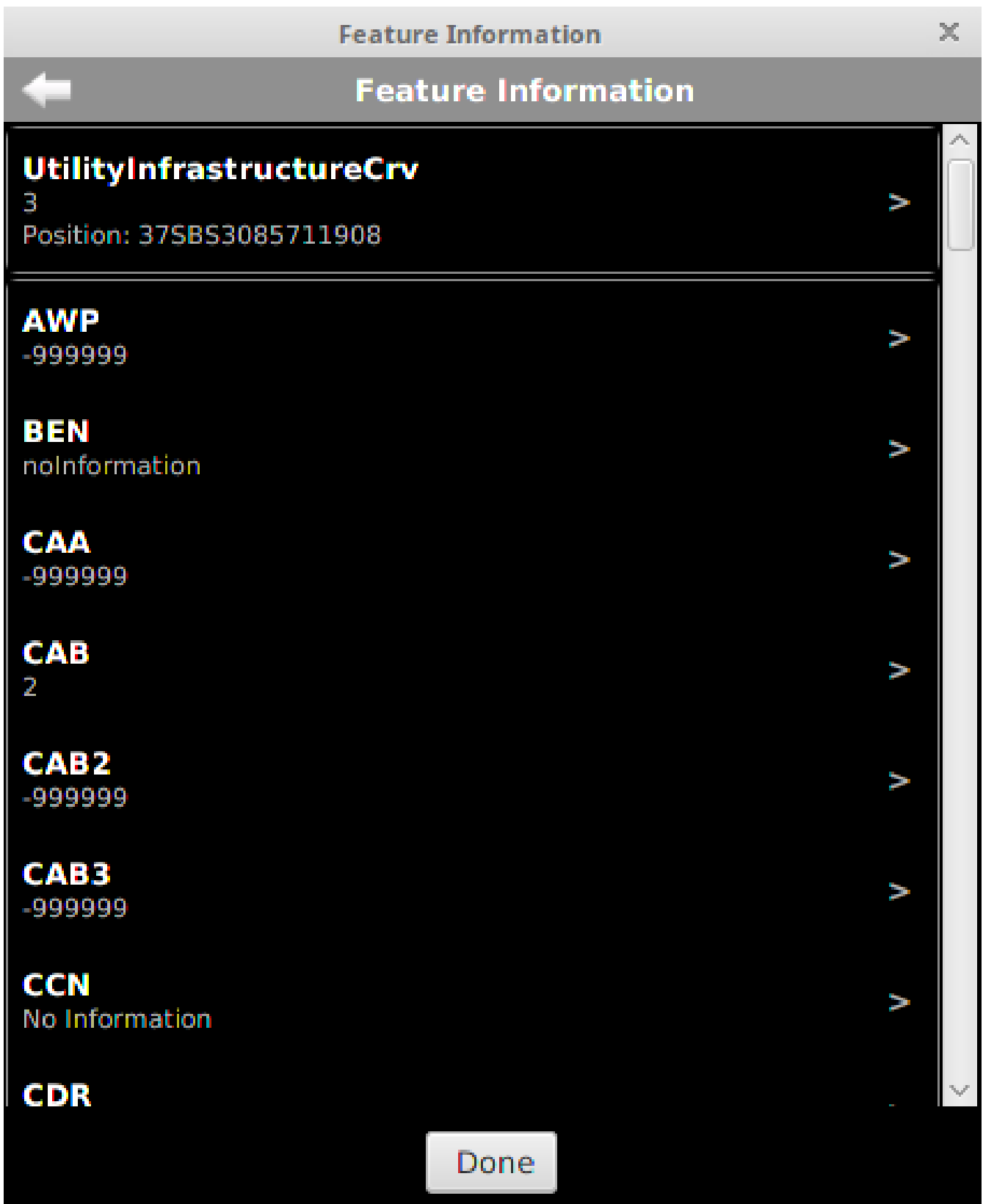
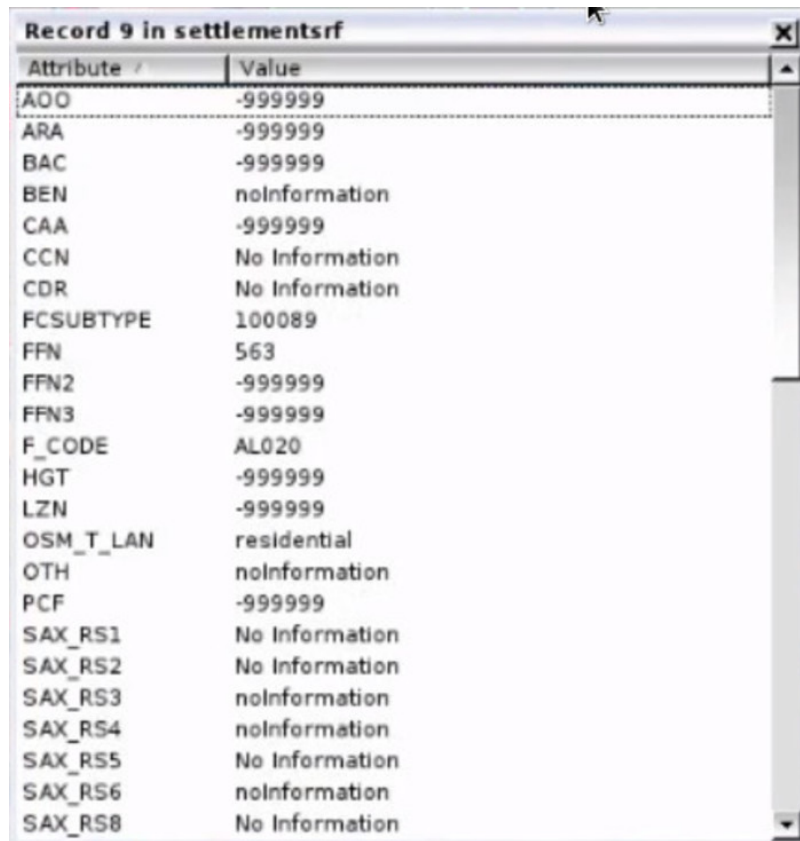


Figure 91. Compusult GOMobile Feature Info (GeoPackage)

Ecere

The Ecere client can query attributes from tiled feature data. For GeoPackages, this works with both embedded attributes as well as through an attributes table. Use of the attributes table allows complex queries, which would be further empowered by the use of spatial indexing based on the storage of feature extents outside the tiled feature data. The ability to query attributes from WMTS and some WFS3 instances is currently impeded by the lack of an implemented mechanism to

retrieve attributes schemas.



The screenshot shows a window titled "Record 9 in settlementsrf" with a table of attributes and their values. The table has two columns: "Attribute" and "Value".

Attribute	Value
AOD	-999999
ARA	-999999
BAC	-999999
BEN	noInformation
CAA	-999999
CCN	No Information
CDR	No Information
FCSUBTYPE	100089
FFN	563
FFN2	-999999
FFN3	-999999
F_CODE	AL020
HGT	-999999
LZN	-999999
OSM_T_LAN	residential
OTH	noInformation
PCF	-999999
SAX_RS1	No Information
SAX_RS2	No Information
SAX_RS3	noInformation
SAX_RS4	noInformation
SAX_RS5	No Information
SAX_RS6	noInformation
SAX_RS8	No Information

Figure 92. Ecere's GNOSIS Cartographer querying attributes

Chapter 8. Discussion

8.1. Differences between WMTS and Mapbox Layers

It is important to note that a WMTS layer is not the same as a Mapbox layer. A WMTS layer is defined by its styles, where each style renders one or more feature types. It is these individual feature types that are equivalent to Mapbox layers. Therefore, a WMTS tile (which can only be for a single WMTS layer) can contain more than one Mapbox layer, and the set of Mapbox layers that a WMTS tile contains is dictated by the requested style.

8.2. Differences Between SLD and Mapbox Styles

Mapbox styles approach scale dependencies by allowing specification of zoom levels, either as limits for display, or in tables to associate different symbolizer properties at different scales. The zoom levels refer to the common Web Mercator tileset, creating difficulties in styling maps in other coordinate reference systems.

A `StyledLayerDescriptor` element in an SLD document is a complex hierarchical container that may contain multiple styles and serve as a style library (see "library mode" in the SLD 1.0 specification). In particular, a `StyledLayerDescriptor` element can contain multiple `NamedLayer` elements, and each `NamedLayer` element can be associated with multiple `UserStyle` elements, to be used as alternatives (one of them can be marked as the default). Each `UserStyle` in turn can contain multiple `FeatureTypeStyle` elements, each one referring to a different "feature type". So, theoretically, a single `NamedLayer` could contain features with different structure and different type name, and the `UserStyle` element could style them all.

