

OGC Testbed-15
Open Portrayal Framework Engineering Report

Table of Contents

1. Subject	4
2. Executive Summary	5
2.1. Document contributor contact points	5
2.2. Foreword	6
3. References	7
4. Terms and definitions	8
4.1. Abbreviated terms	9
5. Overview	10
6. Open Portrayal Framework Scenario	11
6.1. Scenario Details	12
7. Emerging OGC Web APIs	16
7.1. API Styles and Conceptual Model for Style Encoding & Metadata	16
7.2. API Maps and Tiles	17
7.3. API Images	18
7.4. API Changesets	19
8. GeoPackage	20
8.1. Introduction	20
8.2. GeoPackage Extensions and Profiles	20
8.3. GeoPackage to support the Open Portrayal Framework	23
8.3.1. Conveying Styling Rules	24
8.3.2. Coupling Layers and Styles	24
8.3.3. Making Styles Accessible Operationally	25
8.4. Implementations and Lessons Learned	25
9. Technical Discussions	27
9.1. Background Color	27
9.2. Styleable Layer Set	33
9.3. Conversions Between style Encodings	34
9.3.1. GeoSolutions	34
9.3.2. Ecere	36
9.4. Sprites	38
9.5. Mediatypes for an Open Portrayal Framework	39
10. Further Information and Videos	41
Appendix A: Annex A: OPF Implementations	42
A.1. Implementation GeoSolutions	42
A.1.1. Available services and layers	42
A.1.2. OGC API implementation approach	43
A.1.3. Features API endpoint	44
A.1.4. Images API endpoints	46

A.1.5. Tiles API endpoints	47
A.1.6. Styles API endpoints	49
A.2. Client Components and Scenario Details	50
A.2.1. Links	62
A.3. Implementation interactive instruments	63
A.3.1. Overview	63
A.3.2. New API building blocks	64
A.4. Implementation (Ecere)	68
A.4.1. Service components	69
A.4.2. GeoPackage producer	83
A.4.3. Client components	83
A.5. Implementation CubeWerx	94
A.5.1. Available services and layers	94
A.5.2. Tiles API endpoints	95
A.5.3. Styles API endpoints	96
A.5.4. Image API endpoints	96
A.6. Implementation Compusult	97
A.6.1. Overview	97
A.6.2. Scenario Details	98
A.7. Implementation Image Matters	123
Appendix B: Revision History	125

Publication Date: 2020-02-06

Approval Date: 2019-11-22

Submission Date: 2019-10-31

Reference number of this document: OGC 19-018

Reference URL for this document: <http://www.opengis.net/doc/PER/t15-D015>

Category: OGC Public Engineering Report

Editor: Martin Klopfer

Title: OGC Testbed-15: Open Portrayal Framework Engineering Report

OGC Public Engineering Report

COPYRIGHT

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

WARNING

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

LICENSE AGREEMENT

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

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

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

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

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

indicate compliance with any LICENSOR standards or specifications.

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

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

Chapter 1. Subject

This Engineering Report (ER) describes the OGC Testbed-15 Open Portrayal Framework (OPF) Thread requirements, scenario, high-level architecture, and solutions. Main topics addressed in the OPF Thread include style changing and sharing, converting style encodings, client- / server-side rendering of vector- and raster data and data provision in denied, disrupted, intermittent, and limited bandwidth (DDIL) infrastructure situations. The work in the OPF Thread was focused on an OGC Application Programming Interface (API) oriented approach.

Chapter 2. Executive Summary

This Engineering Report provides a detailed summary of the *Open Portrayal Framework* (OPF) Thread in OGC Testbed-15, executed from April to November 2019.

The *Open Portrayal Framework* is a set of emerging specifications that support interoperable portrayal of heterogeneous geospatial data. The Open Portrayal Framework facilitates the rendering of geospatial data in a uniform way, according to specific user requirements. The primary topics addressed in the OPF thread covered supporting style sharing and updates, client- and server-side rendering of both vector- and raster data, and converting styles from one encoding to another; all following a single conceptual style model. In addition, the requirement to render data according to style definitions in a scenario with denied, disrupted, intermittent, and limited bandwidth (DDIL) infrastructure has been addressed.

This Engineering Report describes the *Open Portrayal Framework* requirements, scenario, high-level architecture, and solutions that were developed. Further details on the work carried out in the OPF Thread are provided in the following ERs:

- [OGC Testbed-15: Encoding and Metadata Conceptual Model for Styles Engineering Report](http://www.opengis.net/doc/PER/t15-D011) [http://www.opengis.net/doc/PER/t15-D011]

The shift from traditional Web Services towards Web APIs has strongly influenced the work in the OPF Thread and as a result three draft specification Engineering Reports have been developed:

- [OGC Testbed-15: Styles API Engineering Report](http://docs.opengeospatial.org/per/19-010r2.html) [http://docs.opengeospatial.org/per/19-010r2.html]
- [OGC Testbed-15: Maps and Tiles API Engineering Report](http://docs.opengeospatial.org/per/19-069.html) [http://docs.opengeospatial.org/per/19-069.html]
- [OGC Testbed-15: Images and ChangesSet API Engineering Report](http://docs.opengeospatial.org/per/19-070.html) [http://docs.opengeospatial.org/per/19-070.html]

A high-level summary is available in the following ER:

- [OGC Testbed-15: Portrayal Summary Engineering Report](http://www.opengis.net/doc/PER/t15-D017) [http://www.opengis.net/doc/PER/t15-D017]

2.1. Document contributor contact points

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

Contacts

Name	Organization	Role
Martin Klopfer	Frisia IT	Editor
Jeff Yutzler	Image Matters	Contributor
Joe Jagiella	Image Matters	Contributor
Clemens Portele	interactive instruments	Contributor
Keith Pomakis	CubeWerx Inc.	Contributor

Name	Organization	Role
Andrea Aime	GeoSolutions	Contributor
Stefano Bovio	GeoSolutions	Contributor
Jerome St-Louis	Ecere	Contributor
Joan Maso Pao	Universitat Autònoma de Barcelona (CREAF)	Contributor
Jeff Harrison	AGC	Contributor
Matt Sorenson	AGC	Contributor
Carl Reed	OGC	Contributor
Ingo Simonis	OGC	Contributor

2.2. Foreword

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

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

Chapter 3. References

The following normative documents are referenced in this document.

- [OGC 06-121r9, OGC® Web Services Common Standard \(2010\)](https://portal.opengeospatial.org/files/?artifact_id=38867&version=2) [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- [OGC: OGC 02-070, Styled Layer Descriptor, Version 1.0 \(2002\)](http://portal.opengeospatial.org/files/?artifact_id=1188) [http://portal.opengeospatial.org/files/?artifact_id=1188]
- [OGC: OGC 05-078r4, Styled Layer Descriptor, Version 1.1 \(2007\)](http://portal.opengeospatial.org/files/?artifact_id=22364) [http://portal.opengeospatial.org/files/?artifact_id=22364]
- [OGC: OGC 17-069r3, OGC API - Features - Part 1: Core \(2019\)](http://docs.opengeospatial.org/is/17-069r3/17-069r3.html) [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html]
- [OGC: OGC 05-077r4, OpenGIS Symbology Encoding Implementation Specification \(2006\)](http://portal.opengeospatial.org/files/?artifact_id=16700) [http://portal.opengeospatial.org/files/?artifact_id=16700]

Chapter 4. Terms and definitions

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

Style

a sequence of rules of symbolizing instructions to be applied by a rendering engine on one or more features and/or coverages

Style encoding

specification to express a style as one or more files

NOTE In Testbed-15 Mapbox Styles, OGC SLD versions 1.0 and 1.1 are used.

Icons

In computing, an icon is a pictogram or ideogram displayed on a computer screen in order to help the user navigate a computer system. [https://en.wikipedia.org/wiki/Icon_\(computing\)](https://en.wikipedia.org/wiki/Icon_(computing))

Layer

A layer is an abstraction of reality specified by a geographic data model (feature, coverage...) and represented using a set of symbols (Style) to plot it. A layer contributes to a single geographic subject and may be a theme.

Sprites

A sprite is a computer graphics term for a two-dimensional bitmap that is integrated into a larger scene.

Stylesheet

representation of a style in a style encoding.

Style metadata

essential information about a style needed to support users to discover and select styles for rendering their data and for visual style editors to create user interfaces for editing a style.

Coverages API

OGC API-Coverages provides the API building block to access coverages as defined by the Coverage Implementation Schema (CIS) 1.1 on the Web.

Features API

OGC API-Features provides the API building block to create, modify and query features on the Web.

Maps API

OGC API-Maps provides the API building block to describe, build and retrieve web maps.

Styles API

OGC API-Styles is a Web API that enables map servers and clients as well as visual style editors to manage and fetch styles.

Tiles API

OGC API-Tiles provides API building block to describe, build and retrieve tiles from any resource that can be subdivided in a regular set of tiles (e.g., maps, features and coverages).

Web API

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

4.1. Abbreviated terms

API

Application Programming Interface

OGC

Open Geospatial Consortium

SLD

OGC Styled Layer Descriptor

SE

Symbology Encoding

WCS

Web Coverage Service

WFS

Web Feature Service

WMS

Web Map Service

WMTS

Web Map Tile Service

Chapter 5. Overview

An overview of the major work items of the Testbed-15 Open Portrayal Framework thread is shown in [Figure 1](#).

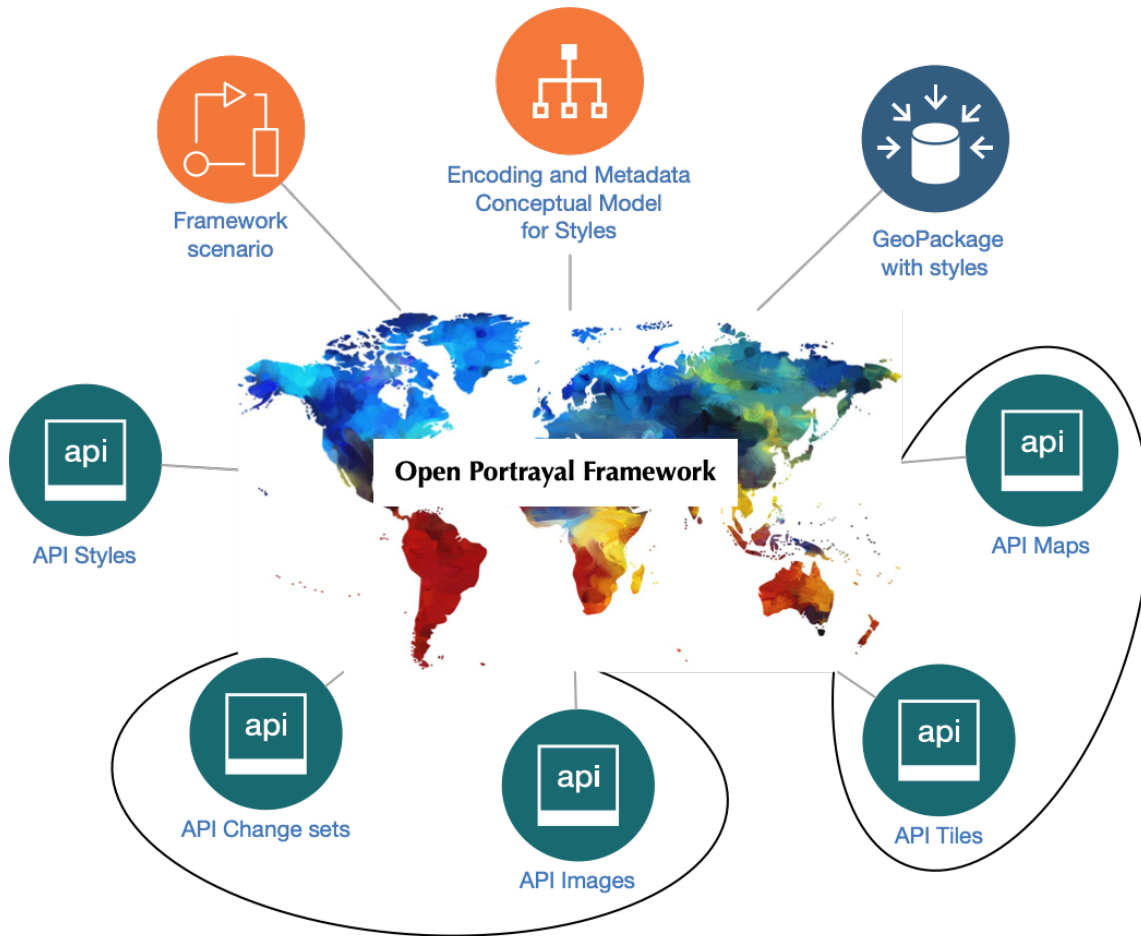


Figure 1. Overview of the Testbed-15 Open Portrayal Framework major work items

The scenario for the emerging Open Portrayal Framework is described in detail in the following chapter [Open Portrayal Framework Scenario](#). The scenario discussion explains the driving requirements for the various developments conducted in Testbed-15. Among these developments are a series of draft OGC Web APIs.

The draft OGC APIs are discussed from a more abstract viewpoint, with some of the major discussion items and design decisions being highlighted in the chapter [Emerging OGC Web APIs](#). A detailed documentation of the APIs can be found in the respective ERs that are listed in the [Executive Summary](#).

An enhanced GeoPackage model was developed to facilitate advanced styling in offline situations. The main results are discussed in the chapter [GeoPackage](#).

The ER concludes with the documentation of a number of participant general discussions and design decisions required to complete the OPF Thread. These are reported in the chapter [Technical Discussions](#).

A detailed description of implementation aspects and presentation of the results provided by the participants is presented in [Annex A: OPF Implementations](#).

Chapter 6. Open Portrayal Framework Scenario

The goal of the Testbed-15 Open Portrayal Framework thread was to implement a data discovery, access, and styled rendering scenario. The scenario included data updates performed as background tasks and support for online/offline functionality. The scenario is illustrated in Figure 2. Conceptually close steps are colored in matching shades of gray (i.e. steps 1-4, 5-6, 7-9, and 10-11). Logically differentiated *Users 1-3* in this scenario can be realized as a single physical user.

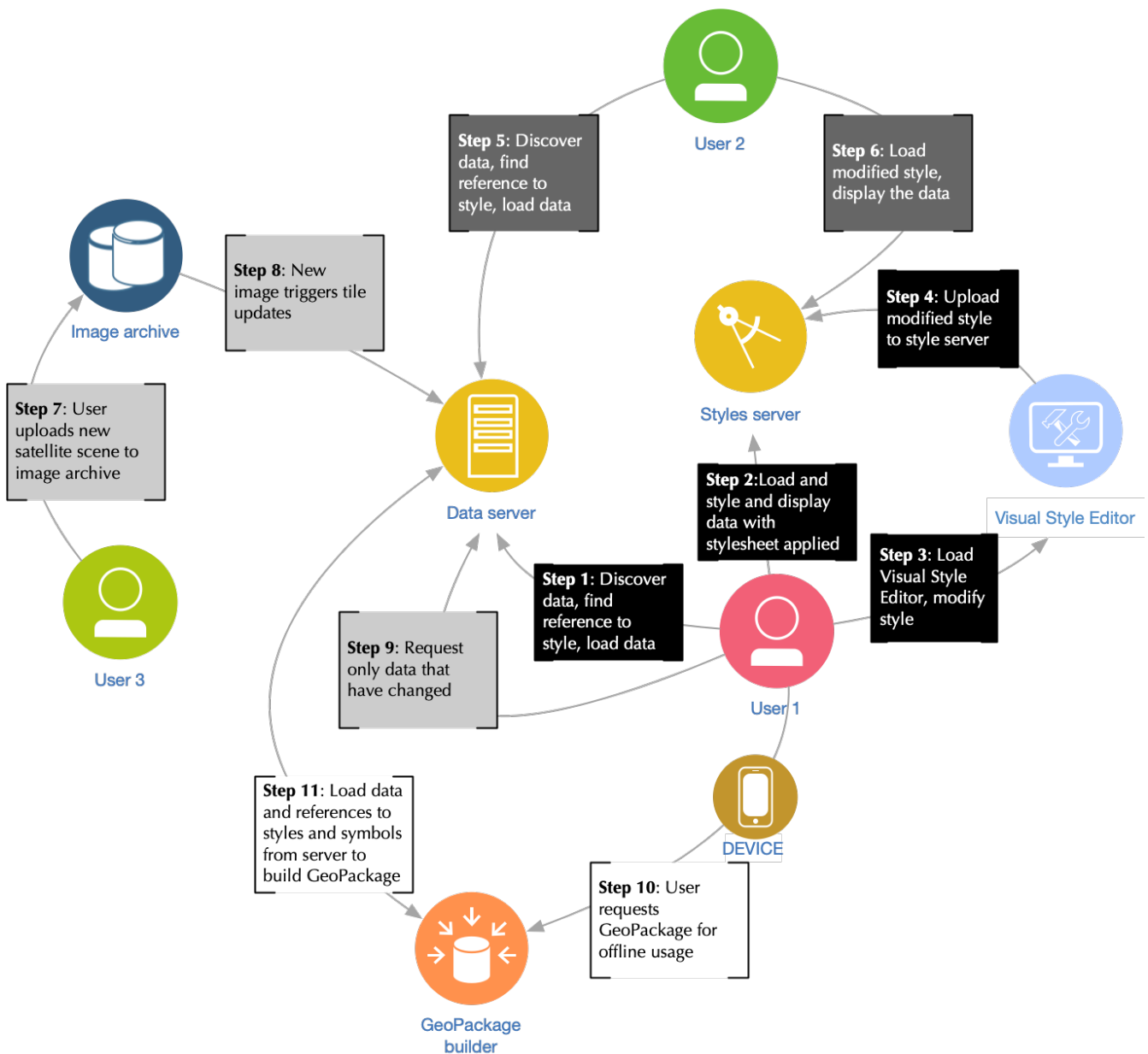


Figure 2. The Testbed-15 Open Portrayal Framework scenario

Step 1: The user discovers API endpoints that provide vector, raster, or tiled vector or tiled raster data, maps, or coverages. The data can be retrieved via implementations of the OGC API – Features standard, or via emerging OGC API specifications for -Tiles and -Maps in addition to traditional OGC Web Services (OWS) such as WMS, WMTS, or WFS/WCS. Corresponding styles are made available at a dedicated endpoint. Links between the two allow understanding which style works with which data, and vice versa.