A Mapbox style instead defines a single map to be displayed, listing layers and their styling. In this respect, a `MBStyle` document resembles a SLD `UserStyle`, while no similar structure is provided for the `StyledLayerDescriptor` level.

8.3. Using Offerings to Correlate Tiled Feature Data Layers to Style Sheets

By populating the `gpkgext_context_offerings` table proposed in the [GeoPackage OWS Context Extension](#), a data manager creates an explicit association between tiled feature data layers and style sheets. If new style sheets are developed, they can be added to the `gpkgext_stylesheets` table and new offerings can be added to the `gpkgext_context_offerings` table. Unfortunately, no participants were able to explore this capability during the VTPExt.

8.3.1. Analysis

By making OWS Context tables atomic, implementers can establish links between tile sets and styles without the full complexity of OWS Context.

One participant proposed a `styles_set` column in `gpkgext_vt_layers`. This does not align well to the CONOPS. Operationally, the data provider may not be the cartographer tasked with producing the

map. GeoPackages will appear with vector tiles in them and styles may be applied later down the line. While it is the intent, there is no guarantee that style sheets will be universal. Even if URIs are used for `styles_set` instead of numeric IDs, this approach will still break down if two different organizations are responsible for producing their own stylesheets.

8.4. Using OWS Contexts to Describe Map Views

OWS Context is designed to describe a set of geospatial resources. In this scenario, there is one context for each common operational picture (e.g., topographic, satellite, and night view). Each context contains resources and offerings (map tiles, feature tiles, etc.) needed to represent that view. This extension is presented in [GeoPackage OWS Context Extension](#). Unfortunately, no participants were able to explore this capability during the VTPExt.

8.4.1. Analysis

Use of OWS Context does add complexity to the ensuing GeoPackage. In an attempt to mitigate this complexity, the number of tables was reduced to three and the stylesheets table was split into a separate extension.

It is possible that a GeoPackage client may not support the format of the stylesheet specified in an OWS Context but that it will support a format present in another equivalent stylesheet row (same `styles_set` and option, different format). This scenario will be a nuisance for the client but is not insurmountable.

An initial proposal was presented in the GeoPackage Extensions ER, but it was incomplete and a consensus was not reached during the initial Pilot period. In addition to changes in the stylesheets table as described previously, there were some minor changes from the proposed table structure from the GeoPackage Extensions ER:

- Name the top level table `gpkgext_contexts` instead of `gpkgext_context` for consistency.
- Move `layer_name` and `query` from `gpkgext_resources` to `gpkgext_offerings`.
- The proposal in the previous ER did not indicate how to specify a layer within a vector tiles set. Since a vector tiles table is not usable without specifying the actual layer, one way to do this would be to have a compound string like `table_name/layer_name` as the `layer_name` of `gpkgext_context_resources`. The `code` would indicate that it is a vector tiles table so a client would know to expect a compound layer there.

Appendix A: GeoPackage Tiled Feature Data Extensions (Informative)

Five GeoPackage extensions were developed as part of this Pilot project. This section presents each one using the [GeoPackage Extension Template](http://www.geopackage.org/spec120/#extension_template) [http://www.geopackage.org/spec120/#extension_template]. This section is informative, providing information relevant to a developer or administrator who wishes to understand how these extensions are supposed to work. Potential normative requirements are presented in [GeoPackage Extensions Requirements \(Normative\)](#).

A.1. Tiled Feature Data Extension

A.1.1. Extension Title

Tiled Feature Data

A.1.2. Introduction

The GeoPackage Tiled Feature Data extension defines the rules and requirements for encoding tiled feature data in a GeoPackage data store.

WARNING

This extension does not define an encoding for tiled feature data. To be usable, an additional extension such as [GeoPackage Mapbox Vector Tiles Extension](#) or [GeoPackage 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, but non-supporting, software packages.

A.1.3. Extension Author

Image Matters LLC, in collaboration with the participants of the OGC Vector Tiles Pilot.

A.1.4. Extension Name or Template

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

A.1.5. Extension Type

This extension provides new requirements dependent on GeoPackage [Clause 2.2 \(tiles\)](#) [http://www.geopackage.org/spec120/index.html#tiles].

A.1.6. Applicability

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

A.1.7. Scope

read-write

A.1.8. 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.

There are two additional required metadata tables, `gpkgext_tfd_layers` and `gpkgext_tfd_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]. This allows client software to understand the feature schemas without having to open individual tiles.

`gpkg_extensions`

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

Table 5. `gpkg_extensions` Table Rows

table_name	column_name	extension_name	definition	scope
<code>gpkgext_tfd_layers</code>	NULL	<code>im_tiled_feature_data</code>	<i>a reference to this file</i>	<i>read-write</i>
<code>gpkgext_tfd_fields</code>	NULL	<code>im_tiled_feature_data</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 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 "tiled-features".

`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.

`gpkgext_tfd_layers`

The `gpkgext_tfd_layers` table describes the layers in a tiled feature data tile 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
- `styleable_layer_set` is the optional name to identify the intended styles and stylesheets for the layer
- `attributes_table_name` is the optional name of an attributes table containing the attributes (when not embedded in the vector tiles)

`gpkgext_tfd_fields`

The `gpkgext_tfd_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_tvd_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. GeoPackage Mapbox Vector Tiles Extension

A.2.1. Extension Title

Mapbox Vector Tiles

A.2.2. 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/) [https://www.mapbox.com/vector-tiles/specification/] [version 2.1](https://github.com/mapbox/vector-tile-spec/tree/master/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) [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, but non-supporting, software packages.

A.2.3. Extension Author

Image Matters LLC, in collaboration with the participants of the OGC Vector Tiles Pilot.

A.2.4. Extension Name or Template

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

A.2.5. Extension Type

This extension defines an encoding for the [Tiled Feature Data Extension](#).

A.2.6. Applicability

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

A.2.7. Scope

read-write

A.2.8. Specification

If this extension is in use, then all of [Tiled Feature Data Extension](#) applies.

User Data Tables

Like other tile-based content, the physical data is stored in [user-defined tiles tables](#) [<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 6. `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 OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

A.3. GeoPackage GeoJSON Vector Tiles Extension

A.3.1. Extension Title

GeoJSON Vector Tiles

A.3.2. 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>].

This extension, like all GeoPackage extensions, is intended to be transparent and to not interfere with GeoPackage-compliant, but non-supporting, software packages.

A.3.3. Extension Author

Image Matters LLC, in collaboration with the participants of the OGC Vector Tiles Pilot.

A.3.4. Extension Name or Template

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

A.3.5. Extension Type

This extension defines an encoding for the [Tiled Feature Data Extension](#).

A.3.6. Applicability

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

A.3.7. Scope

read-write

A.3.8. Specification

If this extension is in use, then all of [\[TiledVectorDataExtensionClause\]](#) applies.

User Data Tables

Like other tile-based content, the physical data is stored in a [GeoJSON Feature Collection](#) [<https://tools.ietf.org/html/rfc7946#section-3.3>].

`gpkg_extensions`

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

Table 7. `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_geojson</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 OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

A.4. GeoPackage Styles Extension

WARNING

This subsection is under discussion and may change radically.

A.4.1. Extension Title

Styles

A.4.2. Introduction

This extension provides a mechanism for styles in a GeoPackage.

A.4.3. Extension Author

Image Matters LLC, in collaboration with the participants of the OGC Vector Tiles Pilot and the OWS Context SWG.

A.4.4. Extension Name or Template

`styles` (will become `gpkg_styles` if adopted by OGC)

A.4.5. Extension Type

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

A.4.6. Applicability

This extension allows for stylesheets to be stored in a GeoPackage. How those stylesheets are used is outside of the scope of this specification, but they could be incorporated into OWS Context (see [GeoPackage OWS Context Extension](#)).