Step 2: User-1 loads styling instructions from the style server via the Styles API. Requirement: Style information shall be available at a dedicated endpoint that allows users to manage and modify styles.

Step 3: A user can modify any style using Visual Style Editors. These tools allow modifying styles and ideally transform styles from one encoding to another (e.g. from OGC SLD to Mapbox Style or vice versa).

Step 4: Styles are retrieved from the style API endpoint, modified in the Visual Style Editor, and uploaded to the style server again. The modified style is available to other users.

Step 5: Other users (here User-2) discover the same data. The references to the applicable styles have not changed.

Step 6: User-2 follows the reference to the styles and loads these. The rendered data now looks as defined by User-1.

Step 7: A third user or remote system makes new satellite imagery available to an image archive.

Step 8: This process triggers the re-building of individual tiles as served by Web API endpoints (APIs -Tiles or -Maps).

Step 9: Users interested in the changes since a given checkpoint can access the changed tiles (i.e. the change set) exclusively. This would allow large tile stores to be updated incrementally.

Step 10/11: In the future, the full scenario should be available to both online and offline situations. In this Testbed the focus was on adding styles and other data layers to OGC GeoPackages.

6.1. Scenario Details

The following paragraphs show the various implementations that have been created by participants in Technology Integration Experiments (TIEs) conducted in Testbed-15 and provide additional information.

Step 1 The user discovers API endpoints that provide vector, raster, or tiled vector or tiled raster data, maps, or coverages. The user loads the data and finds references to corresponding styles to render the data correctly (or retrieves pre-rendered tiles that apply the stylesheets).

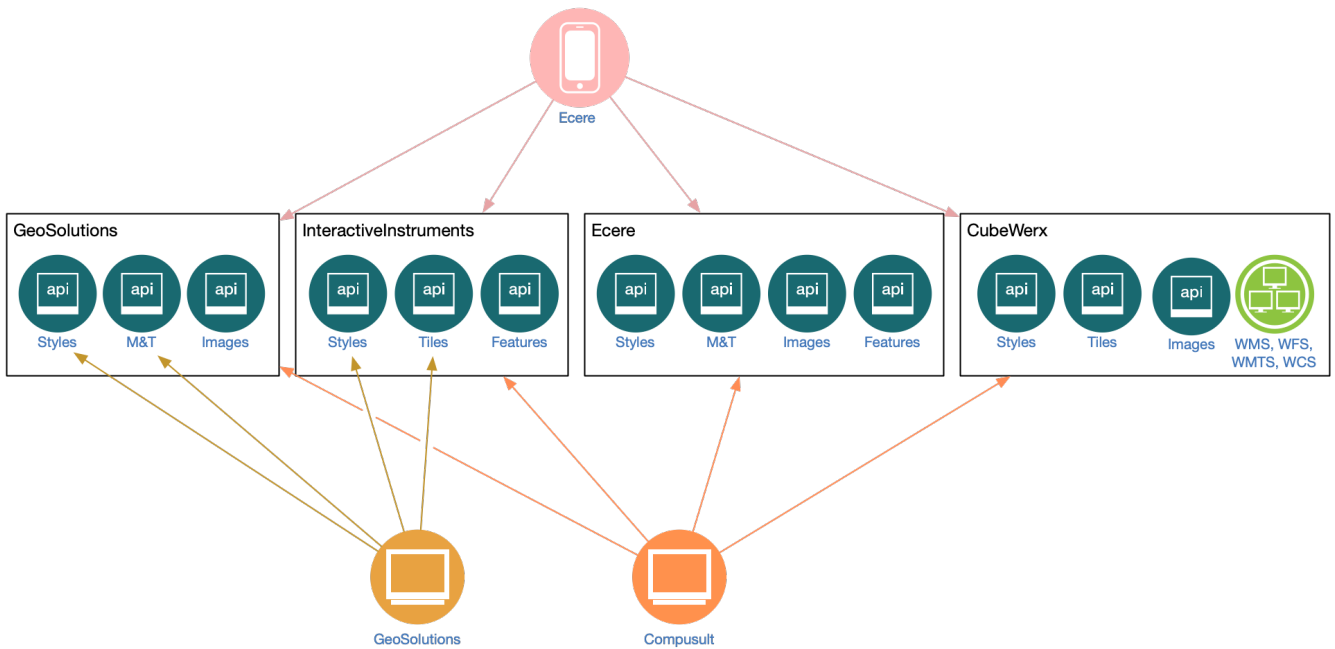


Figure 3. Step 1 & 2: Discover data and styles, load data, display data

Step 2 The references to the applicable styles point to a server that provides access to the styles via the Styles API. The user loads the style and applies the stylesheet to the data. Not happy with the result, the user decides to modify the style.

Three clients have been produced in Testbed-15 that interact with the various service endpoints as illustrated below.

Step 3 A user can modify any style using a Visual Style Editor. These tools allow modifying styles and ideally transform styles from one media type to another (e.g. from OGC SLD to Mapbox Style or vice versa).

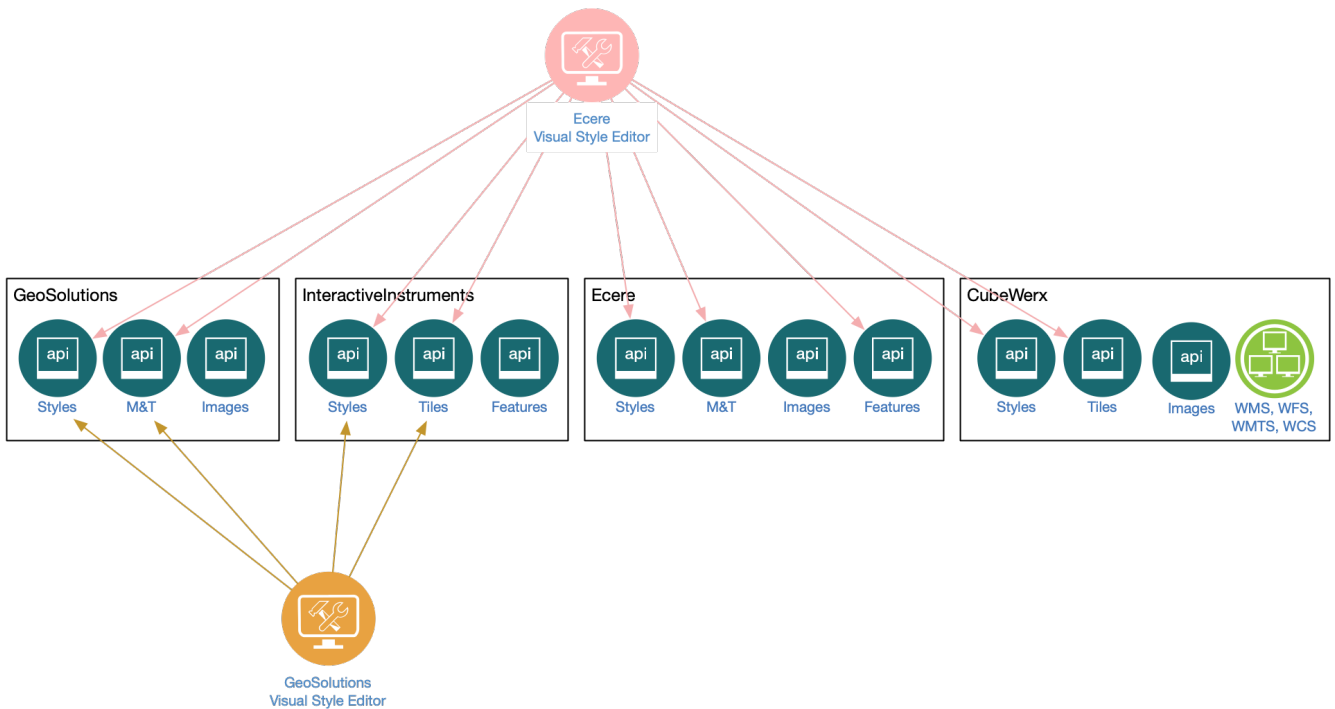


Figure 4. Step 3: Load Visual Style Editor, change and update style

Two Visual Style Editors were implemented in Testbed-15. Both are described and illustrated in

[Annex A: OPF Implementations](#) and illustrated in the videos listed in the section [Videos and Outreach Material](#). Both editors are capable of retrieving a style from a style server, updating the style locally, and re-distributing the style by uploading it to the style server.

Step 4 Once the stylesheet has been adapted to the user’s needs, the user may decide to make the modified style available via the Style API. The Style API provides transactional capabilities for that purpose.

Step 5 In Step 5, another User-2 discovers the same data as User-1. The references in the data to the applicable styles have not changed, since the data server is not aware of any modifications to styles. User-2 retrieves the data via the various API endpoints (API Maps or Tiles) similar to User-1.

Step 6 User-2 discovers references to the applicable styles and loads these similar to User-1 before. Except, that the loaded stylesheets now also contain the styling instructions which User-1 defined before, using the Visual Style Editor.

Step 7 & 8 Other users or systems make new satellite imagery available to an image archive. This process triggers the re-building of individual tiles as served by tile service endpoints.

The steps involve an implementation of the new OGC API - Images draft specification that allows uploading images to an image archive. The API basically functions like a catalog API and is further described in the section [API Images](#). Three implementations of this API have been developed in the Testbed-15 activities.

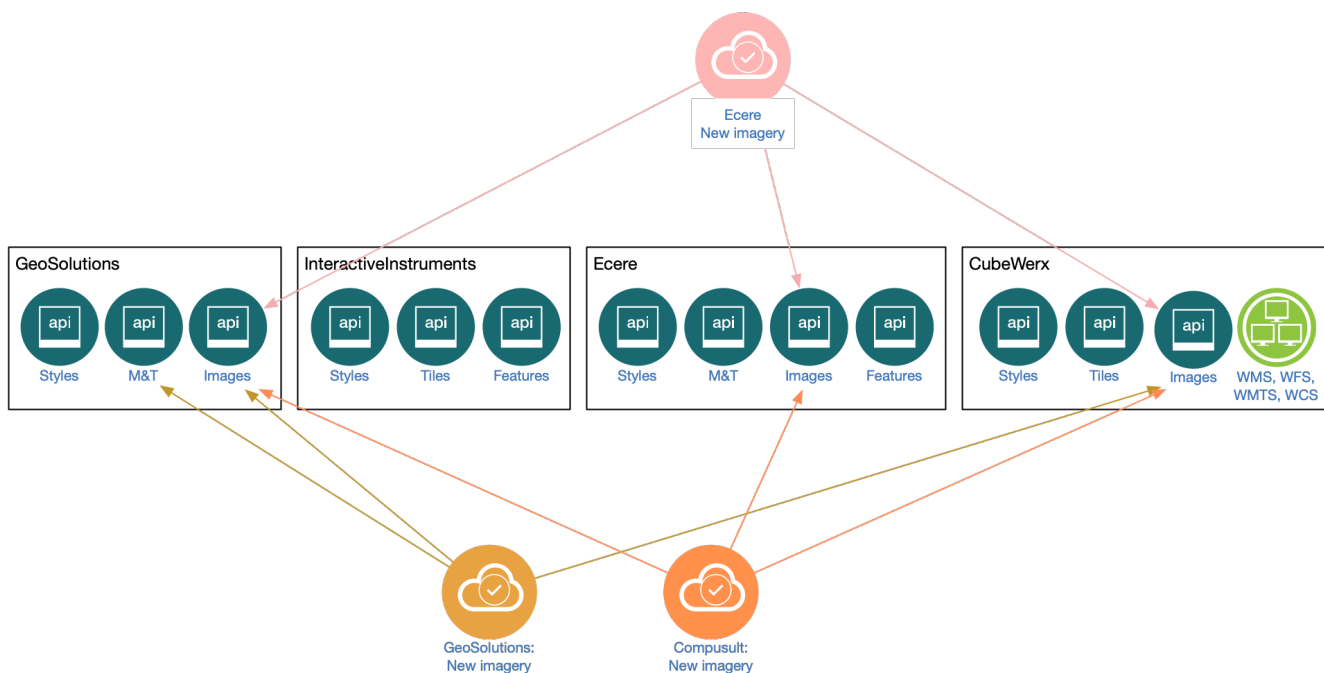


Figure 5. Step 7 & 8: New sat-scene added to image server and update tiles behind the scene

Step 9 User-1 needs to have the latest data for a given region. To avoid loading the entire data set again, User-1 requests all changed data since the last checkpoint (here: since the last request). Because new imagery data is now available, the data server provides a set of updated tiles.

The step makes use of the draft OGC API - Change Set specification as documented in the section [API Changesets](#).

Step 10 & 11 The Open Portrayal Framework is envisioned to provide full support for online/offline

mixed environments. For the offline situation, GeoPackages are used as a data container that supports a draft extension for styles and symbols. The necessary extensions are described in the section [GeoPackage](#).

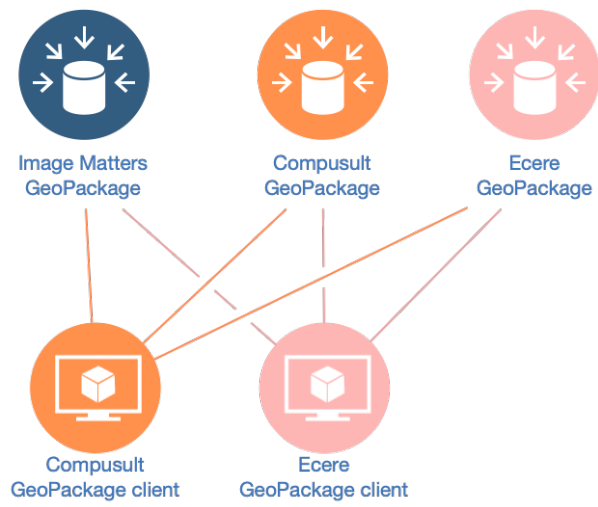


Figure 6. Step 10 & 11: Offline handling using Geopackage

GeoPackage builders ideally load all data using the same API endpoints as the users before.

Chapter 7. Emerging OGC Web APIs

Testbed-15 required revision of the Web Map Tile Service (WMTS) standard to align the service definition with the emerging OGC API family of standards and specifications that are documented using OpenAPI and to allow for image transactions and style support. Early in the process, the Testbed participants decided to implement these requirements in the form of a set of Web APIs, each API handling specific functionality. Eventually, the Open Portrayal Framework developed the following draft APIs.

- **Styles** [<http://www.opengis.net/doc/PER/t15-D012>] to manage styles at dedicated style endpoints
- **Maps** [<http://www.opengis.net/doc/PER/t15-D014>] to handle pre-rendered geospatial data
- **Tiles** [<http://www.opengis.net/doc/PER/t15-D014>] to handle tiled geospatial data available for rendering
- **Images** [<http://www.opengis.net/doc/PER/t15-D016>] to allow uploading of images to image archives, and
- **ChangeSets** [<http://www.opengis.net/doc/PER/t15-D016>] to allow clients to request only incremental changes since a given checkpoint.

The Maps and Tiles APIs as well as Images and Change Sets APIs are documented together in the *OGC Testbed-15 Images and Change Sets Engineering Report OGC 19-070* [<http://www.opengis.net/doc/PER/t15-D014>]. Eventually, the standards vetting and approval process for each API will be handled by a dedicated Standards Working Group within the OGC Standards Program. This work could result in five individual OGC APIs being documented and approved as OGC standards.

This report briefly discusses general aspects of each of these draft API specifications in the following chapters.

7.1. API Styles and Conceptual Model for Style Encoding & Metadata

The Styles API is a Web API that enables map servers and clients as well as visual style editors to manage and fetch styles. This draft API is documented in *OGC Testbed-15: Styles API Engineering Report (OGC 19-010)* [<http://www.opengis.net/doc/PER/t15-D012>]. The Styles API is consistent with the emerging OGC API family of standards. The Styles API implements the draft conceptual model for style encodings and style metadata as documented in the *OGC Testbed-15: Concept Model for Style Encoding & Metadata Model Engineering Report (OGC 19-023)* [<http://www.opengis.net/doc/PER/t15-D011>]. The draft conceptual model provides information for understanding styles intended usage, availability, compatibility with existing layers, as well as supporting style search. The conceptual model defines three main concepts:

- The **style** is the main resource.
- Each style is available in one or more **stylesheets** - the representation of a style in an encoding like *OGC Styled Layer Descriptor (SLD) 1.0* [http://portal.opengeospatial.org/files/?artifact_id=1188], *Symbology Encoding (SE) 1.1* [http://portal.opengeospatial.org/files/?artifact_id=16700], *Cascading Style Sheets (CSS)* [<https://docs.geoserver.org/latest/en/user/styling/css/index.html>], or *Mapbox GL*

[<https://docs.mapbox.com/mapbox-gl-js/style-spec/>]. Clients can use the stylesheet of a style that fits best based on the capabilities of available tools and their preferences.

- For each style there is **style metadata** available, with general descriptive information about the style, structural information (e.g., layers and attributes), and so forth to allow users to discover and select existing styles for their data.

The conceptual model was developed taking into account the knowledge gained in the draft [OGC Symbology Conceptual Model: Core part](https://portal.opengeospatial.org/files/89616) [https://portal.opengeospatial.org/files/89616] and the [Vector Tiles Pilot \(VTP\)](http://docs.opengeospatial.org/per/18-101.html) [http://docs.opengeospatial.org/per/18-101.html] activities. The VTP, conducted just before Testbed-15 started, produced a first prototype of a Styles API independent of the style encoding

Testbed-15 developed the Styles API to allow management, sharing, and usage of styles independently from concrete API endpoints. With the new API, datasets that are shared by other Web APIs implementing the OGC API - Features - Part 1: Core standard or the draft OGC API - Coverages or draft OGC API - Tiles specifications can reference applicable styles served by dedicated Style Web API endpoints.

The Styles Web API proof-of-concept supports full CRUD (create, read, update, delete) and thus features reusability of styles in an unprecedented way. In combination with Visual Style Editors, commonly used styles can be adapted, transformed from one serialization format to another (Testbed-15 experimented with OGC SLD/SE, MapBox GL, CSS, and CNOSIS internal style format) and shared again after modification.

Web APIs implementing the draft *OGC API - Maps* specification fetch styles and render spatial data (features or coverages) on the server. Map clients fetch styles and render spatial data (features or coverages) on the client.

In order to support styles, data APIs (for example, supporting the OGC API-Features standard and/or the draft OGC API Tiles) require additional capabilities, too. These are:

- List and manage the applicable styles per feature collection (path `/collections/{collectionId}`).
- Add a `queryables` resource (path `/collections/{collectionId}/queryables`) to support clients such as visual style editors to construct expressions for selection criteria in queries on features in the collection. "Queryable" means that the property may be used in styling rules or other filter expressions.

The Styles API specification uses [OpenAPI 3.0](http://spec.openapis.org/oas/v3.0.2) [http://spec.openapis.org/oas/v3.0.2] to specify the building blocks of the API.

7.2. API Maps and Tiles

The [OGC Web Map Tile Service](http://portal.opengeospatial.org/files/?artifact_id=35326) [http://portal.opengeospatial.org/files/?artifact_id=35326] (WMTS) implementation standard provides a standard based solution to serve digital maps using predefined image tiles. The standard builds on the *Capabilities model*, which is common to all OGC Web Service standards from the Key-Value-Pair (KVP) and Simple Object Access Protocol (SOAP) era, to advertise the tiles the service has available through a standardized declaration in the *ServiceMetadata* (aka *Capabilities*) document. This declaration defines the tiles available in each layer (i.e. each type of content), in each graphical representation style, in each format, in each

coordinate reference system, at each scale, and over each geographic fragment of the total covered area. The ServiceMetadata document also declares the communication protocols and encodings through which clients can interact with the server. Clients can interpret the ServiceMetadata document to request specific tiles.

The new version of WMTS should be better aligned with the emerging OGC API family of standards. These APIs

- leverage fundamental Web concepts,
- are documented using OpenAPI, and
- replace the *Capabilities concept* with a combination of landing page content and hypermedia controls.

This new family of (emerging) standards reflects the design change from existing Web service standards (W*S) towards Web APIs in order to serve geospatial data more natively on the Web. The Web API *OGC API - Maps and Tiles* draft specifications were further developed in Testbed-15 in response to these design changes.

As documented in [the beginning of this chapter](#), Testbed-15 participants developed a series of APIs to cover functionality offered by a revised, transactional WMTS. The draft Maps and Tiles APIs, documented in detail in [OGC document 19-069](#) [<http://www.opengis.net/doc/PER/t15-D014>] adopt an approach similar to that of the OGC API - Features - Part 1: Core standard. The OGC API - Tiles draft specification describes a service that retrieves data representations as tiles. Tiles are organized into Tile Matrix Sets consisting of regular tile matrices available at different scales or resolutions. The OGC API - Tiles draft specification is described as a building block that can be plugged into an OGC API - Features service to retrieve tiled feature data (sometimes called vector tiles or tiled vector data) or to an OGC API - Maps implementation to retrieve rendered tiles (sometimes called map tiles). Thus, depending on the source data and the behavior of the corresponding service, a client may receive tiled vector data (e.g. as Mapbox, GeoJSON, etc.), or raster data (e.g. TIFF, NetCDF, etc.). In the future, tiles could even support multiple data formats in a single tile container, i.e. raster data and vector data in a single tile. The same concept is applicable to layers. A tile container could contain multiple layers (that can be of different types as well).

The maps part of the draft OGC API - Maps and Tiles specification describes an API that presents some data as maps by applying a style. These maps can be retrieved as tiles (if OGC API - Tiles is also adopted by the implementation) or as maps of any size generated on the fly.

7.3. API Images

The requirement for a Web API to handle new imagery came up in the context of updated tile caches. Basically, a new image becomes available and needs to be added to an image archive. The provisioning of that image then triggers the re-generation of tiles in an adjacent tile store for the area affected by the new image (or even the entire area, depending on tiling concepts and methodologies). So far, a direct connection between the image archive and the tile store is assumed. In a future initiative, this connection can be further analyzed and further decoupled. The Images API is described in full detail in [OGC Testbed-15:Images and Changeset API Draft Specification Engineering Report](#) [<http://www.opengis.net/doc/PER/t15-D016>] (OGC 19-070).

The initial discussion focused on an image specific solution. However, it quickly became clear that the image archive case is nothing more than an asset catalog. An image catalog has just one main difference to most other catalogs: An image catalogue stores the images themselves and not just references, as most other catalogs do. This resulted in the [Spatial Temporal Asset Catalog \(STAC\)](https://github.com/radiantearth/stac-spec) [https://github.com/radiantearth/stac-spec] being considered for use in the Testbed.

STAC is a Web API-enabled generic catalog solutions for spatio-temporal assets. *"The Spatio Temporal Asset Catalog (STAC) specification aims to standardize the way geospatial assets are exposed online and queried. A 'spatiotemporal asset' is any file that represents information about the earth captured in a certain space and time. The initial focus is primarily on remotely-sensed imagery (from satellites, but also planes, drones, balloons, etc), but the core is designed to be extensible to SAR (Synthetic Aperture Radar), full motion video, point clouds, hyperspectral, LiDAR (Light Detection and Ranging) and derived data like NDVI (Normalized Difference Vegetation Index), Digital Elevation Models, mosaics, etc."* ([STAC](https://github.com/radiantearth/stac-spec)) [https://github.com/radiantearth/stac-spec].

7.4. API Changesets

The Changeset building block describes a mechanism for partial data updates to maintain synchronization between a server and a client cache that could be applied to any data service. Each retrieval of data from the server has associated a checkpoint identifier that is used in further communications. The Testbed-15 Open Portrayal Framework thread concentrated on updates to imagery stores. The filter mechanisms use checkpoint identifiers to constrain the response to tiles updated for a given target area since a provided checkpoint.

This work on *changesets* is closely related to the work on Delta Updates as performed in the Testbed-15 "Delta Updates" thread. In that thread, the goal was to develop a general solution to reduce the volume of transferred feature data in Denied, Degraded, Intermittent and Limited (DDIL) Bandwidth environments. *Delta Updates* explored how servers can decide to further reduce the amount of update data by leveraging classification schemes. Depending on the applied classification schema, the server may only send top priority updates in DDIL conditions and further lower priority updates, once conditions improve. The [OGC Testbed-15: Delta Updates Engineering Report](http://www.opengis.net/doc/PER/t15-D005) [http://www.opengis.net/doc/PER/t15-D005] describes the delta updates solution and describes how prioritized delta updates can be served using a transactional extension for the OGC API – Features and the WPS standard/OGC API – Processes in front of WFS instances.

Within the OPF thread, the participants decided to keep the classification scheme available, since this approach might prove beneficial in future work. For Testbed-15, all tiles have been part of the same class, i.e. the schema has been ignored at implementation stage.

The Changeset API is described in full detail in [OGC Testbed-15: Images and ChangesSet API Draft Specification Engineering Report](http://www.opengis.net/doc/PER/t15-D016) [http://www.opengis.net/doc/PER/t15-D016].

Chapter 8. GeoPackage

Testbed-15 contributed in a large extent to the development of the [Proposed OGC GeoPackage Enhancements Discussion Paper](https://portal.opengeospatial.org/files/?artifact_id=89670&version=1) [https://portal.opengeospatial.org/files/?artifact_id=89670&version=1] as published as OGC document 19-047. In this section, the main elements are briefly summarized. Focus is provided on experimentation experiences and lessons learned during the implementation phase of the *Testbed-15 Open Portrayal Framework Thread*.

8.1. Introduction

The Open Geospatial Consortium (OGC) GeoPackage Encoding Standard was developed for the purpose of providing an open, standards-based, platform-independent, portable, self-describing, compact format for transferring geospatial information. GeoPackage has proven to be an effective "container" mechanism for bundling and sharing geospatial data for a variety of operational use cases. However, GeoPackage stakeholders have observed persistent interoperability issues, particularly with regards to metadata, extensions, and portrayal.

While OGC 19-047 discusses a series of enhancements, this chapter focuses on enhancements directly relevant for portrayal, including enhancements to deal with change sets that are part of the Testbed-15 Open Portrayal Framework [scenario](#).

8.2. GeoPackage Extensions and Profiles

[Section 6.1.2 of OGC 19-047](https://portal.opengeospatial.org/files/?artifact_id=89670&version=1#_styles_metadata) [https://portal.opengeospatial.org/files/?artifact_id=89670&version=1#_styles_metadata] introduces a concept for building metadata profiles. Metadata profiles are an optional agreement on how metadata is to be used in a GeoPackage to meet a particular purpose. Through the GeoPackage extension mechanism, a community of interest may define one or more metadata profiles that specify the encoding, scope, and purpose of a particular type of metadata. Without this mechanism, a GeoPackage client would be forced to inspect all metadata content to respect an out-of-band (non-machine-encoded) agreement to determine how metadata is being used in that file.

The [GeoPackage Style Metadata Profile](https://portal.opengeospatial.org/files/?artifact_id=89670&version=1#metadata_styles_extension) [https://portal.opengeospatial.org/files/?artifact_id=89670&version=1#metadata_styles_extension] satisfies the need for metadata for styles. The second profile introduces data governance aspects to GeoPackages. The [GeoPackage Delta Update Profile](https://portal.opengeospatial.org/files/?artifact_id=89670&version=1#metadata_updates_extension) [https://portal.opengeospatial.org/files/?artifact_id=89670&version=1#metadata_updates_extension] allows tracking local updates to a GeoPackage, so that they can be applied to another repository. GeoPackage is a single-user database with no built-in security, so any management of updates beyond version and checksum will rely on the trustworthiness of software and users. With that understanding, the GeoPackage Metadata Extension supports the elements needed to track updates made locally. Please note: The name Delta Update was used in early stages of Testbed-15 development and was only superseded by *Change Set* towards the end of the initiative.

An important GeoPackage mechanism produced during Testbed-15 was [Semantic Annotations](https://portal.opengeospatial.org/files/?artifact_id=89670&version=1#_semantic_annotations) [https://portal.opengeospatial.org/files/?artifact_id=89670&version=1#_semantic_annotations]. This activity addressed interoperability issues caused by lack of formal semantics in data descriptions. There is an unbounded set of ancillary information that may be operationally relevant to GeoPackage users.

Members of the GeoPackage community have periodically proposed adding additional columns to existing GeoPackage tables to address one-off operational needs. This approach does not scale, and adding additional columns to existing tables introduces interoperability risks and does not necessarily meet operational needs due to unclear semantics.

The proposed Semantic Annotations extension allows such information to be placed on any GeoPackage business objects. The schema for a Semantic Annotation is straight-forward, including resolvable URIs and types along with a human-readable name and description. Semantic Annotations are linked to business objects via the Related Tables Extension. [Section 6.3.1](#) [https://portal.opengeospatial.org/files/?artifact_id=89670&version=1#sa_layers_to_styles] explains how the mechanism can be applied to link layers to styles, which is essential to identify appropriate styling rules, symbols, and fonts for data layers. Since the styles that will work for a particular layer are independent of the data (and may be produced by one or more completely different organizations), a loose coupling between layers and styles is preferred. By annotating layers (either conventional or tiled vector data) with known valid styles, a GeoPackage client may provide a user a reasonable set of style options to choose from. Whether the styles are stored directly in the GeoPackage or are accessible through a separate style service, the client will be able to retrieve the styles and apply them to the data in the layer.

The [following figure](#) illustrates the Semantic Annotation concept. Based on GeoPackage's [Related Tables Extension](#) [<http://docs.opengeospatial.org/is/18-000/18-000.html>], the data content and its organization in a layer is linked to the stylesheets by leveraging the Semantic Annotation concept.