A.4.7. Scope

read-write

A.4.8. Specification

`gpkg_extensions`

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

Table 8. `gpkg_extensions` Table Rows

table_name	column_name	extension_name	definition	scope
<code>gpkgext_stylesheets</code>	null	<code>styles</code>	<i>a reference to this file</i>	<code>read-write</code>
<code>gpkgext_symbols</code>	null	<code>styles</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 OGC, it will gain the "gpkg_" prefix and get a different definition permalink.

gpkgext_stylesheets

This table contains stylesheets, organized by style set and option. The columns of this table are:

- `id` is a primary key
- `layer_set` is text defining a layer set that is suitable for styling in a common way
- `style` is text describing a specific implementation for a layer set
- `format` is the format of the stylesheet (e.g., `mbstyle` or `sld`)
- `stylesheet` is the actual stylesheet BLOB
- `title` is a text title
- `description` is a text description

gpkgext_symbols

This table contains symbols, organized by style set and option. The columns of this table are:

- `id` is a primary key
- `symbol_id` is a string identifier such as a URI that can uniquely identify the symbol
- `content_type` is the media type (formerly MIME type, e.g., `image/svg+xml` or `image/png`) of the symbol
- `symbol` is the actual symbol BLOB
- `title` is a text title
- `description` is a text description

NOTE

As with other GeoPackage tables, this specification takes no position on how either of these tables are to be used by a client.

A.5. GeoPackage OWS Context Extension

WARNING

This subsection is under discussion and may change radically.

A.5.1. Extension Title

OWS Context

A.5.2. Introduction

This extension provides a mechanism for storing [OWS Context](http://owscontext.org) [http://owscontext.org] content in a GeoPackage. It is aligned with the [OWS Context Conceptual Model](https://portal.opengeospatial.org/files/?artifact_id=55182) [https://portal.opengeospatial.org/files/?artifact_id=55182].

A.5.3. Extension Author

Image Matters LLC, in collaboration with the participants of the OGC Vector Tiles Pilot and the OWS

Context SWG.

A.5.4. Extension Name or Template

`owscontext` (will become `gpkg_owscontext` if adopted by OGC)

A.5.5. Extension Type

New requirement dependent on [GeoPackage Core \(Clause 1\)](http://www.geopackage.org/spec/#core) [http://www.geopackage.org/spec/#core]. It is optionally dependent on the [Styles Extension](#).

A.5.6. Applicability

This extension adds an additional level of organization to existing GeoPackage data.

A.5.7. Scope

read-write

A.5.8. Specification

The following UML diagram illustrates the relationship between the four OWS Context tables.

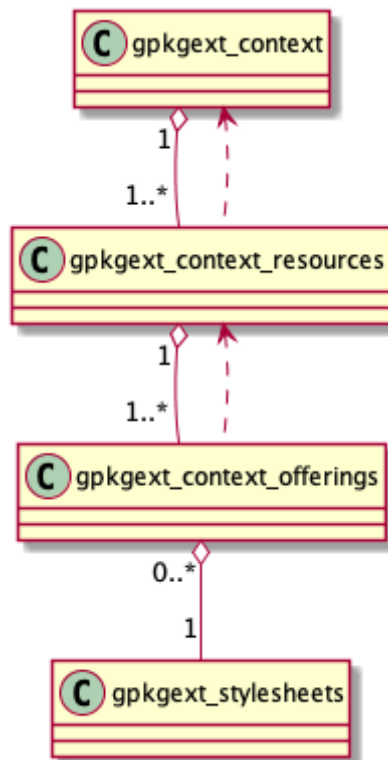


Figure 93. OWS Context Model

`gpkg_extensions`

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

Table 9. `gpkg_extensions` Table Rows

table_name	column_name	extension_name	definition	scope
gpkgext_contexts	null	ows_context	a reference to this file	read-write
gpkgext_context_resources	null	ows_context	a reference to this file	read-write
gpkgext_context_offerings	null	ows_context	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.

gpkgext_contexts

This table describes OWS Context instances. The columns of this table are:

- **id** is a primary key
- **title**, **abstract**, **author**, **publisher**, **creator**, **rights**, and **keywords** are text descriptions
- **last_change** is a timestamp in ISO 8601 format
- **min_x**, **min_y**, **max_x**, **max_y**, **srs_id**, **min_time**, and **max_time** are the spatio-temporal extents of the context
- **metadata_id** is a foreign key to **gpkg_metadata** for use with the **metadata extension** [http://www.geopackage.org/spec121/#extension_metadata].

gpkgext_context_resources

This table represents an `owc:SQLResource`, which could be a file, service, or inline content. The columns of this table are:

- **id** is a primary key
- **context_id** is a foreign key to **id** from **gpkgext_contexts**
- **title**, **abstract**, **author**, **publisher**, **rights**, **description**, and **keywords** are text descriptions
- **min_x**, **min_y**, **max_x**, **max_y**, **srs_id**, **min_time**, and **max_time** are the spatio-temporal extents of the context
- **active** is a boolean flag indicating the state of the resource within the context document
- **min_scale_denominator** and **max_scale_denominator** are the minimum and maximum display scales
- **order** is the ascending order of the resource
- **requestURL** is the service request URL or file URL (. for the current file)
- **code** identifies the type of resource (e.g., **GPKG** or **WMS**)

gpkgext_context_offerings

This table contains `owc:Offering` instances which could be a service layer or table. The columns of

this table are:

- `id` is a primary key
- `resource_id` is a foreign key to `id` from `gpkg_context_resources`
- `stylesheet_id` is a foreign key to `id` from `gpkgext_stylesheets` (see `gpkgext_stylesheets`)
- `code` is a code identifying the type of offering
- `method` is the name of the operation method request (e.g., `GET`)
- `type` is the media type of the return result
- `layer_name` is a single layer, table, or view name (for tiled feature data, put both the table name and layer name, separated by `/`, for example `"tiles_Daraa/AgricultureSrf"`)
- `query` is an actual SQL or HTTP query
- `contents` is the actual data (for inline data)

TIP | TODO: Provide some example codes.

Appendix B: GeoPackage Extensions Requirements (Normative)

This section describes the normative requirements for the GeoPackage extensions described in this document. It is in the form of a draft specification that may ultimately be adopted by OGC.

B.1. GeoPackage Tiled Feature Data Extension

Requirements Class : Tiled Feature Data	
http://www.opengis.net/spec/gpkg-tfd/1.0/tiled-feature-data	
Target type	Token
Dependency	www.opengis.net/spec/gpkg/1.2.0/opt/tiles

WARNING

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

B.1.1. `gpkg_contents`

Table Values

Requirement TFDE1 – Tiled Feature Data Presence	<i>/req/tiled_feature_data/presence</i> A GeoPackage with at least one row in <code>gpkg_contents</code> with a <code>data_type</code> of "tiled-feature-data" SHALL comply with this extension.
Requirement TFDE2 – <code>gpkg_contents</code> Rows	<i>/req/tiled_feature_data/gcr</i> Each row in <code>gpkg_contents</code> with a <code>data_type</code> of "tiled-feature-data" SHALL be a tile set as described by this extension.

WARNING

A tile used in this extension MAY be [deflate compressed](https://tools.ietf.org/html/rfc1951) [https://tools.ietf.org/html/rfc1951]. For maximum interoperability, a client SHOULD inspect a tile, determine whether it is compressed, and respond accordingly.

B.1.2. `gpkg_extensions`

Table Values

Requirement TFDE3 – gpkg_extensions Rows	<i>/req/tiled_feature_data/ger</i> A GeoPackage that complies with this extension SHALL contain rows in the gpkg_extensions table for gpkgext_tfd_layers and gpkgext_tfd_fields as described in gpkg_extensions Table Rows .
---	--

Requirement TFDE4 – gpkg_extensions Encoding	<i>/req/tiled_feature_data/ger-encoding</i> For each tile set of tiled feature data, there SHALL be one and only one corresponding row in gpkg_extensions specifying an extension describing the encoding for the tiles in the tile_data column of the tile pyramid user data table.
---	---

B.1.3. gpkgext_tfd_layers

Table Definition

Requirement TFDE5 – gpkgext_tfd_layers Table	<i>/req/tiled_feature_data/gpkgext_tfd_layers/table</i> A GeoPackage that complies with this extension SHALL contain a gpkgext_tfd_layers table as per Tiled Feature Data Layers Table Definition and [gpkgext_tfd_layers_sql] .
---	--

Table 10. Tiled Feature Data Layers Table Definition

Column Name	Column Type	Column Description	Null Allowed	Key	Unique
id	INTEGER	Autoincrement primary key	no	PK	yes
table_name	TEXT	Name of the table user-defined tiles table containing the tiled feature data	no	FK	yes (see [tfde4])
name	TEXT	Layer Name	no		yes (see [tfde4])
description	TEXT	Text description	yes		
minzoom	INTEGER	Minimum zoom level	yes		
maxzoom	INTEGER	Maximum zoom level	yes		