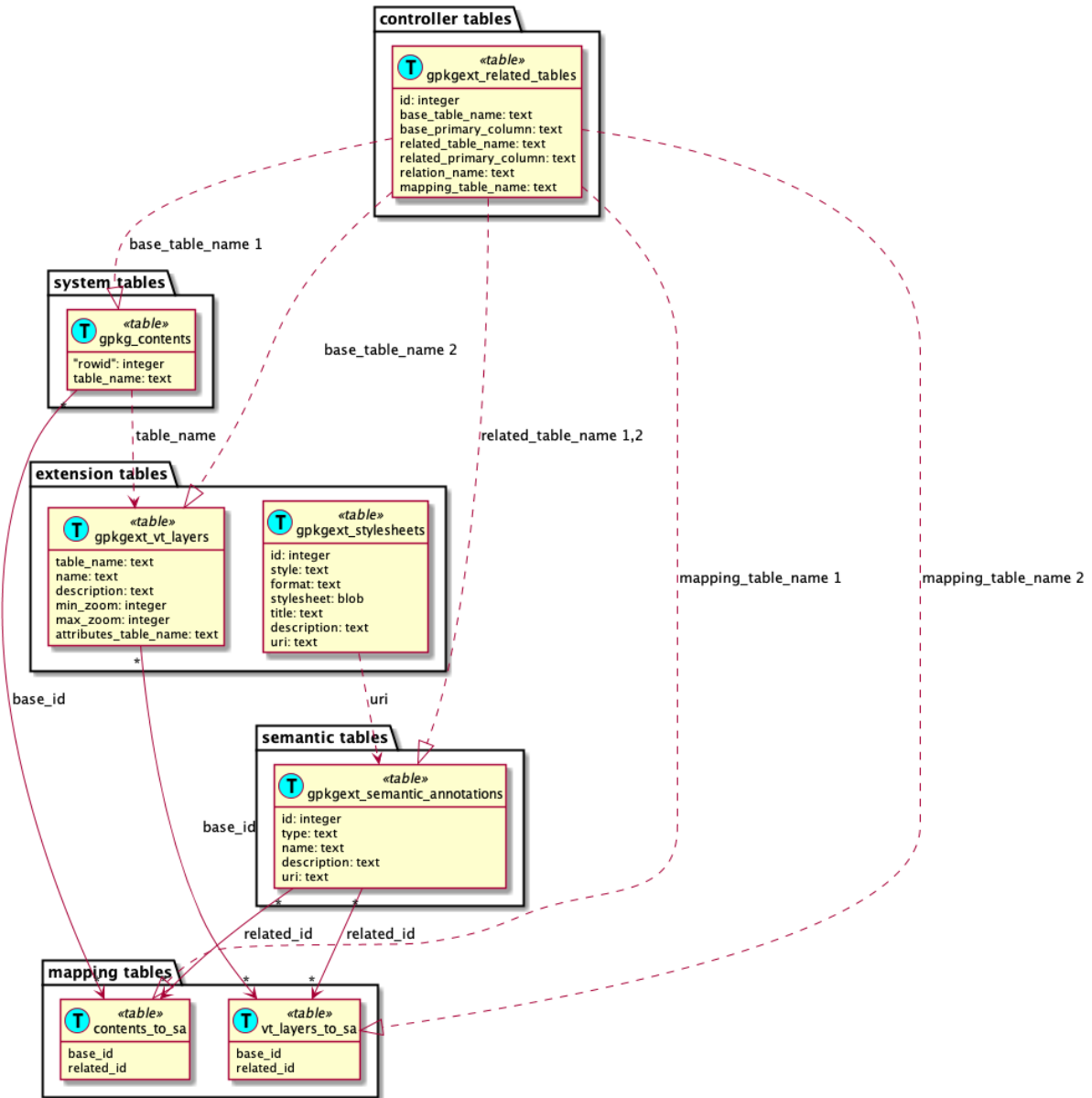


Figure 7. Styles linked to layers based on the GeoPackage related table and Semantic Annotation concepts

The same approach can be applied to *Stylable Layer Sets*. The *stylable layer set* concept was introduced to group multiple styles that apply to the same data set or schema. For example, a stylable layer set could apply to a theater of operations and could group a set of styles (e.g., "topographic", "satellite overlay", and "night"). The original proposal called for a column to be added to a pair of tables, but the semantic annotation approach would be more flexible and could be applied to both conventional and tiled vector layers. In this latter case, both the layers and the styles could be annotated with one or more stylable layer sets.

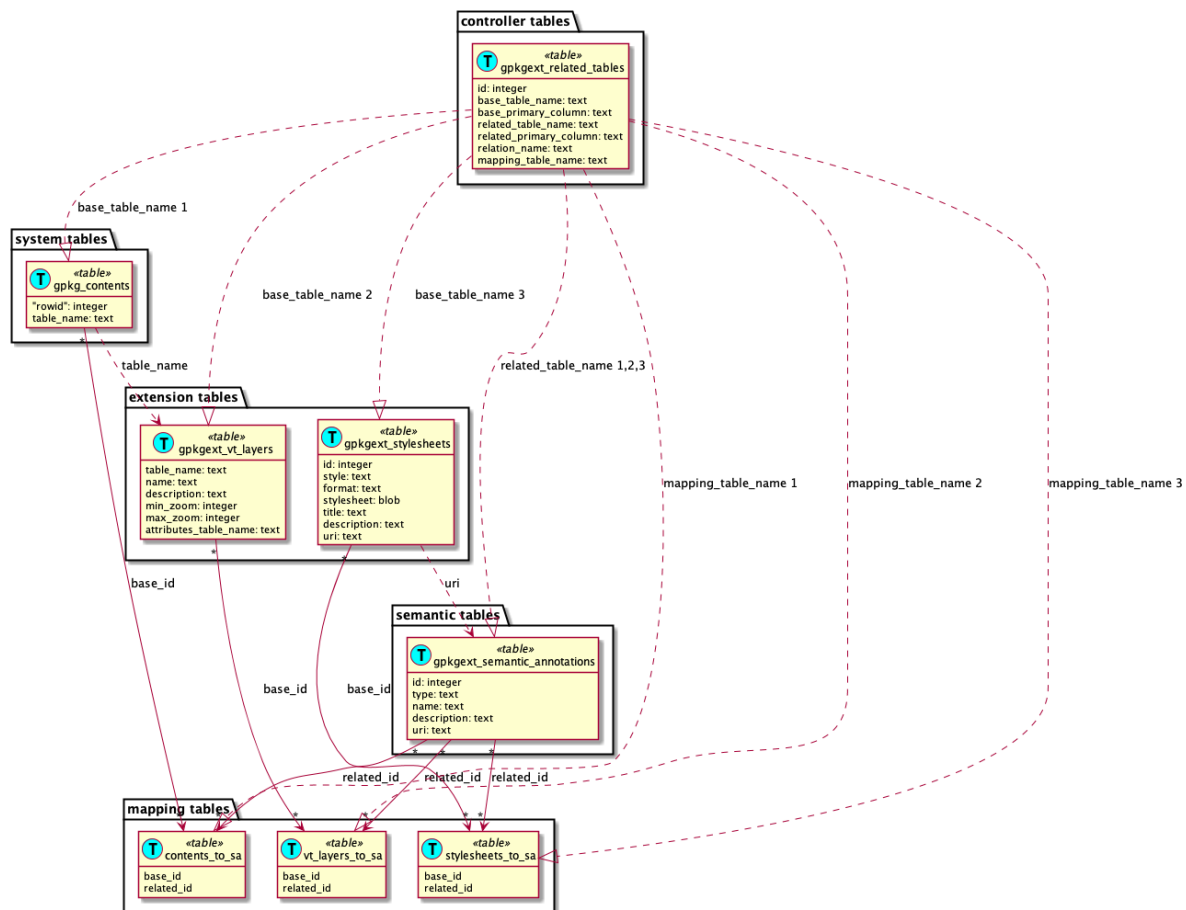


Figure 8. Styles linked to layers based on the GeoPackage related table and Semantic Annotation concepts

A third practical example for Semantic Annotation is [Layer-to-Layer Linking](https://portal.openeospatial.org/files/?artifact_id=89670&version=1#_example_3_layer_to_layer_linking) [https://portal.openeospatial.org/files/?artifact_id=89670&version=1#_example_3_layer_to_layer_linking]. When a GeoPackage contains a relatively large volume of vector data (either in conventional feature or tiled vector layers), a recommended practice is to pre-render raster tiles that cover a large geographic area. This reduces the need to render extreme numbers of features at run-time. In scenarios where a full OWS Context document is considered to be over-kill for the scenario, a simple semantic annotation can be used to link vector and raster layers that represent the same underlying data.

8.3. GeoPackage to support the Open Portrayal Framework

The longterm goal of the Open Portrayal Framework is to realize smooth online/offline experiences when dealing with high quality maps based on heterogeneous source data. Pre-rendered and raw vector- and raster data needs to be visualized using externally provided and governed styles. Updates in image archives causing re-generation of tile sets are made transferrable to tile caches using tile change-sets in both, online and offline scenarios. These requirements lead to the aforementioned extensions and Semantic Annotation concepts and have been implemented as documented below. In principle, three major challenges had to be addressed:

1. Conveying the styling rules
2. Coupling layers with styles

3. Making the styling rules accessible operationally

8.3.1. Conveying Styling Rules

Styling rules must be encapsulated in a common encoding that can be used by GeoPackage clients to render (draw) the features onto the screen in a prescribed way. Participants in the OGC Testbed-15 Open Portrayal Framework Thread asserted that it is not realistic for the community to select a single encoding for styling rules. There are too many diverse requirements, existing encodings are too entrenched, and introducing a completely new encoding would be too risky and expensive. That said, the GeoPackage community would benefit from a default encoding. An ideal encoding would be standardized, easy to implement and use, and capable of handling the gamut of basic portrayal needs. Unfortunately, no single encoding has all three of these traits. As part of the OPF thread in Testbed-15, participants used two encodings.

1. OGC's Styled Layer Descriptor (SLD) is an OGC standard and handles most basic portrayal requirements. However, the use of an XML-based structure is considered by many to be difficult to use. In addition, XML is a poor choice in mobile computing environments, where GeoPackage is commonly used.
2. Mapbox Styles is not standardized but it does handle most basic portrayal requirements and its JSON-based structure is considered relatively easy to use.

GeoPackage should support other encodings, either explicitly or through the use of extensions. This will allow organizations to innovate as technology evolves.

While allowing multiple style encodings does not maximize interoperability, this is the best solution available at this time. This solution considers the needs of both, the procurement community and the developers, for operating in a more agile environment. Each portrayal engine will have its own internal model for handling styles. The emergence of a symbology conceptual model (currently under development as OGC 18-067) will assist developers in mapping from one or more style encodings to their internal model.

8.3.2. Coupling Layers and Styles

Since styles and data layers are often developed independently, an effective spatial data infrastructure needs a way to couple layers with styling rules that are appropriate for those layers. This is particularly important in GeoPackage, where software capabilities tend to be more limited to support streamlined workflows that are easy to train for. The first requirement for effective style management is to provide a resolvable URI for each style. After that, there is some flexibility.

Independent map views, each with their own sets of layers and corresponding styles, can be encoded in the GeoPackage as OWS Context files using the proposed OWS Context Extension. For example, a topographic map may have no background (or possibly a single-colored base map as a background) and one or more feature layers, each with a specific style (referenced by its URI). A corresponding satellite overlay map may have tiled raster data as a background with the same set of feature layers, but with a different set of styles. The GeoPackage client can then provide a list of available contexts to the operator. This approach is the simplest for the operator, but does not provide the operator any way to customize the display.

Semantic Annotations can be used to couple layers with styles. In this case, it is the GeoPackage

client's responsibility to provide the appropriate list of styles to the operator. The operator can then select the desired style on a layer-by-layer basis.

Semantic Annotations can also be used to couple layers with stylable layer sets. In this case, the GeoPackage client's and operator's responsibilities are similar, but the organization of the styles is slightly different.

8.3.3. Making Styles Accessible Operationally

While the previous section describes how to couple layers and styles together, it does not address the challenge of getting the styles to the GeoPackage client, so that they can be used to visualize the data. Since there are a number of operational settings that a GeoPackage client may operate in, the infrastructure needs a number of ways to retrieve style information. Some of these ways include the following:

- In a connected environment, a GeoPackage client may retrieve a style directly via its URI. In a mature architecture, the service hosting the style would be able to provide the style in more than one encoding so that the service can meet the needs of disparate clients.
- In a disconnected environment, a GeoPackage client may retrieve a style directly from a GeoPackage via the proposed Styles Extension. When used in conjunction with manifests, a client would be able to discover that the Styles Extension is in use and what specific options (particularly encodings) are supported. The GeoPackage producer would then be responsible for ensuring that the styles are added in all of the necessary encodings and that the manifest fully reflects the GeoPackage's contents.
- In a mixed environment, a GeoPackage client may retrieve a style through a style service (i.e., registry) that can resolve a style via its URI. In this scenario, the service decouples the style from where it is stored. This provides flexibility beyond what would be appropriate for a GeoPackage Client to manage by itself.

8.4. Implementations and Lessons Learned

The following section provides an overview of the various participant implementations. The overview is complemented with important lessons learned during implementation and technical interoperability testing exercises. The following figure provides an overview of Testbed-15 Open Portrayal Framework GeoPackage components.

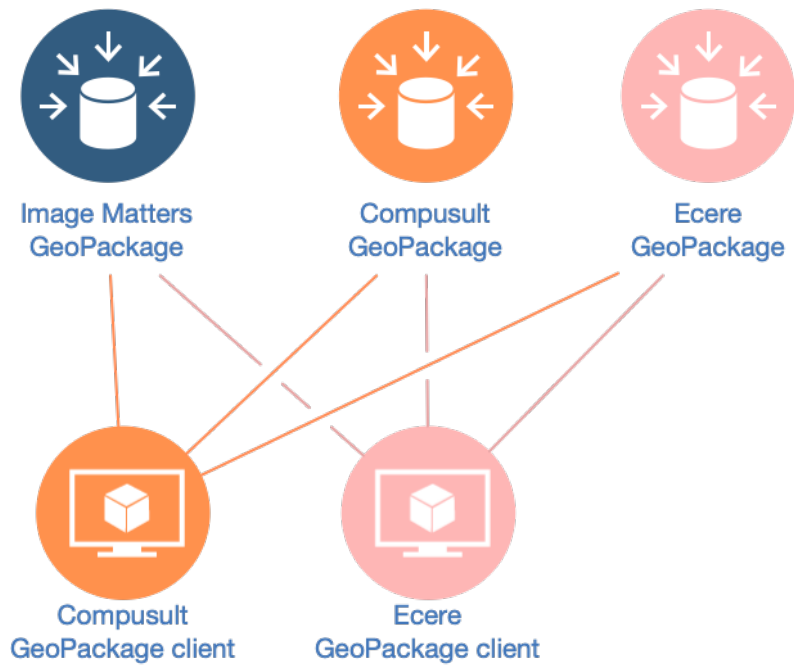


Figure 9. GeoPackage components developed Testbed-15 Open Portrayal Framework

Image Matters has developed a GeoPackage Provisioner that receives a JSON-formatted REST request and creates and populates a GeoPackage to meet that request. The Provisioner was originally written to support tile formats such as WMTS, but has been extended to support tiled vector data. The Provisioner supports other GeoPackage extensions in certain cases, but these are not currently handled in the case of vector tiles. Currently, the provisioner supports only the Coordinate Reference Systems (CRS) EPSG:3857 (Web Mercator) and EPSG:4326 (WGS84).

For example, a request could specify a CRS, a bounding box, a service, a layer, and a list of included zoom levels, and the provisioner would fetch all tiles with those specifications that fit within the bounding box, and put them into a GPKG. Optimistically, styling information is also gathered and included, according to the Style extension.

Chapter 9. Technical Discussions

The following sections present the relevant technical aspects that have been discussed in the *Open Portrayal Framework* thread during the execution of Testbed-15:

9.1. Background Color

The Testbed-15 scenario required the usage of specific background color: gray. Whereas the MapBox styles technology allows setting background color values, OGC SLD does not. Testbed-15 developed a proposal as to how a background-color parameter should go in an SLD document and how this approach should work. After some discussions, the following solution emerged:

The SLD `<UserStyle>` element should be augmented to include an optional `<BackgroundColor>` element, whose value is a hexadecimal RGB color value. Colors in SLD and SE are expressed in the `#RRGGBB` form. The default value of this element should be `"#FFFFFF"` (white). This element declares the **preferred** background color of the style. Change request [#620](http://ogc.standardstracker.org/show_request.cgi?id=620) [http://ogc.standardstracker.org/show_request.cgi?id=620] was submitted against the Styled Layer Descriptor Profile of the Web Map Service Implementation Specification version 1.1.0 (OGC 05-078r4) to add this element.

Further discussion focused on whether the `BackgroundColor` could also be sensibly placed inside *StyledLayerDescriptor* for styles that have a list of `NamedLayer/UserStyle` (which is possible, and GeoServer's preferred way of handling multi-layer styles, even outside the confines of tiled vector data). At the end, the idea was discarded because the preferred background color is as style-specific as the foreground colors and line thicknesses, etc.

If a tile is requested in a format that does not support transparency (such as JPEG), or if the request is made with `TRANSPARENT=FALSE`, the server should return the tile with a solid background of the color specified by the `<BackgroundColor>` element of the style. Change request [#621](http://ogc.standardstracker.org/show_request.cgi?id=621) [http://ogc.standardstracker.org/show_request.cgi?id=621] was submitted against the Web Map Tile Service Implementation Standard version 1.0.0 (OGC 07-057r7) to add this optional `TRANSPARENT` parameter.

If tiles are cached in an opaque format such as JPEG, clients cannot expect that the server changes the background color on the fly. A server could still conceivably cache PNG and return JPEG, but not all users will want to do this. Still, any server should not be expected to handle the "transparent" parameter (or any query parameter for that matter). A client application can always add parameters to a request, if the client desires the behaviors the parameters are intended to invoke. Services that don't support or recognize these additional parameters will simply ignore them.

When a client application that is aware of this new mechanism is displaying a map and the bottom-most layer is a WMTS-tile layer, the client should request the tiles of that layer with `TRANSPARENT=FALSE` (or as a JPEG). This allows the map to be displayed with the background color that is appropriate for the selected style, without the client application having to know anything about the style (other than its ID). This can be done even for WMTS servers that do not declare this profile. This is because earlier WMTS servers will simply ignore the `TRANSPARENT` parameter and return transparent tiles, which will be displayed on the (presumably white) background of the client application, as it is currently done.

Even if the tile is returned with a transparent background (which is the normal case), the server should - if possible - endeavor to have the background be a fully transparent version of the color specified by the <Background> element of the style. This allows client applications to infer this information out of its cached tiles, if the client decides at a later time to provide an appropriate background color for them.

When a client application that is aware of this proposed mechanism is requesting tiled data and rendering the map, it will presumably have retrieved the style definition by making a GetStyle(s) request. From this, the client will be able to determine the correct background color by looking up the style's <Background> element in the SLD. Then, the client can simply set the canvas to that background color.

This mechanism alleviates the need for a WMTS server to accept an arbitrary background color in a GetTile request, increasing cache use. The WMTS server could still potentially pre-render all tiles: pre-render a transparent and an opaque version of each.

As an example, the "Night" style in a previous initiative required a blue background. This information was communicated by the MapBox style definitions, but not the SLD style definitions. As a result, any SLD-based client application showing this style

- either rendered a white background,
- implemented coding specific knowledge of that background color into the client, or
- inserted a large blue polygon layer to serve as the background color.

However, with the extension suggested above, the SLD encoding of the "Night" style could include this information as shown in the following example. This example assumes that a separate WMTS Layer is provided for each feature set:

SLD with `<BackgroundColor>` element, multiple layers

```
<sld:StyledLayerDescriptor version="1.1.0" [...]>
  <sld:NamedLayer>
    <Name>AgricultureSrf</Name>
    <sld:UserStyle>
      <Name>Night</Name>
      <BackgroundColor>0x1E193E</BackgroundColor>
      <FeatureTypeStyle>
        [...]
      </FeatureTypeStyle>
    </sld:UserStyle>
  </sld:NamedLayer>
  <sld:NamedLayer>
    <Name>CulturePnt</Name>
    <sld:UserStyle>
      <Name>Night</Name>
      <BackgroundColor>0x1E193E</BackgroundColor>
      <FeatureTypeStyle>
        [...]
      </FeatureTypeStyle>
    </sld:UserStyle>
  </sld:NamedLayer>
  [...]
</sld:StyledLayerDescriptor>
```

or like this (if a single WMTS Layer contains all feature sets):

SLD with `<BackgroundColor>` element, single layer with all feature sets

```
<sld:StyledLayerDescriptor version="1.1.0" [...]>
  <sld:NamedLayer>
    <Name>Daraa</Name>
    <sld:UserStyle>
      <Name>Night</Name>
      <BackgroundColor>0x1E193E</BackgroundColor>
      <FeatureTypeStyle>
        <FeatureTypeName>AgricultureSrf</FeatureTypeName>
        [...]
      </FeatureTypeStyle>
      <FeatureTypeStyle>
        <FeatureTypeName>CulturePnt</FeatureTypeName>
        [...]
      </FeatureTypeStyle>
    </sld:UserStyle>
  </sld:NamedLayer>
</sld:StyledLayerDescriptor>
```

The following demo provided by CubeWerx demonstrates the concepts described above. Note that

some links may not work any longer at the time of reading, but the URL structure and corresponding figures should still provide valuable information.

Look at the SLD at [this link](https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/layers/Daraa_vectors) [https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/layers/Daraa_vectors] or in [this annex](#).

Notice that the "Topographic" style has a `<sld:BackgroundColor>#CCCCCC</sld:BackgroundColor>` element (look, gray!) and the "Night" style has a `<sld:BackgroundColor>#1E193E</sld:BackgroundColor>` element. This specifies the preferred background color for these styles. The "Overlay" style does not have a BackgroundColor element, so the preferred background color defaults to white.

Make a tile request to https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/tiles/Daraa_vectors/Topographic/smerc/14/6621/9838.png. Notice that the standard tile still has a transparent background.



Figure 10. Tile with a transparent background

If one executes the following endpoint https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/tiles/Daraa_vectors/Topographic/smerc/14/6621/9838.png?transparent=false. the client will display an opaque tile, where the background color is not the default white, but the color specified by the BackgroundColor element in the style! If a client application always requests the bottommost layer with transparent=false, the client will automatically get the background color that is the most appropriate for that layer.



Figure 11. Tile with a background as defined by the style

The application can also accomplish this same capability by specifying a format that does not support transparency. Try https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/tiles/Daraa_vectors/Topographic/smerc/14/6621/9838.jpg.



Figure 12. JPEG tile

As an experimental extension (not proposed as part of the API Tile specification), the Cubewerx tile requests also accept a `BackgroundColor` parameter that can be used to specify an arbitrary background color. Executing the following endpoint service https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/tiles/Daraa_vectors/Topographic/smerc/14/6621/9838.jpg?BackgroundColor=red allows various color syntaxes to be accepted.



Figure 13. Tile with red background

This mechanism is also supported by our WMS. By default you will get the background color specified by the style, rather than white. Try https://tb15.cubewerx.com/cubewerx/cubeserv/op?SERVICE=WMS&VERSION=1.3.0&REQUEST=GetMap&FORMAT=image%2Fjpegorpng&LAYERS=Daraa_vectors&STYLES=Topographic&CRS=EPSG%3A3857&WIDTH=640&HEIGHT=325&BBOX=4024937,3840638,4028353,3842372

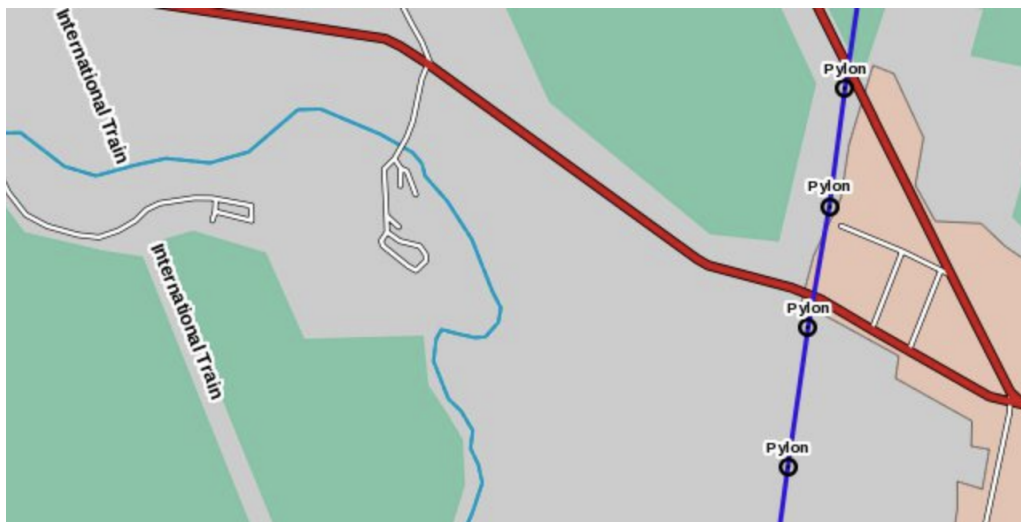


Figure 14. WMS response with background as defined by style

Cubewerx created a short demonstration, available at <https://tb15.cubewerx.com/cubewerx/demos/BackgroundColorDemo.html> to illustrate the final effect. In addition to the gray background of the Topographic style, check out the blue background of the Night style! Also note that when the "Daraa Mosaic 2019" imagery is selected, the imagery becomes the background. The following figures show the client in action:

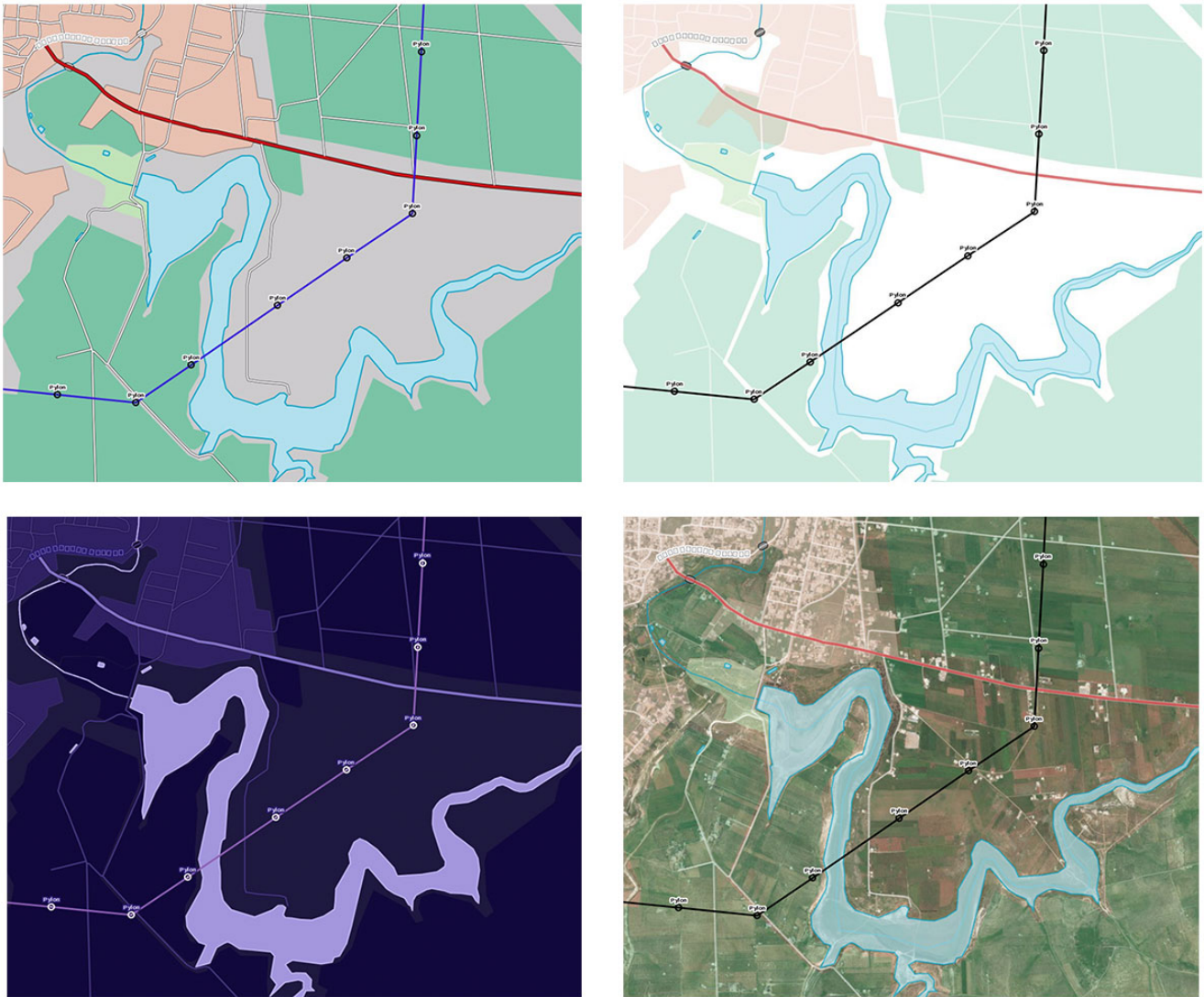


Figure 15. Different map representations. Upper left: gray background, upper right: overlay with white background; lower left: night style; lower right: overlay with satellite image tiles as background

9.2. Styleable Layer Set

The *Styleable Layer Sets* concept was introduced to group multiple styles that apply to the same data set or schema. For example, a styleable layer set could apply to a theater of operations and could group a set of styles (e.g., "topographic", "satellite overlay", and "night"). The concept was originally developed as part of the Vector Tiles Pilot and is described in detail in the [OGC Engineering Report 18-101](http://docs.openeospatial.org/per/18-101.html) [http://docs.openeospatial.org/per/18-101.html] with [the concept's definition](http://docs.openeospatial.org/per/18-101.html#terms_clause) [http://docs.openeospatial.org/per/18-101.html#terms_clause] and a [corresponding UML diagram](http://docs.openeospatial.org/per/18-101.html#img_Feature_Tile_Conceptual_Model) [http://docs.openeospatial.org/per/18-101.html#img_Feature_Tile_Conceptual_Model].

Styleable layer sets allow defining bi-directional data layer to style and style to data layer associations. In this context, please note that a styleable layer set is necessarily constrained to vector data.

GeoPackage has support for styleable layer sets as well. This capability is described in section [GeoPackage: Coupling Styles and Layers](#). The recommended approach uses the semantic annotation model. Here, the original proposal called for a column to be added to a pair of tables. However, the semantic annotation approach would be more flexible and could be applied to both conventional

and tiled vector layers. In this case, both the layers and the styles could be annotated with one or more styleable layer sets.

9.3. Conversions Between style Encodings

9.3.1. GeoSolutions

The Visual Style Editor component implementation allowed experimenting with some rendering and style conversion JavaScript libraries. Tests focused mainly on vector layers representing the urban area of Daraa, including points of interest, polygon surfaces and lines connections. The implementation focused on simple styles to get a good match between parameters of different encodings, in particular SLD and Mapbox Style. The following symbolizers and properties were considered in the GeoSolutions implementation:

- Point
 - Image source.
 - Scale/size of marker.
- Line
 - Stroke color.
 - Stroke width.
 - Stroke opacity.
- Polygon
 - Outline color.
 - Outline width.
 - Fill color.
 - Fill opacity.

GeoStyler, a JavaScript library for encoding and translation, was integrated in MapStore, and results were displayed in both an OpenLayers and a MapBox Client. [Figure 16](#)

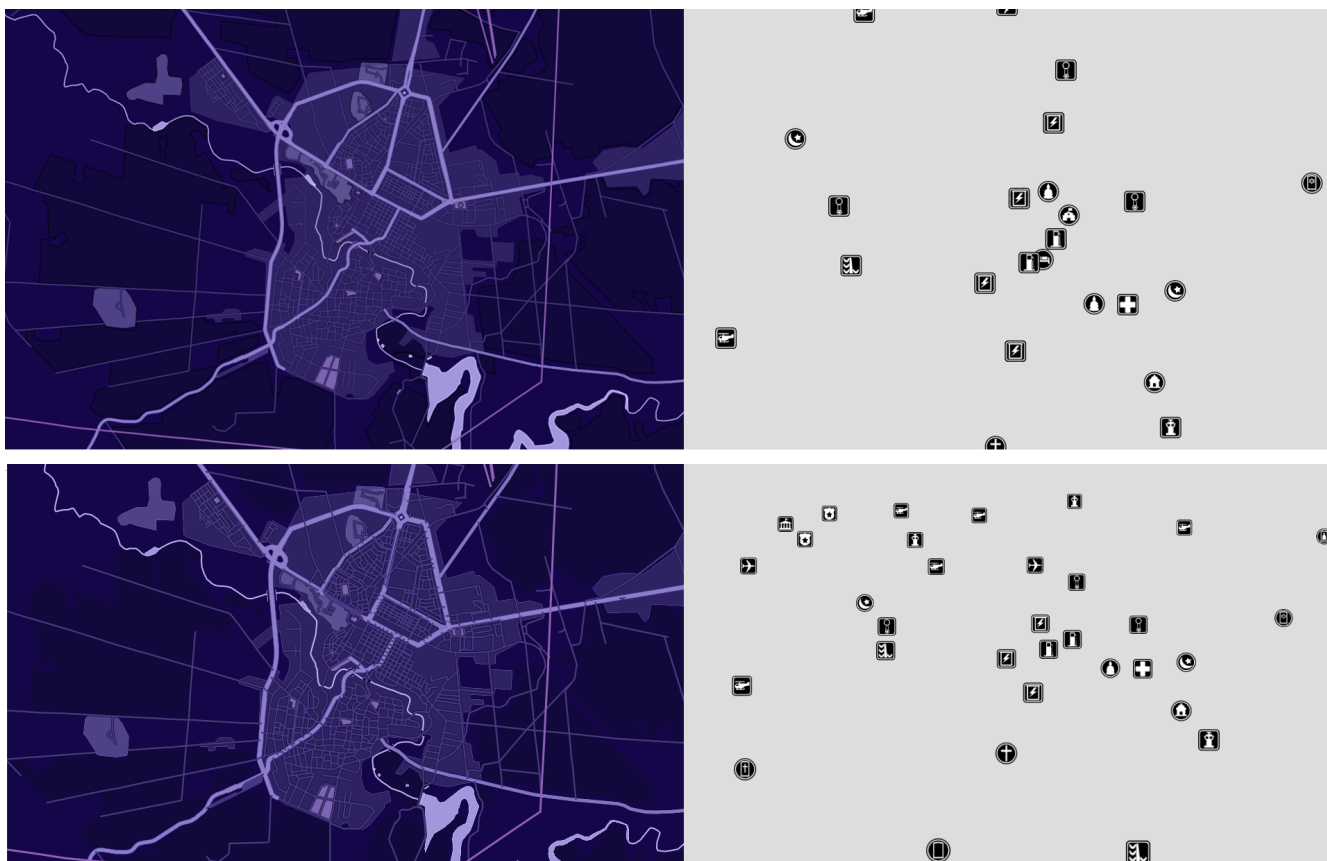


Figure 16. MapStore Client Demo - testing conversion and rendering of styles on the client side. (Top: OpenLayers Client, bottom: Mapbox GL Client)

It is also worth considering the GeoServer server side conversion. GeoServer rendering engine is based on a style object model mapping one to one the SLD 1.0 structures, with a few extensions. A user is however allowed to enter a style description in multiple encodings, including SLD 1.0, SE 1.1, GeoCSS, Mapbox Styles, YSLD. Each style encoding support includes:

- A parser for the encoding, able to parse the whole syntax, for most encodings, or just partially, for example, for Mapbox GL styles.
- A translator from the encoding structure to the GeoServer style object model, which always allows a complete encoding in SLD 1.0 (with extensions).
- An optional encoder, from the GeoServer style object model, back to the encoding. Currently only YSLD supports this functionality.