Column Name	Column Type	Column Description	Null Allowed	Key	Unique
<code>style_layer_set</code>	TEXT	name to identify the intended styles and stylesheets for the layer	yes		
<code>attributes_table_name</code>	TEXT	name of an attributes table	yes		

Table Values

Requirement TFDE6 – gpkgext_tfd_layers table reference	<p><i>/req/tiled_feature_data/gpkgext_tfd_layers/table_ref</i></p> <p>For each row in <code>gpkgext_tfd_layers</code>, there SHALL be a table or view of the name referenced in <code>table_name</code> and that table or view SHALL have an entry in <code>gpkg_contents</code>.</p>
---	---

Requirement TFDE7 – gpkgext_relations Base Table	<p><i>/req/tiled_feature_data/gpkgext_tfd_layers/uniqueness</i></p> <p>The rows in <code>gpkgext_tfd_layers</code> SHALL have a jointly unique <code>table_name</code> and <code>name</code>.</p>
---	---

Requirement TFDE10 – Attributes Table	<p><i>/req/tiled_feature_data/gpkgext_tfd_layers/attributes</i></p> <p>Each non-null <code>attributes_table_name</code> value in <code>gpkgext_tfd_layers</code> SHALL have a corresponding entry in <code>gpkg_contents</code> with a <code>data_type</code> of "attributes".</p>
--	--

NOTE If a row of `gpkgext_tfd_layers` has a non-null `attributes_table_name`, it is expected that the corresponding tiled feature data will not have attributes in its tiles and that a client will retrieve select attributes from the attributes table.

B.1.4. gpkgext_tfd_fields

Table Definition

Requirement TFDE8 – gpkgext_tfd_fields Table	<p><i>/req/tiled_feature_data/gpkgext_tfd_fields/table</i></p> <p>A GeoPackage that complies with this extension SHALL contain a <code>gpkgext_tfd_fields</code> table as per Tiled Feature Data Fields Table Definition and [gpkgext_tfd_fields_sql].</p>
---	--

Table 11. Tiled Feature Data Fields Table Definition

Column Name	Column Type	Column Description	Null Allowed	Key
id	INTEGER	Autoincrement primary key	no	PK
layer_id	INTEGER	An id from <code>gpkgext_tfd_layers</code>	no	FK
name	TEXT	Field name	no	
type	TEXT	"String", "Number", or "Boolean"	no	

Table Values

Requirement TFDE9 – gpkgext_tfd_fields table reference	<p><i>/req/tiled_feature_data/gpkgext_tfd_fields/table_ref</i></p> <p>For each row in <code>gpkgext_tfd_fields</code>, the <code>layer_id</code> SHALL have a corresponding entry in <code>gpkgext_tfd_layers</code>.</p>
---	---

B.2. GeoPackage Mapbox Vector Tiles Extension

Requirements Class : Mapbox Vector Tiles	
http://www.opengis.net/spec/gpkg-vt/1.0/mapbox-vector-tiles	
Target type	Token
Dependency	www.opengis.net/spec/gpkg/1.2.0/opt/tiles
Dependency	http://www.opengis.net/spec/gpkg-tfd/1.0/tiled_feature_data

NOTE

At the time of writing, Mapbox Vector Tiles is not a standard. Because of this, this extension is not a candidate for becoming an OGC standard. However, it is a candidate for being a [community extension](http://www.geopackage.org/extensions.html) [http://www.geopackage.org/extensions.html] and an [OGC Best Practice](http://www.opengeospatial.org/docs/bp) [http://www.opengeospatial.org/docs/bp].

B.2.1. gpkg_extensions

Table Values

Requirement MVTE1 – gpkg_extensions Rows	<p><i>/req/mvte/ger</i></p> <p>A GeoPackage that contains a row in the <code>gpkg_extensions</code> table for <code>im_vector_tiles_mapbox</code> or its alias as described in gpkg_extensions Table Rows SHALL comply with the Mapbox Vector Tiles Extension as described by this document.</p>
---	--

B.2.2. User Defined Tiles Tables

Table Values

Requirement MVTE2 – Tile Format	<i>/req/mvte/tile_format</i> For each tile set governed by this extension, tile_data column values of the corresponding tile pyramid user data table SHALL contain Mapbox Vector Tiles as per 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].
--	--

B.3. GeoPackage GeoJSON Vector Tiles Extension

Requirements Class : GeoJSON Vector Tiles	
http://www.opengis.net/spec/gpkg-tfd/1.0/geojson-vector-tiles	
Target type	Token
Dependency	www.opengis.net/spec/gpkg/1.2.0/opt/tiles
Dependency	http://www.opengis.net/spec/gpkg-tfd/1.0/tiled_feature_data

B.3.1. **gpkg_extensions**

Table Values

Requirement GVTE1 – gpkg_extensions Rows	<i>/req/gvte/ger</i> A GeoPackage that contains a row in the gpkg_extensions table for im_vector_tiles_geojson or its alias as described in gpkg_extensions Table Rows SHALL comply with the GeoJSON Extension as described by this document.
---	--

B.3.2. User Defined Tiles Tables

Table Values

Requirement GVTE2 – Tile Format	<i>/req/gvte/tile_format</i> For each tile set governed by this extension, tile_data column values of the corresponding tile pyramid user data table SHALL contain GeoJSON Feature Collections as per The GeoJSON Format [https://tools.ietf.org/html/rfc7946].
--	---

B.4. GeoPackage Styles Extension

B.4.1. `gpkg_extensions`

Table Values

Requirement SS1 – <code>gpkg_extensions</code> Rows	<i>/req/gsse/ger</i> A GeoPackage that complies with this extension SHALL contain rows in the <code>gpkg_extensions</code> table for <code>gpkgext_stylesheets</code> as described in [styles_ger_table] .
--	---

B.4.2. `gpkgext_stylesheets`

The `gpkgext_stylesheets` table implements `owc:StyleSet`.

Table Definition

Requirement SS2 – <code>gpkgext_stylesheets</code> Table	<i>/req/gsse/gpkgext_stylesheets/table</i> A GeoPackage that complies with this extension SHALL contain a <code>gpkgext_stylesheets</code> table as per Stylesheet Table Definition and [gpkgext_stylesheets_sql] .
---	--

Table 12. Stylesheet Table Definition

Column Name	Column Type	Column Description	Null Allowed	Key
<code>id</code>	INTEGER	Autoincrement primary key	no	PK
<code>styles_set</code>	TEXT	a string defining the styles set	yes	
<code>style</code>	TEXT	a enumeration corresponding to the <code>styles_set</code> option	yes	
<code>format</code>	TEXT	style format (e.g., "mbstyle" or "sld")	no	
<code>stylesheet</code>	BLOB	the actual stylesheet	no	
<code>title</code>	TEXT	a title	yes	
<code>description</code>	TEXT	a description	yes	

Table Values

Requirement SSE3 – gpkgext_stylesheets Base Table	<p><i>/req/gsse/gpkgext_stylesheets/uniqueness</i></p> <p>The rows in <code>gpkgext_stylesheets</code> SHALL have a jointly unique <code>styles_set</code>, <code>style</code>, and <code>format</code> for rows with a non-null <code>styles_set</code> and <code>style</code>.</p>
--	--

B.4.3. gpkgext_stylesheets

The `gpkgext_stylesheets` table implements `owc:StyleSet`.

Table Definition

Requirement SS4 – gpkgext_stylesheets Table	<p><i>/req/gsse/gpkgext_stylesheets/table</i></p> <p>A GeoPackage that complies with this extension SHALL contain a <code>gpkgext_stylesheets</code> table as per Stylesheet Table Definition and [gpkgext_stylesheets_sql].</p>
--	--

Table 13. Symbols Table Definition

Column Name	Column Type	Column Description	Null Allowed	Key
<code>id</code>	INTEGER	Autoincrement primary key	no	PK
<code>symbol_id</code>	TEXT	a string defining the styles set	no	
<code>content_type</code>	TEXT	media type (e.g., "image/png" or "image/svg")	no	
<code>symbol</code>	BLOB	the actual symbol	no	
<code>title</code>	TEXT	a title	yes	
<code>description</code>	TEXT	a description	yes	

Table Values

Requirement SSE5 – gpkgext_stylesheets Base Table	<p><i>/req/gsse/gpkgext_symbols/uniqueness</i></p> <p>The rows in <code>gpkgext_symbols</code> SHALL have a jointly unique <code>symbol_id</code> and <code>content_type</code>.</p>
--	--

B.5. GeoPackage OWS Context Extension

B.5.1. `gpkg_extensions`

Table Values

Requirement OWCE1 – gpkg_extensions Rows	<i>/req/owce/ger</i> A GeoPackage that complies with this extension SHALL contain rows in the <code>gpkg_extensions</code> table as described in [owscontext_ger_table] .
Requirement OWCE2 – Stylesheet Extension Dependency	<i>/req/owce/ger-stylesheet</i> A GeoPackage with at least one non-null <code>stylesheet_id</code> in <code>gpkgext_context_offerings</code> SHALL comply with GeoPackage Styles Extension .
Requirement OWCE3 – Metadata Extension Dependency Rows	<i>/req/owce/ger-metadata</i> A GeoPackage with at least one non-null <code>metadata_id</code> in <code>gpkgext_contexts</code> SHALL comply with the GeoPackage Metadata Extension [http://www.geopackage.org/spec/#extension_metadata].