Given this structure, the ability to translate between languages is often offered in a single direction, towards GeoServer extended SLD 1.0, but normally quite complete, and limited only by the plugin ability to parse the source structure. It is however important to notice that most of the encodings discussed above have the same expressive power as SLD 1.0, and differ only by syntax and rule mechanics, both of which can be bridged down to SLD with some effort.

Outside of the Testbed-15 GeoSolutions also had experiences with QGIS style translation to SLD. In that case, there is a wide variety of styling ability, symbolizers in particular, that simply cannot be translated to standard SLD. Some examples follow:

- Layer compositing (alpha masking and color blending).
- Z-ordering (within layer, or cross layer, e.g., between roads, rails and lakes).

- Gradient fills.
- [Shapeburst fills](https://nyalldawson.net/2014/06/shapeburst-fill-styles-in-qgis-2-4/) [https://nyalldawson.net/2014/06/shapeburst-fill-styles-in-qgis-2-4/].
- Inverted polygon rendering (coloring the outside of the polygon, instead of the inside).
- Advanced raster color stretching, such as cutting values outside of a given range, and/or stretching the colors to a given range.

Summarizing, style conversion limitations could be due to:

- Incomplete parsing of the source.
- Incomplete translation between the styling models.
- Lack of concepts in the target encoding, or mismatch of the same.
- Different mechanics requiring actions outside of the mere style (e.g. translation of source scalable SVG resources, or marks, to a sprite with the exact images required, feature by feature).
- Incomplete encoding of the target.

9.3.2. Ecere

Ecere implemented styles conversion from scratch between SLD/SE, Mapbox GL and GNOSIS Cascading Map Style Sheets (CMSS).

The functionality was developed as importing and exporting capabilities for the SLD/SE and Mapbox GL styles encodings to and from GNOSIS's native CMSS. The conversion process can also convert between SLD/SE and Mapbox GL through CMSS acting as a bridge.

This functionality was used and demonstrated in three different ways:

- using a simple command-line tool;
- on the Ecere's Styles API service where a style can be supplied in one or two formats and automatically be made available in the missing formats;
- as well as within GNOSIS Cartographer's Visual Style Editor, where styles can be imported and exported, either simply from local files, or from styles accessed from GNOSIS data stores, GeoPackages or OGC API - Styles.

A major distinction of CMSS is that the order of rules defines the order of overrides, rather than the drawing order of layers.

These rules overrides allow for easily specializing symbology through nested rules. The rules overrides also allow for defining complex multi-dimensional styling where flattening the rules would result in a very large number of negative rules. These negative rules do not have to be expressed with support for overrides (e.g. see styling Thermokast map from OGC Arctic Spatial Data Platform initiative). To convert CMSS to SLD/SE or MapboxGL, the rules must be flattened and these negative filters must be generated (except for the top-level ElseFilter in SLD/SE). To import the styles back to CMSS, the parser must attempt to deduct the nesting and overrides from the selectors in an attempt to simplify the styles.

With CMSS, all enabled layers are rendered by default with default styles (e.g. black strokes and

white fill), whereas in MapboxGL and SLD/SE a layer is not rendered, unless explicitly specified in the style.

In CMSS, ordering is specified explicitly using a `zOrder` property, which allows for example to change the z-ordering property within the layer or across different layers.

Another important distinction is how CMSS explicitly defines lines casing as a property of the stroke, rather than defining a separate rule for rendering a separate line on top. This allows, for example, to render the line in a single pass, and allows defining a casing width in pixels while using real-world units such as meters for the thickness of the stroke. For 3D views, this in turn enables smooth strokes across different zoom level transitions. Importing the stroke and casing widths from the other encodings was a challenge, especially as different styles definitions sometimes use different mechanism. The parser attempted to identify multiple line symbolizers as stroke and casing. MapboxGL offers the 'step' function which was used to encode real-world widths based on zoom level. SLD/SE supports specifying of units of measure, but does not support specifying a casing width in pixels while the main stroke is defined in meters.



Figure 17. Example of rendering artifacts in 3D with fixed pixel stroke and casing widths

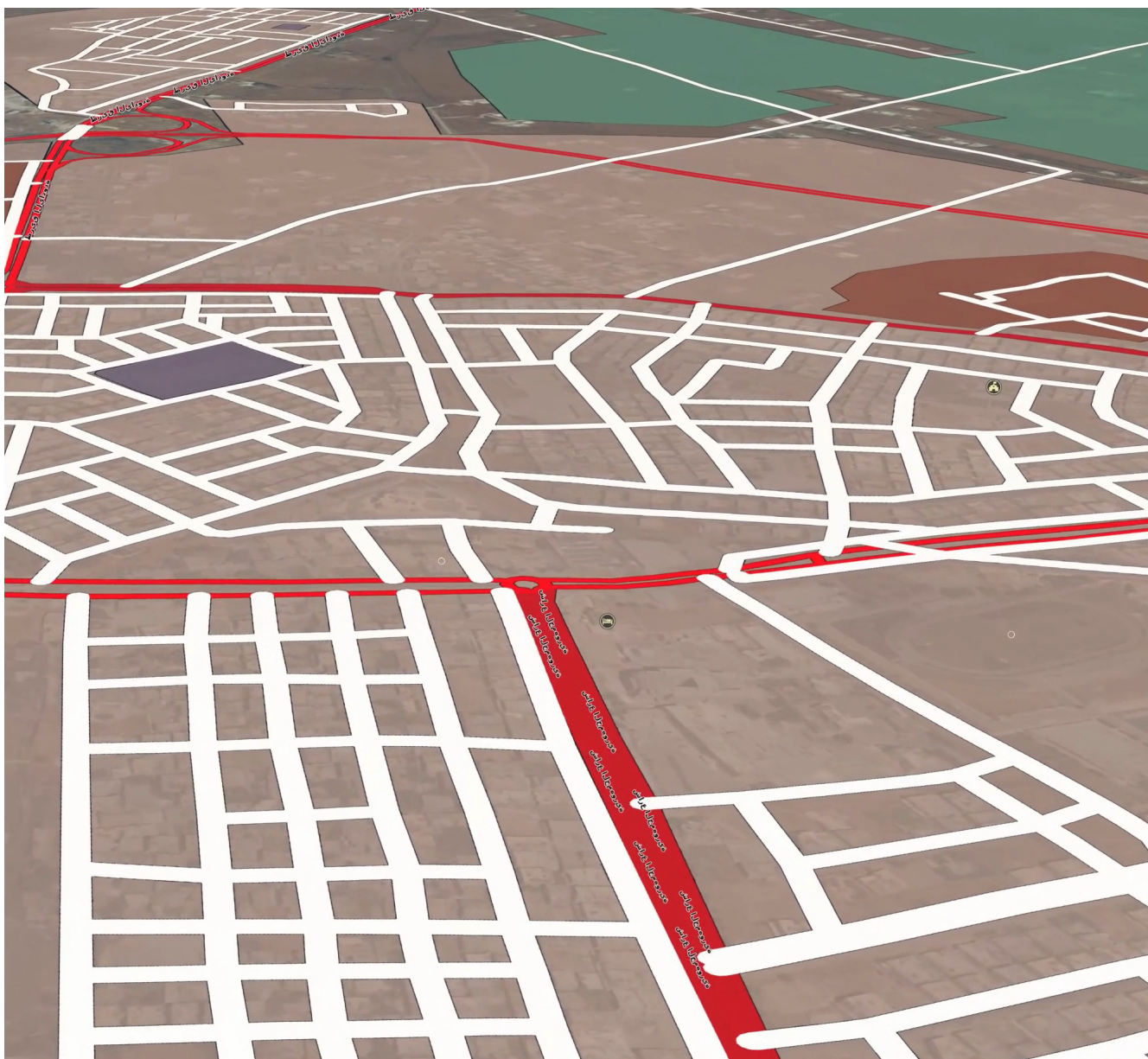


Figure 18. **Smooth lines across zoom levels with real-world (meters) stroke width and pixel casing width**

Image scaling is also a tricky conversion issue, as some encodings define the size as a scale factor relative to the original size of the image, while others define it as an absolute number of pixels. This complicates the process, as the symbol image must be accessed to query its size in order to perform the conversion.

Some style encodings define text size in pixels, while others use typography points.

Support for styling coverages was also added to CMSS during this initiative, with conversion support to SLD/SE and Mapbox GL as well.

9.4. Sprites

A *sprite* is a computer graphics term for a two-dimensional bitmap that is integrated into a larger scene. In the context of the Open Portrayal Framework, sprites can be used to store various symbols in a single bitmap (i.e. single file), instead of a set of SVG or PNG symbols.

Sprites are used in the Mapbox technology, which does not allow standalone symbols such as sets of SVGs or PNGs. There was general consensus that this constraint should be lifted in future to allow for more flexibility. Being limited to sprites alone was perceived as cumbersome.

At the moment, GeoPackage does not support sprites, but instead requires individual symbols. The OGC SLD/SE standard supports sprites.

Another element that needs to be further explored in the future is governance of sprite sheets. In the context of the Open Portrayal Framework, where dedicated style servers allow users to manage and share styles, sprites are hard to manage from a governance perspective.

Another issue is that large sprite sheets may cause transfer issues in Denied, Degraded, Intermittent and Limited (DDIL) bandwidth environments.

9.5. Mediatypes for an Open Portrayal Framework

The Web APIs used for the Open Portrayal Framework often make reference to specific media types. Though most commonly used media types such as `application/json` or `text/html` are already registered with IANA, several others are not. Section 9 of the Styles API Engineering Report discusses this topic in more detail. Basically, no media types have been formally registered with IANA for the style encodings (Mapbox Styles and OGC SLD). Temporary media types in the vendor tree "vnd" are used for now; i.e. `application/vnd.mapbox.style+json` for Mapbox Styles encoded in JSON, and `application/vnd.ogc.sld+xml` for OGC SLD styles encoded in XML.

An additional issue that was identified late in the discussion is the continued evolution of the Mapbox Styles specification. The Mapbox Styles specification is continuously evolving. The evolution is based on the technical and business need of the company and not necessarily aligned with the requirement for defining standards and future interoperable solutions. The current Mapbox Styles specification is version 8. One cannot easily find the previous versions from the site: <https://github.com/mapbox/mapbox-gl-js/tree/master/src/style-spec>. On the other hand, specifications such as HTML change based on community needs, while at the same time the media type `text/html` remaining unmodified. In contrast to HTML, though, the evolution of Mapbox is not fully comprehensible. The aforementioned Mapbox website has a note about using syntax that is only supported starting with a specific version of Mapbox GL JS 0.41.0, indicating what seems to be a strong coupling between drawing engine implementation and the specification: <https://docs.mapbox.com/help/tutorials/show-changes-over-time/>

As far as Testbed participants can judge, the Mapbox Styles specification is contained in a JSON document, which contains a schema of sorts: <https://github.com/mapbox/mapbox-gl-js/blob/master/src/style-spec/reference/v8.json> and this "v8" is actually being actively modified, see the history of changes: <https://github.com/mapbox/mapbox-gl-js/commits/master/src/style-spec/reference/v8.json>

Even what is called version "8" is not a fixed structure, but one that changes continuously, and gets released along with the JavaScript library implementing it with an aggressive schedule (one release every few weeks it seems), see here: <https://github.com/mapbox/mapbox-gl-js/releases>

In the case of SLD, there are two separate content types for SLD 1.0 and SLD/SE 1.1. The question arose if the Mapbox GL media type should be versioned? In other words, how does the client know what it will get from the media type alone? And how does a service or client identify a stable

version, if the only documentation link points to the current version of the specification?

Eventually, the participants decided to leave the definition and registration of a media type to the OGC Standards Program. For the sake of this testbed, implementations can use an "in-house" (i.e. fake) media type. Future discussions should take the registration of "foreign" media types in IANA and the vnd-branch into account, as discussed above.

Chapter 10. Further Information and Videos

More details on the overall results of the OGC Testbed-15 are available on the [OGC IP Initiatives - Testbed 15](https://www.opengeospatial.org/projects/initiatives/testbed15) [https://www.opengeospatial.org/projects/initiatives/testbed15] website.

Demo videos on the OPF scenario and various implementations are available at the [OGC YouTube Channel](https://www.youtube.com/playlist?list=PLQsQNjNIDU85HBDZWc8aE7EvQKE5nIedK) [https://www.youtube.com/playlist?list=PLQsQNjNIDU85HBDZWc8aE7EvQKE5nIedK].

Appendix A: Annex A: OPF Implementations

This section provides the detailed descriptions of implementation aspects and presents the results that have been achieved by the participants:

A.1. Implementation GeoSolutions

A.1.1. Available services and layers

GeoSolutions set up a server for Testbed-15 at <https://http://ows.geo-solutions.it/geoserver>. This server implements OGC APIs as well as classic OGC web services, as listed below:

Table 1. OGC web services provided by GeoSolutions' GeoServer

service type	service versions	URL of capabilities document
WMTS	1.0.0	http://ows.geo-solutions.it/geoserver/gwc/service/wmts?REQUEST=GetCapabilities
WMS	1.0.0, 1.1.0, 1.1.1, 1.3.0	http://ows.geo-solutions.it/geoserver/ows?service=wms&request=GetCapabilities
WCS	1.0.0, 1.1.1, 2.0.1	http://ows.geo-solutions.it/geoserver/ows?service=WCS&request=GetCapabilities
WFS	1.0.0, 1.1.0, 2.0.0	http://ows.geo-solutions.it/geoserver/ows?service=wfs&request=GetCapabilities
Features API	1.0.0	http://ows.geo-solutions.it/geoserver/ogc/features
Tiles API	1.0.0	http://ows.geo-solutions.it/geoserver/ogc/tiles
Styles API	1.0.0	http://ows.geo-solutions.it/geoserver/ogc/styles
Images API	1.0.0	http://ows.geo-solutions.it/geoserver/ogc/images

Layers served by GeoServer

The services expose over 150 layers, including the following groups of layers significant to the Testbed. The layers are divided into three workspaces:

- **vtp** covering the Daraa local area, providing a list of vector layers for various themes (e.g. **AeronauticPnt**, **AgriculturePnt**, **AgricultureSrf**), as well as the **Daraa_DTED** digital elevation model layer and the **imagery** layer with satellite imagery of the area.
- **syria_vtp** covering Syria, providing a list of vector layers for various themes (e.g. **building_s**, **built_up_area_s**, **cemetery_s**, **crop_land_s**)
- **iraq_vtp** covering Iraq, providing a longer list of vector layers over medium area (e.g.

`aircraft_hangar_s`, `amusement_park_s`, `annotated_location_s`, `apron_s`, `archeological_site_s`, `barn_s`, `bridge_c`)

Given the volume of the provided layers, the service collection listing can be overwhelming. In light of this, a filtered view of the layers can be provided. This can be done by adding the workspace name right after `geoserver` in the URL.

For example, the following URL, returns the "vtp" layers, the ones in the Daraa vicinity, while also losing the `workspace` prefix at the beginning of the collection identifiers:

<http://ows.geo-solutions.it/geoserver/vtp/ogc/features>

In addition to uniform content collections, a few grouped collections are available that combine multiple layers into a single container, including:

- `daraa_vtp`, including all Daraa layers
- `syria_vtp`, including all Syria layers
- `iraq_vtp`, including all Iraq layers

These collections can be used to retrieve multi-layer tiled data (i.e. data from more than one collection) from the draft Tiles API specification, as well as tiled maps to build server side rendered base maps.

In terms of supported Coordinate Reference Systems (CRS), all the layers are served in WGS84 (EPSG:4326) and Web Mercator (EPSG:3857).

Finally, in terms of formats, all vector layers are provided as PNG (8 bit), GeoJSON and Mapbox vector tiles, while raster layers are provided as PNG, JPEG and "JPEG or PNG".

NOTE

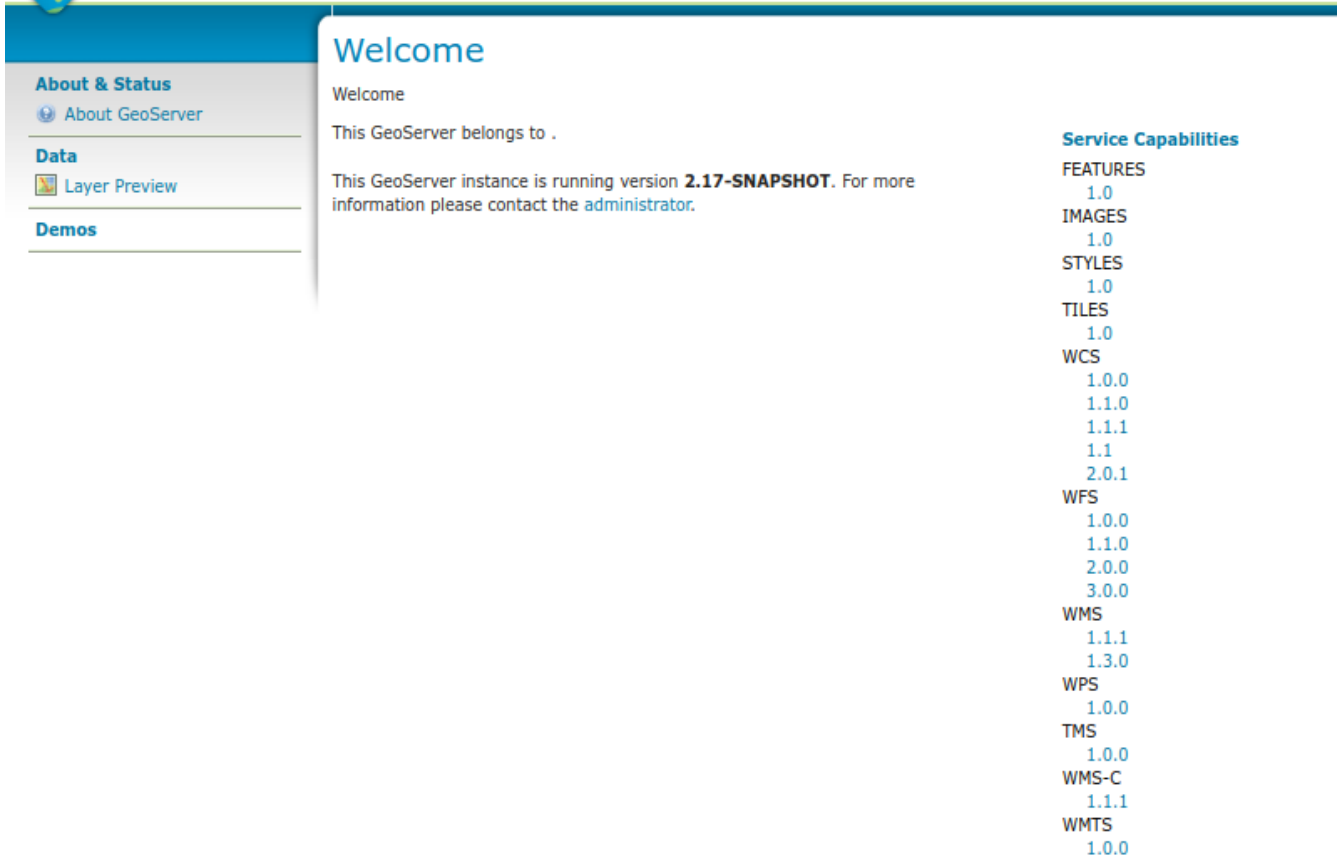
"JPEG or PNG" is identified via the media type `image/vnd.jpeg-png`, allowing the server to choose the format on the fly based on whether transparent pixels are part of the response, or not. The client should be prepared to handle both formats as a result.

A.1.2. OGC API implementation approach

When implementing more than one OGC API, two extreme approaches can be considered:

- Implement all OGC APIs under a single resource tree, providing a single list of collections that link, depending on their nature, items, images, tiles, coverages
- Implement each API under a separate resource tree

The Testbed-15 GeoServer implementation follows the second approach. However, APIs are cross linked with each other, allowing a client to discover style resources offered by the Styles API from both the Features and Tiles API. Vice versa, the client can find suitable data for a given style.



Welcome

Welcome

This GeoServer belongs to .

This GeoServer instance is running version **2.17-SNAPSHOT**. For more information please contact the [administrator](#).

Service Capabilities

- FEATURES
1.0
- IMAGES
1.0
- STYLES
1.0
- TILES
1.0
- WCS
1.0.0
1.1.0
1.1.1
1.1
2.0.1
- WFS
1.0.0
1.1.0
2.0.0
3.0.0
- WMS
1.1.1
1.3.0
- WPS
1.0.0
- TMS
1.0.0
- WMS-C
1.1.1
- WMTS
1.0.0

Figure 19. Capabilities documents and landing page links, from the GeoServer home page

A.1.3. Features API endpoint

The Features API endpoint provides an OGC API-Features interface. The endpoint was verified compliant with Features API during the Testbed 14 compliance activity.



GeoServer Features 1.0 Service

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

API definition

The [API document](#) provides a machine processable description of this service API conformant to OpenAPI 3. This API document is also available as [application/x-yaml](#), [application/cbor](#), [text/html](#).

Collections

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

Figure 20. GeoServer Features API landing page, as HTML

The Testbed-15 implementation was completely rewritten to use the new OGC API specific code

infrastructure, shared among all the new OGC OpenAPI based services. The implementation also includes references to the styles at the collection level, so that the Features can be styled on the client side:

An example of an OGC API-Features collection representation, with style links

```
{
  "id": "AeronauticPnt",
  "title": "AeronauticPnt",
  ...
  "links": [
    {
      "href": "http://ows.geo-
solutions.it/geoserver/vtp/ogc/features/collections/AeronauticPnt/items?f=text%2Fhtml"
    },
    {
      "rel": "item",
      "type": "text/html",
      "title": "AeronauticPnt items as text/html"
    },
    ...
  ],
  "styles": [
    {
      "id": "point",
      "title": "A boring default style",
      "links": [
        {
          "href": "http://ows.geo-
solutions.it/geoserver/vtp/ogc/styles/styles/point?f=application%2Fvnd.ogc.sld%2Bxml",
          "rel": "stylesheet",
          "type": "application/vnd.ogc.sld+xml"
        },
        {
          "href": "http://ows.geo-
solutions.it/geoserver/vtp/ogc/styles/styles/point/metadata?f=application%2Fx-yaml",
          "rel": "describedBy",
          "type": "application/x-yaml",
          "title": "The style metadata"
        },
        {
          "href": "http://ows.geo-
solutions.it/geoserver/vtp/ogc/styles/styles/point/metadata?f=application%2Fjson",
          "rel": "describedBy",
          "type": "application/json",
          "title": "The style metadata"
        },
        {
          "href": "http://ows.geo-
solutions.it/geoserver/vtp/ogc/styles/styles/point/metadata?f=application%2Fcbor",
          "rel": "describedBy",
          "type": "application/cbor",
        }
      ]
    }
  ]
}
```

```

        "title": "The style metadata"
    },
    {
        "href": "http://ows.geo-
solutions.it/geoserver/vtp/ogc/styles/styles/point/metadata?f=text%2Fhtml",
        "rel": "describedBy",
        "type": "text/html",
        "title": "The style metadata"
    }
]
},
...
]
}

```

A.1.4. Images API endpoints

The Images API allows the management of a collection of satellite images, offering a landing page at <http://ows.geo-solutions.it/geoserver/ogc/images>.



Image mosaicks discovery and management interface

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

API definition

The [API document](#) provides a machine processable description of this service API conformant to OpenAPI 3. This API document is also available as [application/x-yaml](#), [application/cbor](#), [text/html](#).

Image Collections

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

Figure 21. GeoServer Images API landing page, as HTML

Each collection is described by a STAC ([Spatio Temporal Asset Catalog](https://github.com/radiantearth/stac-spec) [https://github.com/radiantearth/stac-spec]) collection metadata document, and each image inside of the collection is described as a STAC item. STAC is discussed in more detail in the section [API Images](#). The GeoServer implementation includes the "Manage Images" conformance class allowing for upload and removal of images.

Table 2. Tiles API resources and operations

Operation	Resource path	Comment
GET	/	Landing page.

Operation	Resource path	Comment
GET	/conformance	Information about specifications that this API conforms to.
GET	/collections	Lists the collections in the dataset.
GET	/collections/{collectionId}	Describes a collection.
GET	/collections/{collectionId}/images	Retrieve images of the collection, as a STAC collection.
POST	/collections/{collectionId}/images	Adds a new image by file upload (accepts GeoTIFF files).
GET	/collections/{collectionId}/images/{imageId}	Fetches an image description as a STAC item.
PUT	/collections/{collectionId}/images/{imageId}	Replace an image resource or add a new one. Currently not implemented.
DELETE	/collections/{collectionId}/images/{imageId}	Delete an image resource.

Performing image management operations, such as addition and deletion of images in a collection, also has the following side effects:

- Immediately drops the tile cache involved by the change, including all combinations of style names and tile matrix sets
- Records a checkpoint and an affected area, that the Tiles checkpoint extension will use to compute delta tile packages.

A.1.5. Tiles API endpoints

The GeoServer Tiles API offers an implementation of the OGC API-Tiles draft specification including the "Tile Matrix Set" and "Checkpoint" extension, but without the "Tiles Info" and multi-tiles conformance classes. Please note that the OGC API-Maps draft specification is not implemented.

Tiles server

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

API definition

The [API document](#) provides a machine processable description of this service API conformant to OpenAPI 3. This API document is also available as [application/x-yaml](#), [application/cbor](#), [text/html](#).

Tile matrix sets

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

Tiled Collections

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

Figure 22. GeoServer Tiles API landing page, as HTML

The overall resource layout, as well as operations on the resources, can be summarized as follows:

Table 3. Tiles API resources and operations

Resource path	Comment
/	Landing page.
/conformance	Conformance class declaration.
/collections	The collections in the dataset.
/collections/{collectionId}	Describe a collection.
/tileMatrixSets	Fetch all available tile matrix sets (tiling schemes).
/tileMatrixSets/{tileMatrixSetId}	Fetch a tile matrix sets (tiling scheme) by id.
/collections/{collectionId}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}	Fetch a data tile from a collection.
/collections/{collectionId}/map/{styleId}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}	Fetch a rendered tile from a collection.
/collections/{collectionId}/map/{styleId}/tiles	Allows to retrieve a changeset.

The service offers both map and data tiles on the two following endpoints:

- Data tiles:
`tiles/collections/{collectionId}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`.
- Rendered tiles:
`tiles/collections/{collectionId}/map/{styleId}/tiles/{tileMatrixSetId}/{tileMatrix}/{tileRow}/{tileCol}`

Either of the two endpoints might be missing depending on the configuration:

- Raster layers currently support only rendered tiles, though raw tiles in TIFF format could be offered in the future.
- Vector layers can offer either rendered or tiled data, depending on which formats are configured for tile caching (e.g., the rendered tiles endpoint will not be available if only Mapbox vector tiles support is configured for the collection).

The Tile Matrix Set conformance class allows exposing custom grids, and the `tiles` resource links back to the Tile Matrix Set for the client to discover the tile matrix set structure.

Each collection also reports the full list of available styles, which can be used as plain identifiers in the map tiles endpoints, but also links to the Styles API resource for full stylesheet retrieval. This allows for client side rendering of tiled data (e.g., Mapbox vector tiles).

Regarding the checkpoint conformance class, the multi-tile endpoint allows a client to provide a checkpoint identifier, and to retrieve tiles from the Tile API either:

- A short description of the full set of tiles modified, including affected areas (might be more than one).
- A full set of modified tiles as a ZIP package, along with the current checkpoint, to be used in future update requests.

A.1.6. Styles API endpoints

The GeoServer Styles API offers an OGC Styles API implementation including the "Manage styles" extension, but without support for resource handling.



Styles server

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

API definition

The [API document](#) provides a machine processable description of this service API conformant to OpenAPI 3. This API document is also available as [application/x-yaml](#), [application/cbor](#), [text/html](#).

Styles

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

Figure 23. GeoServer Styles API landing page, as HTML

The overall resource layout, as well as operations on the resources, can be summarized as follows:

Table 4. Styles API resources and operations

Operation	Resource path	Comment
GET	/	Landing page.
GET	/styles	Lists available styles.
POST	/styles	Adds a new style.
GET	/styles/{styleId}	Fetch a style by id.
PUT	/styles/{styleId}	Replace a style or add a new style.
DELETE	/styles/{styleId}	Delete a style.
GET	/styles/{styleId}/metadata	Fetch the metadata about a style.
PUT	/styles/{styleId}/metadata	Update the metadata document of a style.
PATCH	/styles/{styleId}/metadata	Update parts of the style metadata

The Styles API supports lookup, description and storing of styles in SLD 1.0, Symbology Encoding (SE) 1.1, Mapbox GL Styles and GeoServer GeoCSS encodings.

Each style can be stored in a single encoding, which GeoServer might be able to translate to other style encodings. In general, all encodings can be translated to SLD 1.0, though more conversion paths can be implemented in the future. GeoServer leverages this ability to offer SLD 1.0 encoding for all styles, regardless of the native encoding.

The Styles API is linked to the OGC Features and Tiles API implementations via the StyleLayer sample data links. In particular for each vector layer associated to the style, a tiles and an items link is provided to retrieve Mapbox vector tiles and GeoJSON features for client side rendering. Raster layer support is not implemented yet. Such support will be added as the Tiles API implementation adds support for tiled data, or when the Coverages API is defined and implemented.

It might be of interest that the GeoServer Styles API requires very little additional configuration. Only the style metadata needs to be entered, while the following information is inferred from the style itself:

- Style title, derived from the description elements in the SLD encoding of the style.
- Style layers, derived from the style structure.
- Style attributes, derived from the expressions used in the style, with eventual lookup in the associated vector layers to infer a data type
- Style layers sample data, from the existing associations between styles and layers in the GeoServer configuration.