B.5.2. `gpkgext_contexts`

Table Definition

Requirement OWCE4 – gpkgext_contexts Table	<i>/req/owce/gpkgext_contexts/table</i> A GeoPackage that complies with this extension SHALL contain a <code>gpkgext_contexts</code> table as per OWS Context Table Definition and [gpkgext_contexts_sql] .
---	--

Table 14. OWS Context Table Definition

Column Name	Column Type	Column Description	Null Allowed	Default	Key
<code>id</code>	INTEGER	Autoincrement primary key	no		PK

Column Name	Column Type	Column Description	Null Allowed	Default	Key
title	TEXT	A human-readable title for the OWS Context document	no		
abstract	TEXT	A human-readable description of the OWS Context document purpose and/or content	yes	"	
last_change	DATETIME	timestamp of last change to content, in ISO 8601 format	no	<code>strftime('%Y-%m-%dT%H:%M:%fZ', 'now')</code>	
min_x	DOUBLE	Bounding box minimum easting or longitude for the users of the context document	yes		
min_y	DOUBLE	Bounding box minimum northing or latitude for the users of the context document	yes		
max_x	DOUBLE	Bounding box maximum easting or longitude for the users of the context document	yes		
max_y	DOUBLE	Bounding box maximum northing or latitude for the users of the context document	yes		

Column Name	Column Type	Column Description	Null Allowed	Default	Key
<code>srs_id</code>	INTEGER	Spatial Reference System ID: <code>gpkg_spatial_ref_sys.srs_id</code> for the geographic extents	yes		FK
<code>author</code>	TEXT	Identifier for the author of the document	yes		
<code>publisher</code>	TEXT	Identifier for the publisher of the document	yes		
<code>creator</code>	TEXT	The tool/application used to create the context document and its properties	yes		
<code>rights</code>	TEXT	Rights which apply to the context document	yes		
<code>keywords</code>	TEXT	Comma-delimited list of keywords related to this context document	yes		
<code>metadata_id</code>	INTEGER	<code>id</code> from <code>gpkg_metadata</code>	yes		
<code>min_time</code>	DATETIME	Beginning of time interval, in ISO 8601 format	yes		
<code>max_time</code>	DATETIME	End of time interval, in ISO 8601 format	yes		

Table Values

Requirement VTE5 – gpkgext_contexts srs reference	<i>/req/owce/gpkgext_contexts/srs_table_ref</i> For each row in <code>gpkgext_contexts</code> with a non-null <code>srs_id</code> , there SHALL be a corresponding row in <code>gpkg_spatial_ref_sys</code> with an <code>id</code> matching the <code>srs_id</code> .
--	---

Requirement VTE6 – gpkgext_contexts metadata reference	<i>/req/owce/gpkgext_contexts/srs_table_ref</i> For each row in <code>gpkgext_contexts</code> with a non-null <code>metadata_id</code> , there SHALL be a corresponding row in <code>gpkg_metadata</code> with an <code>id</code> matching the <code>metadata_id</code> .
---	--

NOTE

The rights described apply to the Context Document itself and not to any of its contents. The intent of the temporal and spatial extents is to indicate to a GeoPackage client the expected view of the information in area in time, not to describe the referenced resources themselves.

B.5.3. `gpkgext_context_resources`

Table Definition

Requirement OWCE7 – gpkgext_context_re sources Table	<i>/req/owce/gpkgext_context_resources/table</i> A GeoPackage that complies with this extension SHALL contain a <code>gpkgext_context_resources</code> table as per OWS Context Resources Table Definition and [gpkgext_context_resources_sql] .
---	---

Table 15. OWS Context Resources Table Definition

Column Name	Column Type	Column Description	Null Allowed	Default	Key
<code>id</code>	INTEGER	Autoincrement primary key	no		PK
<code>context_id</code>	INTEGER	<code>id</code> from <code>gpkgext_contexts</code>	no		FK
<code>title</code>	TEXT	A human-readable title for the OWS Context resource	no		

Column Name	Column Type	Column Description	Null Allowed	Default	Key
abstract	TEXT	A human-readable description of the OWS Context document purpose and/or content	yes	"	
author	TEXT	Identifier for the author of the document	yes		
publisher	TEXT	Identifier for the publisher of the document	yes		
rights	TEXT	Rights which apply to the context document	yes		
min_x	DOUBLE	Bounding box minimum easting or longitude for the users of the context document	yes		
min_y	DOUBLE	Bounding box minimum northing or latitude for the users of the context document	yes		
max_x	DOUBLE	Bounding box maximum easting or longitude for the users of the context document	yes		
max_y	DOUBLE	Bounding box maximum northing or latitude for the users of the context document	yes		

Column Name	Column Type	Column Description	Null Allowed	Default	Key
<code>srs_id</code>	INTEGER	Spatial Reference System ID: <code>gpkg_spatial_ref_sys.srs_id</code> for the geographic extents	yes		FK
<code>min_time</code>	DATETIME	Beginning of time interval, in ISO 8601 format	yes		
<code>max_time</code>	DATETIME	End of time interval, in ISO 8601 format	yes		
<code>description</code>	TEXT	A reference to a description of the Context resource in alternative format	yes		
<code>active</code>	BOOLEAN	This flag indicates the state of the resource within the context document. It can be interpreted by the caller as required (this may be defined in a profile or in the specific service extensions)	yes	TRUE	
<code>keywords</code>	TEXT	Comma-delimited list of keywords related to this context document	yes		
<code>min_scale_denominator</code>	DOUBLE	Minimum scale for the display of the layer	yes		

Column Name	Column Type	Column Description	Null Allowed	Default	Key
max_scale_denominator	DOUBLE	Maximum scale for the display of the layer	yes		
order	DOUBLE	The ascending order of the resource	yes		
requestURL	TEXT	Service Request URL or file URL	yes		
code	TEXT	Code identifying the type of resource	no		

Table Values

Requirement OWCE8 – gpkgext_context_resources context table reference	<p><i>/req/owce/gpkgext_context_resources/contexts_table_ref</i></p> <p>For each row in <code>gpkgext_context_resources</code> with a non-null <code>context_id</code>, there SHALL be a corresponding row in <code>gpkgext_contexts</code> with an <code>id</code> matching the <code>context_id</code>.</p>
Requirement OWCE9 – gpkgext_context_resources srs reference	<p><i>/req/owce/gpkgext_context_resources/srs_table_ref</i></p> <p>For each row in <code>gpkgext_context_resources</code> with a non-null <code>srs_id</code>, there SHALL be a corresponding row in <code>gpkg_spatial_ref_sys</code> with an <code>id</code> matching the <code>srs_id</code>.</p>

B.5.4. gpkgext_context_offerings

Table Definition

Requirement OWCE10 – gpkgext_context_offerings Table	<p><i>/req/owce/gpkgext_context_offerings/table</i></p> <p>A GeoPackage that complies with this extension SHALL contain a <code>gpkgext_context_offerings</code> table as per OWS Context Offerings Table Definition and [gpkgext_context_offerings_sql].</p>
---	---

Table 16. OWS Context Offerings Table Definition

Column Name	Column Type	Column Description	Null Allowed	Default	Key
id	INTEGER	Autoincrement primary key	no		PK
resource_id	INTEGER	id from gpkgext_context_resources	yes		FK
code	TEXT	Code identifying the type of offering	no		
stylesheet_id	INTEGER	id from gpkgext_stylesheets	yes		FK
method	TEXT	Name of operation method request	no		
type	TEXT	Media type (formerly known as MIME-Type) of the return result	no		
layer_name	TEXT	A single layer, table, or view name	yes		
query	TEXT	The actual SQL or HTTP query	yes		
contents	BLOB	Actual data (for inline data)	yes		

Table Values

Requirement OWCE11 – gpkgext_context_offerings resources table reference	<p><i>/req/owce/gpkgext_context_offerings/resources_table_ref</i></p> <p>For each row in <code>gpkgext_context_offerings</code> with a non-null <code>resource_id</code>, there SHALL be a corresponding row in <code>gpkgext_context_resources</code> with an <code>id</code> matching the <code>resource_id</code>.</p>
---	---

Requirement OWCE12 – gpkgext_context_offerings stylesheets table reference	<p><i>/req/owce/gpkgext_context_offerings/stylesheets_table_ref</i></p> <p>For each row in <code>gpkgext_context_offerings</code> with a non-null <code>stylesheet_id</code>, there SHALL be a corresponding row in <code>gpkgext_stylesheets</code> with an <code>id</code> matching the <code>stylesheet_id</code>.</p>
---	---

Appendix C: The OpenAPI Styles API

C.1. Overview

This chapter describes API building blocks for managing and fetching styles via a Web API.

The design is based on the architecture of the draft WFS 3.0 Core standard and uses OpenAPI to specify the building blocks.

The resources of the API are specified in the [Conceptual Model](#).

The general intent is that style sheets may be applied to every feature representation, including features partitioned in tiles, i.e., if the style sheet language supports it, a style may be applied to the features or the feature tiles. In the VTPExt, the style sheets have only been used to style feature tiles.

A style may be represented in multiple representations. The same style, e.g., "topographic", may be provided in multiple styling languages, for example, Mapbox Style and SLD/SE. In the terminology of the conceptual model, each representation is a style sheet of that style. The usual mechanisms apply to select the desired style sheet from the server using the media type of the style sheets.

Currently, the Styles API does not support multiple stylable layer sets. There is currently one (implicit) layer set which consists of all collections in the dataset. Of course, one can still define styles that include styling rules for just a subset of all layers in the dataset.

The approach taken is consistent with the architecture for feature resources in WFS 3.0 (i.e. `/collection/{collectionId}/items` or `/items` and sub-resources) and for tile resources (i.e. `/collection/{collectionId}/tiles` or `/tiles` and sub-resources).

The fundamental facts implemented are:

- A dataset has 1..n collections/layers.
- A dataset has 0..n styles.
- Each style is applicable to 1..n collections/layers of the dataset.
- A style is represented by 1..n style sheets.

[Table 2](#) provides an overview of the resources for which this extension defines requirements:

Table 17. Overview of resources, applicable HTTP methods and links to the document sections

Resource	Path	HTTP method	Comment
Landing page	<code>/</code>	GET	Updated to add link to the style set
Conformance classes	<code>/conformance</code>	GET	Updated to add conformance declaration for the Styles API

Resource	Path	HTTP method	Comment
Feature collection metadata	<code>/collections/{collectionId}</code>	GET	Updated to add links to the styles applicable for the features in the collection
Style set	<code>/styles</code>	GET, POST	Fetch the style set or create new styles
Style	<code>/styles/{styleId}</code>	GET, PUT, DELETE	Fetch a style sheet for a style or create/update/delete a style

There are open issues that require future work. They are discussed in the last section of this chapter.

C.2. API definition

C.2.1. Encoding

In the VTPEExt, only a JSON representation of the resources was considered, with the exception of style sheets which use SLD, an XML encoding.

The description of the [implementation by interactive instruments](#) includes a discussion on how to use the style resources in the HTML representation. However, no formal requirements for an HTML representation were discussed during the VTPEExt.

C.2.2. Landing page

Path: /

Supported methods: GET

Requirement 1	<p><code>/req/styles/landing-page</code></p> <p>The landing page of the dataset SHALL include a link to the style set resource with <code>rel=styles</code>.</p>
----------------------	--

Example 1. Link to style set in the landing page response document

```
...
{
  "rel": "styles",
  "type": "application/json",
  "title": "the set of available styles",
  "href": "https://services.interactive-instruments.de/vtp/daraa/styles?f=json"
}
...
```

C.2.3. Style set

Path: `/styles`

Methods: GET, POST

Describe the styles that are available.

Requirement 2	<code>/req/styles/styles-get</code> The server SHALL support the HTTP GET method at the path <code>/styles</code> .
Requirement 3	<code>/req/styles/styles-response</code> A successful execution of the GET method SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema <code>styleset.yaml</code> .

`styleset.yaml`: Schema for style set

```
type: object
required:
  - styles
properties:
  styles:
    type: array
    items:
      $ref: style.yaml
default:
  type: string
```

```
type: object
required:
  - id
  - links
properties:
  id:
    type: string
  description:
    type: string
  links:
    type: array
    items:
      $ref: link.yaml
```

NOTE

The schema uses **id** as the member name. In other places of the API, **identifier** has been used. Moving forward, one of the two should be used consistently throughout the API.

Requirement 4	/req/styles/styles-ids Each style in the style set SHALL have a unique id .
Requirement 5	/req/styles/styles-links Each style in the style set SHALL have a link to each style sheet available for that style with rel=style .
Requirement 6	/req/styles/styles-default If a value for default is provided, it SHALL be the id of a style in the style set.

```
{
  "styles": [
    { "id": "topographic",
      "links": [
        {
          "rel": "style",
          "type": "application/json",
          "href": "https://services.interactive-
instruments.de/vtp/daraa/collections/argiculturesrf/styles/topographic?f=json"
        }
      ]
    },
    { "id": "satellite-overlay",
      "links": [
        {
          "rel": "style",
          "type": "application/json",
          "href": "https://services.interactive-
instruments.de/vtp/daraa/collections/argiculturesrf/styles/satellite-
overlay?f=json"
        }
      ]
    },
    { "id": "night",
      "links": [
        {
          "rel": "style",
          "type": "application/json",
          "href": "https://services.interactive-
instruments.de/vtp/daraa/collections/argiculturesrf/styles/night?f=json"
        }
      ]
    }
  ],
  "default": "topographic"
}
```

There are currently no registered media types for SLD or Mapbox style documents. Therefore, the example uses the generic XML and JSON media types. This is sufficient for VTPExt, but it would no longer work once another style language that uses XML or JSON needs to be supported, too.

NOTE

Some SLD implementations use media types like `application/vnd.ogc.sld+xml`. However, the media type has not been registered in the IANA register nor is it specified in the standards "Styled Layer Descriptor profile of the Web Map Service" or "Symbology Encoding".

Requirement 7	/req/styles/styles-post The server SHALL support the HTTP POST method at the path /styles .
Requirement 8	/req/styles/styles-new-style A successful execution of the POST method where the content of the request is a style sheet of a supported media type SHALL result in the creation of a new style resource and be reported as a response with a HTTP status code 201 with a link to the new style resource.
Recommendation 1	/rec/styles/styles-new-style-name The server SHOULD inspect the style sheet in order to assign a meaningful identifier.
Requirement 9	/req/styles/styles-new-default A successful execution of the POST method where the content of the request is a object with a member default SHALL change the default style to the value of the member, if it is a valid style id. The result SHALL be reported as a response with a HTTP status code 204.

C.2.4. Style

Path: **/styles/{styleId}**

Methods: GET, PUT, DELETE

Get a style sheet of a style. Update or delete a style.

GET will return the style, if a media type is available, that is acceptable to the client as expressed by the Accept header or the URI - using a f parameter or a file extension).

DELETE will delete the style.

PUT will update the style. Or if the style does not yet exist, the style will be created.

Requirement 10	/req/styles/style-get The server SHALL support the HTTP GET method at the path /styles/{styleId} , if styleId is a style in the style set.
-----------------------	---

<p>Requirement 11</p>	<p>/req/styles/style-get-response</p> <p>A successful execution of the GET method SHALL be reported as a response with a HTTP status code 200.</p> <p>The content of that response SHALL be a style sheet in a requested media type.</p>
<p>Requirement 12</p>	<p>/req/styles/style-delete</p> <p>The server SHALL support the HTTP DELETE method at the path /styles/{styleId}, if styleId is a style in the style set.</p>
<p>Requirement 13</p>	<p>/req/styles/style-delete-response</p> <p>A successful execution of the DELETE method SHALL be reported as a response with a HTTP status code 204.</p>
<p>Requirement 14</p>	<p>/req/styles/style-put</p> <p>The server SHALL support the HTTP PUT method at the path /styles/{styleId}.</p>
<p>Requirement 15</p>	<p>/req/styles/style-put-response-existing</p> <p>If styleId is an existing style in the style set and the content of the request is a style sheet of a supported media type, the previous style definition SHALL be replaced and the successful execution of the PUT method SHALL be reported as a response with a HTTP status code 204.</p>
<p>Requirement 16</p>	<p>/req/styles/style-put-response-new</p> <p>If styleId is not an existing style in the style set and the content of the request is a style sheet of a supported media type, the result SHALL be the creation of a new style resource and be reported as a response with a HTTP status code 201 with a link to the new style resource.</p>

C.2.5. Collection metadata

Path: /collections/{collectionId}

Supported methods: GET

Add links to the styles that are applicable for the features of each collection.

Recommendation 2	<p>/rec/styles/collection</p> <p>The collection object SHOULD include a link with <code>rel=style</code> to each style sheet that includes styling rules for features in the collection.</p>
-------------------------	--

For style sheets in languages that explicitly identify the data that is styled, like Mapbox Style, this information is redundant and can be derived from the style definition. For other style sheets it can be essential to enable clients to determine how to apply style sheets to the feature data.

Example 3. Links to 'topographic' and 'night' styles in Mapbox and SLD format

```
...
{
  "rel": "style",
  "type": "application/json",
  "title": "topographic (Mapbox)",
  "href": "https://services.interactive-
instruments.de/vtp/daraa/styles/topographic?f=json"
},
{
  "rel": "style",
  "type": "application/xml",
  "title": "topographic (SLD)",
  "href": "https://services.interactive-
instruments.de/vtp/daraa/styles/topographic?f=sld"
},
{
  "rel": "style",
  "type": "application/json",
  "title": "night (Mapbox)",
  "href": "https://services.interactive-
instruments.de/vtp/daraa/styles/night?f=json"
},
{
  "rel": "style",
  "type": "application/xml",
  "title": "night (SLD)",
  "href": "https://services.interactive-
instruments.de/vtp/daraa/styles/night?f=sld"
}
...

```

In the future, the POST method could be supported, too, to add or remove links for each collection.

C.2.6. Conformance declaration

Path: `/conformance`

Supported methods: GET

NOTE This is a placeholder. Eventually a URI for the conformance class needs to be specified and servers declaring conformance to the Styles API would include the URI in the `conformsTo` array.

C.3. Open issues and future work

The Styles API was implemented and tested during VTPExt. However, a number of open issues and future work items have been identified that need more investigation, discussion and testing.

A key issue is the differences between the concepts that are implemented in the different styling languages and tiled feature data encodings. These differences have an impact on API implementations that have not yet been fully analyzed.

Examples are:

- The Feature Tiles API design supports both fetching tiles with a single layer (`/collection/{collectionId}/tiles`) as well as tiles with multiple layers (`/tiles`). While a Mapbox Vector Tile (MVT) can contain multiple layers, a GeoJSON document contains features from a single feature collection. The `/tiles` path, therefore, only returns MVT, but not GeoJSON.
- A Mapbox style document specifies a single style for multiple layers. This is different in SLD (and the WM(T)S interfaces), where styles are layer-specific.
- How many styles can be expressed in a style sheet? One in Mapbox Style, multiple in SLD.
- Are styling rules explicitly bound to a data source / layer? Yes in Mapbox Style, no in SLD.

This has three consequences:

1. In practice, dependencies exist between the representations used for a style sheet and the feature or feature tile encoding. Not every combination will work. The conceptual model is therefore essentially including a good understanding of how the different mapping languages and feature (tile) encodings implement, profile, and extend that model.
2. There will be limitations for converting the original style sheet of a style that has been used to create or update the style to the server to another styling language. Due to the different concepts, conversions will in general not be lossless, not be bi-directional and sometimes not be feasible at all. One approach to address this could be to allow or require that style producers PUT the different style sheets for a style separately, one PUT request for each styling language.
3. In VTPExt, Mapbox Style has been the primary styling language. This is reflected in the API design, which is closer to the Mapbox model than the SLD model. A geospatial engineer can convert a Mapbox style to an SLD by using a single feature type style with the same name for each layer. This puts constraints on the SLD that could be used in VTPExt.

There are some other specific topics that are related to this and that need further investigation in

order to improve the API and verify that the Styles API design is sound, but still straightforward to implement and use:

- Authoring styling rules for a collection (layer) requires knowledge about the properties that are available so that the styling rules can select the relevant features for which a particular styling rule applies. Currently, no mechanism exists in the WFS 3.0 Core architecture to determine the queryable properties for a collection. Such a mechanism is required and should provide property information for both feature access in the `items` path and for tiled feature data access in the `tiles` path. In OGC Testbed-14, a proposal was made to support an additional resource to list the queryable properties of a collection. This would not only support authoring feature queries, which was the background for the proposal in Testbed-14, but also the authoring of styles. The Complex Feature Handling Engineering Report (18-021) states in section 7.3.3:

"In order to support clients to construct queries, the feature properties that may be queried should be enumerated for each feature type. This could be included in the feature collection metadata or, which is probably preferable, it could be made available in an additional resource listing all queryable properties. For example at `/collections/{collectionId}/queryables`. The result could be a JSON object with a member for each property of the feature. The value of the member could be used to identify the data type. If the property value is a related object in the dataset, the queryables resource of that collection could be referenced. For nested objects, the compound attribute values could be used explicitly."

Note that this is separate from providing a complete schema for features. A schema provides a complete syntactic definition of a feature encoding in a representation like XML or JSON, typically for validation purposes. Schema languages will be much richer and will support more complex syntactic rules, but are also more complex to parse.

- Names of feature types / collections and of properties may differ between feature representations (for example, names in an XML/GML representation include a namespace, in GeoJSON or MVT this is not the case). That is, an SLD/SE representation may include namespaces in the references to feature types and properties since it typically is based on the GML application schema for the data, but when applying it to GeoJSON, a geospatial engineer would have to ignore the namespaces.
- Different representations may have different names for other reasons, too. For example, one participant may use different property names in GeoJSON than in GML or the property name in a Shapefile may be different because of length restrictions, etc.
- In general, style representations differ in their capabilities, in particular for styling rules, so the result of applying SLD/SE or Mapbox style sheets to a feature will result in different results in non-trivial cases.
- Both Mapbox Style and SLD/SE reference external resources (sprites, glyphs, symbols), but each is used in different ways in the styling languages. As these resources are essential parts of a style, they should probably be resources in the API, too.

- Mapbox Style is more a "Map Definition API" than just a styling language as it includes other information such as which background map to use.
- Validation of content that is POSTed or PUT to the server has not been considered yet, but would be important to avoid interoperability issues, especially if clients use style sheets provided by a server.
- Security and access control aspects have not been considered in VTPEExt, but in general, server administrators will want to restrict who can manage styles; so at least the POST/PUT/DELETE operations will often be secured. This was not relevant in the pilot, but in general it will be.
- Additional style metadata was out-of-scope for VTPEExt, but in practice it will be important to be able to manage additional metadata for the styles, so that users can have better information to find and select styles for their intended use. One option could be another resource `/styles/{styleId}/metadata` that could be created using POST to `/styles/{styleId}` with a metadata object or be managed using GET/PUT/DELETE directly.
- Currently, the Styles API is written so that each dataset has its own style set. This was sufficient for VTPEExt, but in general the same style sheet is applicable to multiple datasets (at least for all datasets using a common application schema). This raises the question of whether the Styles API should be decoupled from the API for publishing datasets.
- The lack of registered media type registrations for all styling languages has already been discussed earlier in the chapter. In VTPEExt, `application/json` has been used as the media type of the Mapbox style documents. More appropriate would be something like `application/vnd.mapbox-style` that OGC or Mapbox should register with IANA, if it would be used beyond the pilot. A similar registration would be needed for SLD/SE.

Appendix D: WMTS 1.0 Styles API Profile Specification

NOTE

This has been submitted as [OGC change request #566](http://ogc.standardstracker.org/show_request.cgi?id=566) [http://ogc.standardstracker.org/show_request.cgi?id=566].

D.1. Introduction

This WMTS 1.0 profile defines a set of RESTful endpoints that provide a client with the ability to GET, PUT, and DELETE style definitions, and a KVP GetStyle operation.

D.2. Declaration Of Profile

A WMTS server that provides these RESTful endpoints and/or KVP operations must declare the profile URI [TBD] in its capabilities document.

D.3. Style URL Templates (RESTful)

The RESTful endpoints that provide a client with the ability to GET, PUT, and DELETE style definitions are declared on a per-layer basis through URL templates with a resourceType of "style". These templates must contain exactly one template variable: `{style}`. The following is an example:

Example 4. Style URL Template

```
<ResourceURL format="application/vnd.ogc.sld+xml" resourceType="style"
  template="https://example.com/wmts/1.0.0/layers/MyLayer/{style}.sld"/>
```

A layer should declare a style URL template for each style encoding (format) that it supports, and must not declare any style URL templates if it is unable to provide or accept style definitions for the layer.

If an endpoint is accessed with an HTTP method that it doesn't support, the server must return an HTTP 405 "Method Not Allowed" exception. If the user simply doesn't have the proper authorization to access the endpoint with that method, the server may instead return an HTTP 401 "Unauthorized" exception with the appropriate WWW-Authenticate header.

The following HTTP methods are defined for these endpoints:

D.3.1. GET

Returns the definition of the style with the identifier specified by `{style}` in the encoding declared by the URL template. If the encoding is SLD, an optional `sld_version` parameter may be used to indicate the desired SLD version. If this parameter is not present, or if the WMTS server does not support the parameter, or if the WMTS server does not support the desired SLD version, then the

server should return SLD version 1.1.0.

If the encoding is SLD, the returned SLD must contain exactly one NamedLayer element, which must have the same identifier as the layer that declared the URL template. That NamedLayer element must contain exactly one UserStyle element, which must represent the requested style.

D.3.2. PUT

Sets the definition of the style with the identifier specified by `{style}` in the encoding declared by the URL template. If the layer already has a style with that identifier, then the style's definition is replaced with the newly-supplied one. Otherwise, a style with that identifier is created for the layer.

If the value of an HTTP Content-Type request header is not equivalent to the encoding (format) declared by the URL template, the server must either throw an HTTP 415 (Unsupported Media Type) exception or accept it anyways (if it supports that content type).

If the encoding is SLD, the request body must be an SLD containing exactly one NamedLayer element which must have the same identifier as the layer that declared the URL template. That NamedLayer element must contain exactly one UserStyle element. NamedStyle elements are not supported because they do not provide style definitions. The style must refer to valid feature sets and feature-set properties. If any of these requirements are violated, the server should throw an HTTP 400 (Bad Request) exception. The mechanism for determining the available pool of feature sets and their schemas is currently beyond the scope of the WMTS interface. However, the [Limitations And Future Work](#) section below explores two possible approaches.

It may be the case that a server has particular requirements for syntax or structure of its style identifiers. For example, there may be a restriction on the number of characters, or certain characters may be illegal. The server should return an HTTP 400 (Bad Request) exception if an attempt is made to create a style with an invalid identifier. In such a situation, the response body should contain an explanation of why the identifier is invalid.

On success, an HTTP 200 (OK) or HTTP 204 (No Content) status code should be returned.

D.3.3. DELETE

Deletes the style with the identifier specified by `{style}`. On success, an HTTP 200 (OK) or HTTP 204 (No Content) status code should be returned. If no such style existed, the server may instead choose to return a 404 (Not Found) status code.

The WMTS specification requires all layers to have at least one style. This profile does not dictate the behavior when an attempt is made to delete the last style of a layer. Two possibilities are: a) the request is rejected with an HTTP 400 (Bad Request) exception, or b) the layer ceases to be exposed through the WMTS interface.

D.3.4. OPTIONS

Requests which HTTP methods are available for the endpoint. As per standard HTTP practice, the response must include an "Allow" header whose value is a comma-separated list of allowed HTTP methods for the endpoint.

Example 5. An "Allow" Response Header

```
Allow: GET,PUT,DELETE,HEAD,OPTIONS
```

A client may use this method to determine which HTTP methods are available for an endpoint. Alternatively, it is acceptable for the client to try the request and accept the possibility of an HTTP 405 "Method Not Allowed" response.

D.4. GetStyle Operation (KVP)

In addition to the above RESTful endpoints, this profile defines the following KVP operation.

operation name: **GetStyle**

parameters:

- **layer** (*required*)
- **style** (*required*)
- **format** (*optional; defaults to "application/vnd.ogc.sld+xml"*)
- **sld_version** (*optional; relevant only if format is an SLD format; defaults to "1.1.0"*)

This operation returns the definition of the specified style of the specified layer in the specified encoding (format).

If the WMTS server supports this operation, it must be declared in the OperationsMetadata section of the capabilities document.

If an SLD encoding is requested, the optional **sld_version** parameter may be used to indicate the desired SLD version. If this parameter is not present, or if the WMTS server does not support the parameter, or if the WMTS server does not support the desired SLD version, then the server should return SLD version 1.1.0.

If an SLD encoding is requested, the returned SLD must contain exactly one NamedLayer element, which must have the same identifier as the layer that declared the URL template. That NamedLayer element must contain exactly one UserStyle element, which must represent the requested style.

Vendor-specific extensions to this operation (such as making the **style** parameter optional to allow the client to request an SLD representing all of the styles of a layer) are allowed.

This profile does not define a PutStyle or DeleteStyle operation, since such operations cannot adequately be implemented as traditional KVP requests.

D.5. Security Considerations

The HTTP PUT and DELETE methods should be privileged methods that only certain users are granted the ability to invoke. It may also be the case that the ability to invoke these methods varies

from layer to layer, or even from encoding to encoding. That is, a user may have the authorization to define and adjust the styles for some layers but not others. The means to define such access-control rules is server-specific and beyond the scope of the WMTS specification.

D.6. Limitations And Future Work

In order to create new styles or to modify existing styles, it is often necessary for the client to know what feature sets are available and what their schemas are. A client can determine some of this information by examining existing styles. However, there does not yet exist a mechanism for determining the full set of available feature sets and their schemas.

A few possible approaches were explored during the OGC VT-Pilot Extension project, but none of them thoroughly enough to provide a basis for a formal proposal. Two promising approaches are:

1. Make use of the GetAssociations operation proposed by the [Web Integration Service Engineering Report](https://portal.opengeospatial.org/files/?artifact_id=71525&version=1) [https://portal.opengeospatial.org/files/?artifact_id=71525&version=1] in OGC Testbed 12. This provides a bridge between a WMTS service and one or more companion WFS and/or WCS services, allowing the client to query the WFS service(s) for the list of available feature sets and their schemas. It also provides other advantages such as alleviating the need to define a transactional WMTS service, since any manipulation of the feature set(s) rendered by a WMTS layer could be performed through the appropriate WFS or WCS service.
2. Define a new WMTS endpoint that returns a list of the feature sets that are available (regardless of whether or not they're actually rendered by any of the existing styles of any of the currently-defined layers), and provide an endpoint for each of those feature sets that returns that feature set's schema. This is a more direct approach, but adds redundancy between the WMTS and the WFS that could be avoided by making better use of the associations between services.

Appendix E: Revision History

Table 18. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
December 22, 2018	J. Yutzler	.1	all	Preliminary ER
February 15, 2019	J. Yutzler	.1	all	Draft Final ER
March 15, 2019	J. St-Louis	.1	appendixes	Added GeoPackage prefix to section titles of GeoPackage extensions

Appendix F: Bibliography

1. Meek, S.: OGC Vector Tiles Pilot: Summary Engineering Report. OGC 18-086, Open Geospatial Consortium, <https://docs.opengeospatial.org/per/18-086r1.html> (2019).
2. Ingensand, J., Maia, K.: OGC Vector Tiles Pilot: Tiled Feature Data Conceptual Model Engineering Report. OGC 18-076, Open Geospatial Consortium, <https://docs.opengeospatial.org/per/18-076.html> (2019).
3. Bocher, E., Ertz, O.: OGC Symbology Conceptual Model: Core part. OGC 18-067, Open Geospatial Consortium, <https://portal.opengeospatial.org/files/80686> (2018).
4. Vretanos, P.A.: OGC Vector Tiles Pilot: WFS 3.0 Vector Tiles Extension Engineering Report. OGC 18-078, Open Geospatial Consortium, <https://docs.opengeospatial.org/per/18-078.html> (2019).
5. Vretanos, P.A.: OGC Vector Tiles Pilot: WMTS Vector Tiles Extension Engineering Report. OGC 18-083, Open Geospatial Consortium, <https://docs.opengeospatial.org/per/18-083.html> (2019).
6. Yutzler, J.: Vector Tiles Pilot Extension Engineering Report. OGC 18-101, Open Geospatial Consortium, https://portal.opengeospatial.org/files/?artifact_id=79181&version=1 (2019).