A.2. Client Components and Scenario Details

All client components implemented for Testbed-15 are based on [MapStore](https://mapstore.geo-solutions.it/mapstore/#/) [https://mapstore.geo-solutions.it/mapstore/#/], an open source, framework for web mapping based on [React](https://en.wikipedia.org/wiki/React_(web_framework)) [https://en.wikipedia.org/wiki/React_(web_framework)]. MapStore wraps different mapping JavaScript

libraries, such as OpenLayers, Leaflet and Cesium. OpenLayers was selected as the map component for Testbed-15 development. The style encoding translation, needed in the visual style editor, is based on GeoStyler parsing libraries (sld, Mapbox Style and OpenLayers parsers)

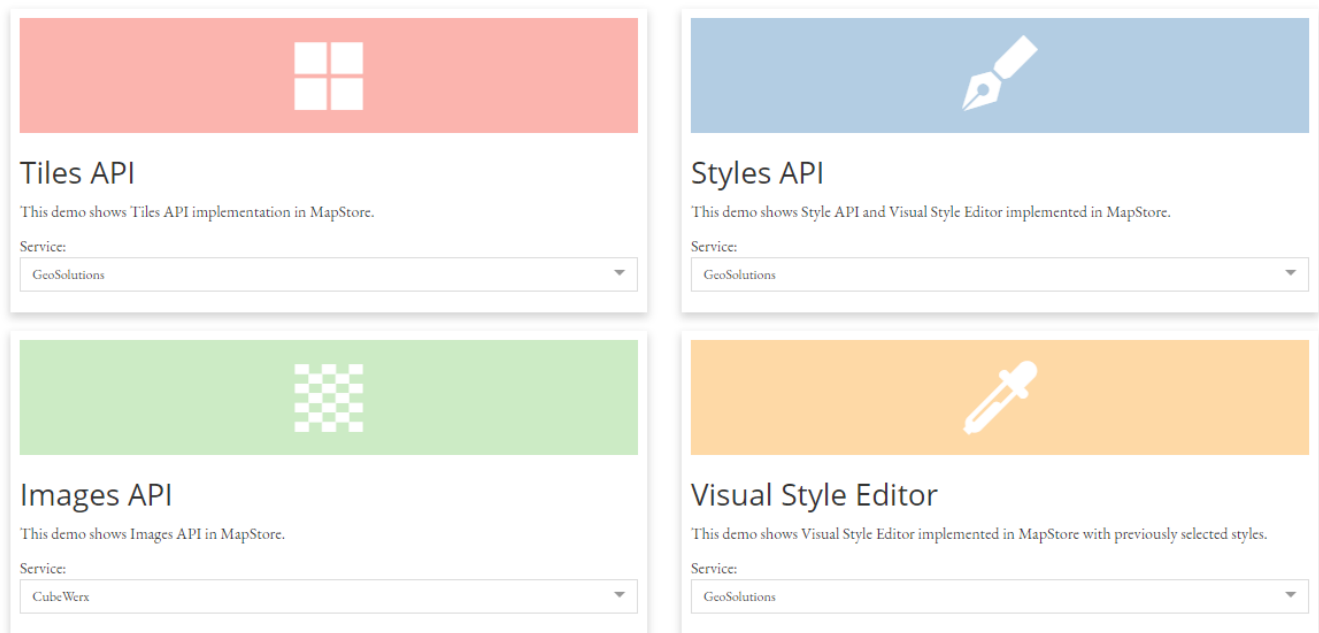


Figure 24. Client based on MapStore shows all components available in the homepage: Tiles API, Styles API, Images API and Visual Style Editor.

The client developed for Testbed-15 allows users to implement the following **scenario steps**:

- Step 1: Client: Discover data - Tiles API. [Figure 25](#)
 - The user explores tiles available from selected service by applying styles, changes format, CRS, and eventually decides to update a style.
- Step 2: Client: Discover data - Styles API. [Figure 30](#)
 - The user opens the styles API client and explores the styles available, looking for one that will be updated.
 - The user selects a style or styles and starts editing.
- Step 3: Client: Edit and Update Styles - Visual Style Editor. [Figure 35](#)
 - The user edits the style in the visual style editor.
 - The user explores different symbolizers in the user interface, modifies them, and verifies them with the feedback from the live preview.
 - The user updates the background color and tries a style encoding conversion.
- Step 4: Client: Edit and Update Styles - Visual Style Editor. [Figure 37](#)
 - The user opens the metadata panel, updates some information, and finally saves the stylesheet and metadata.
- Step 5: Client: Discover data - Styles API. [Figure 31](#)
 - The user again opens the style API client and verifies that the styles have indeed been updated.
- Step 6: Client: Discover data - Tiles API. [Figure 25](#)

- The user opens again the client and tries to use the styles previously updated on the tiles.
- The user decides to use a satellite image as background.
- Step 7: Client: Update Satellite Images - Images API. [Figure 40](#)
 - The user opens the Images API client, removes an image and loads a new one.
- Step 8: Client: Discover data - Tiles API [Figure 29](#)
 - The user again opens the Tiles API client and verifies the update of the satellite images.

Discover data - Tiles API

The Tiles API client provides tools to explore the available Tiles services.

A user can interact with the following tools:

- Catalog: a panel with a search input that paginates a collection from a service. For each record a plus button allows the user to add the collection to the map (right panel). [Figure 26](#)
- List of layers: all collections in the map are listed in this panel (left panel). [Figure 26](#)
- Layer settings: this panel provides controls for changing the format or style - if available in the collection (the panel is accessible only after selecting a layer in the list). [Figure 26](#)
- Coordinate Reference System (CRS) selector: a user can choose between EPSG:4326 or EPSG:3857 from input selector. The layer will be reprojected only if the TileMatrixSet of collection is compatible with the projection (button located at the bottom-left side of the window). [Figure 27](#)

The client has been tested with following services:

- GeoServer Tiles and Styles API
- Interactive Instrument Tiles and Styles API
- CubeWerx (tested only png tile template of satellite image)

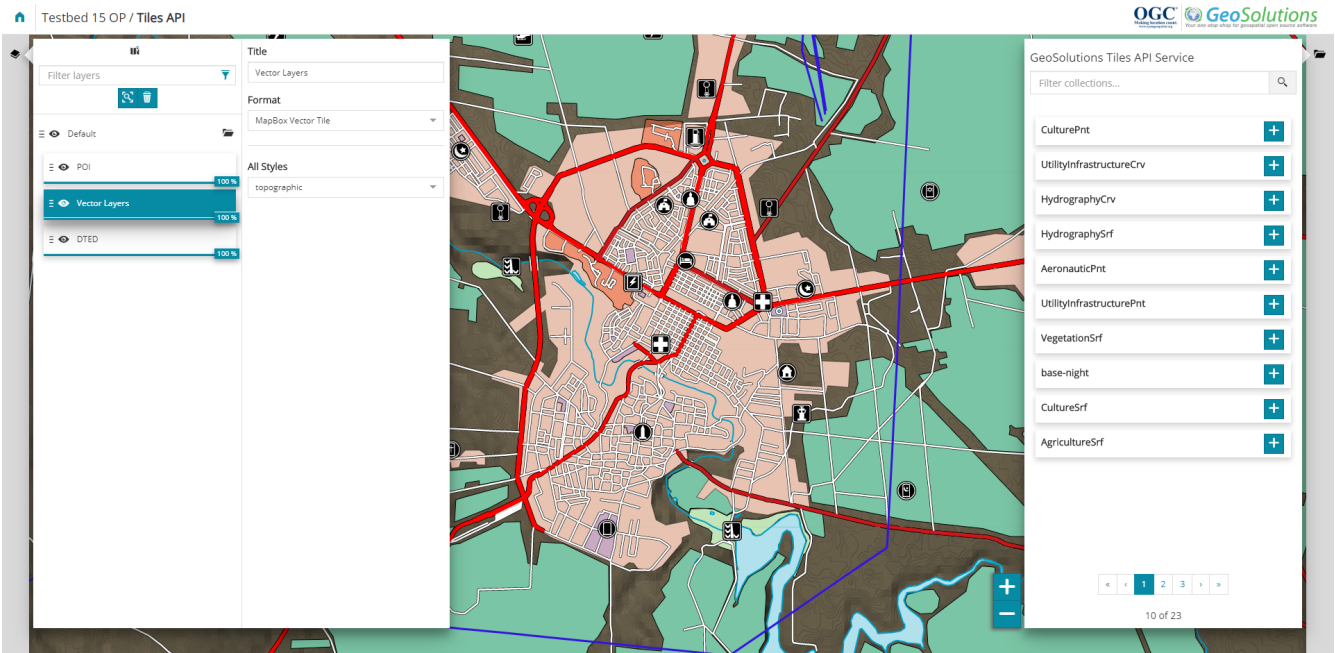


Figure 25. The Tiles API client uses GeoServer endpoints to display Points of Interest (POIs), a multilayer tiled collection and a hillshade layer with topographic style.

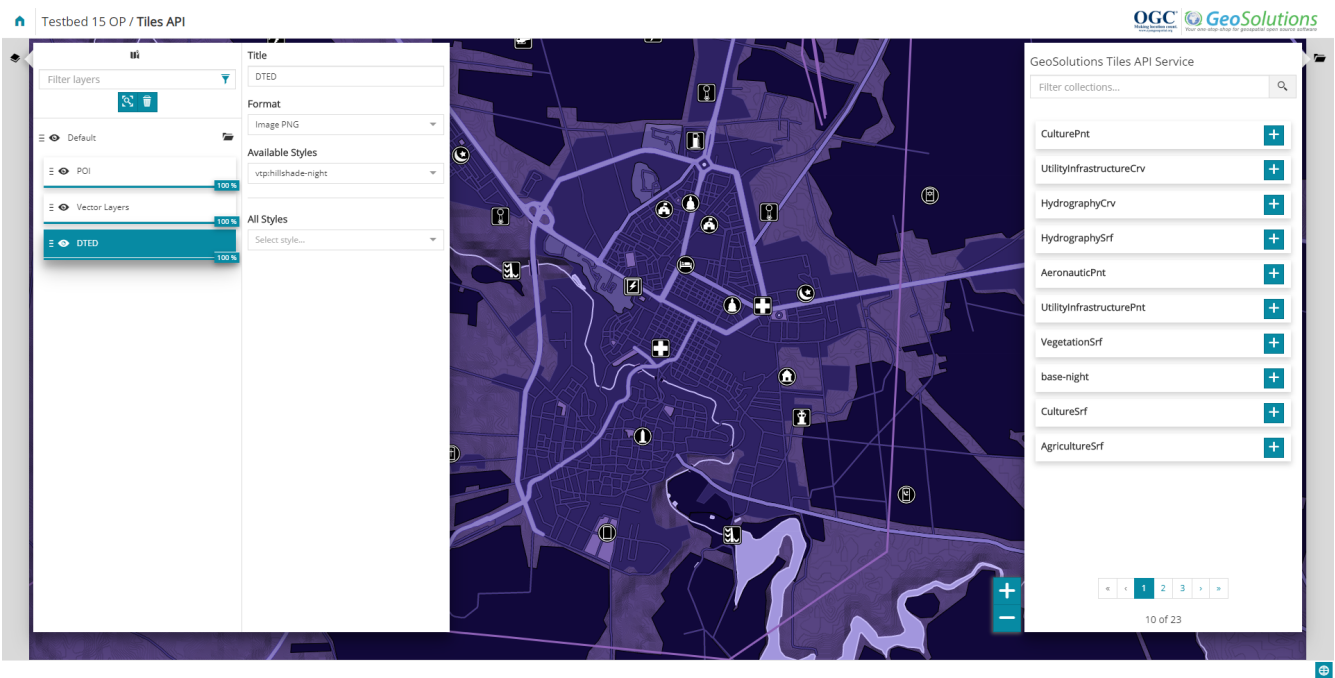


Figure 26. The Tiles API client uses GeoServer endpoints to display POIs, a multilayer collection and a hillshade layer with night style.

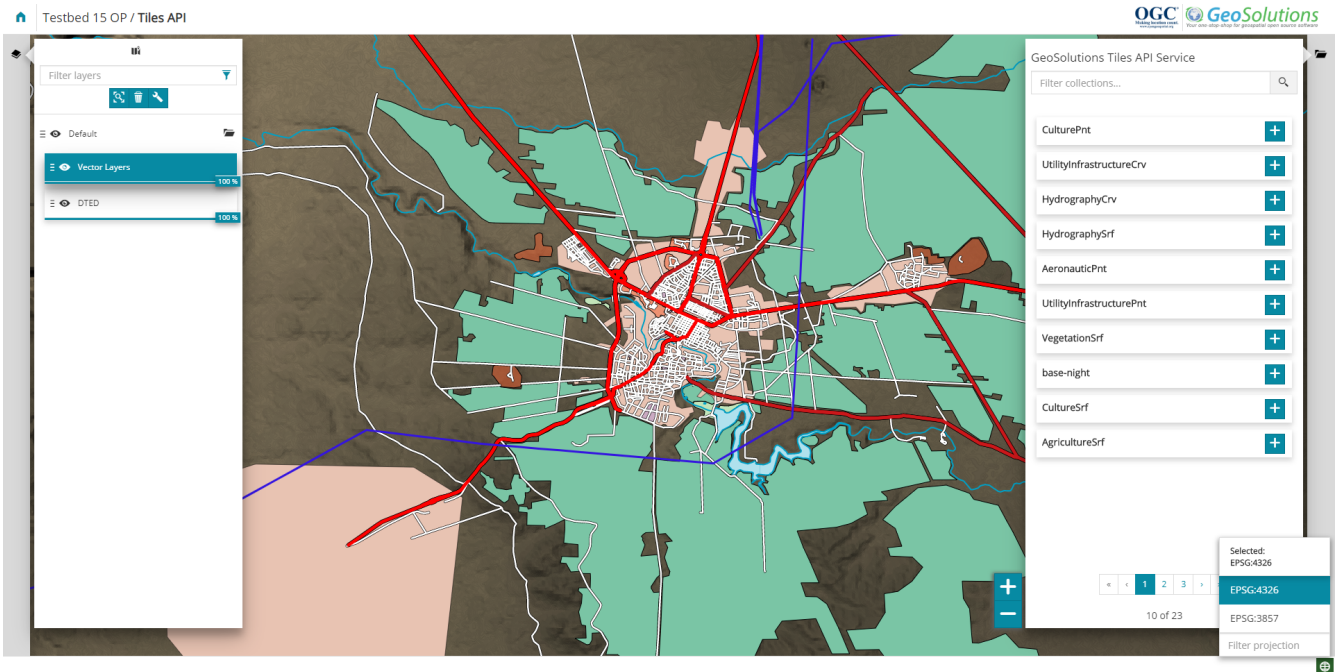


Figure 27. The Tiles API client uses GeoServer endpoints. Layers are reprojected to EPSG:4326. Select the corresponding projection using the CRS selector.

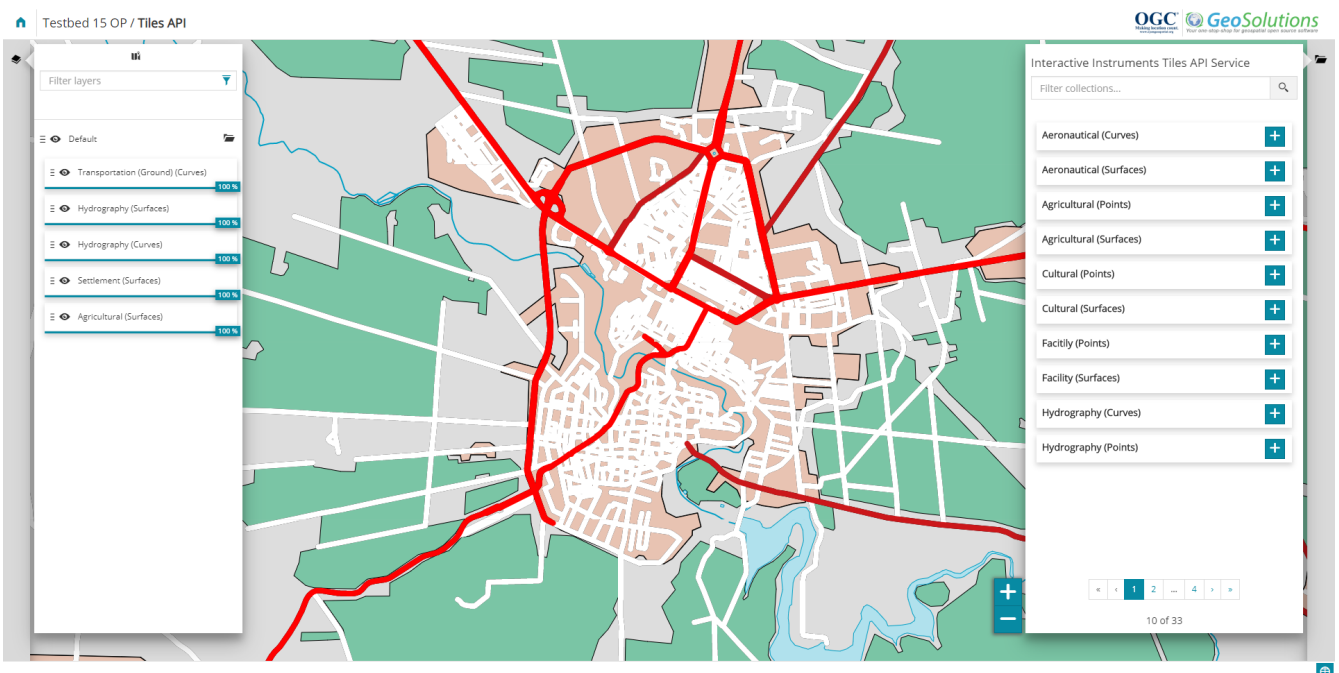


Figure 28. The Tiles API client uses Testbed-15 participant Interactive Instruments endpoints to display topographic style applied to single vector layers.

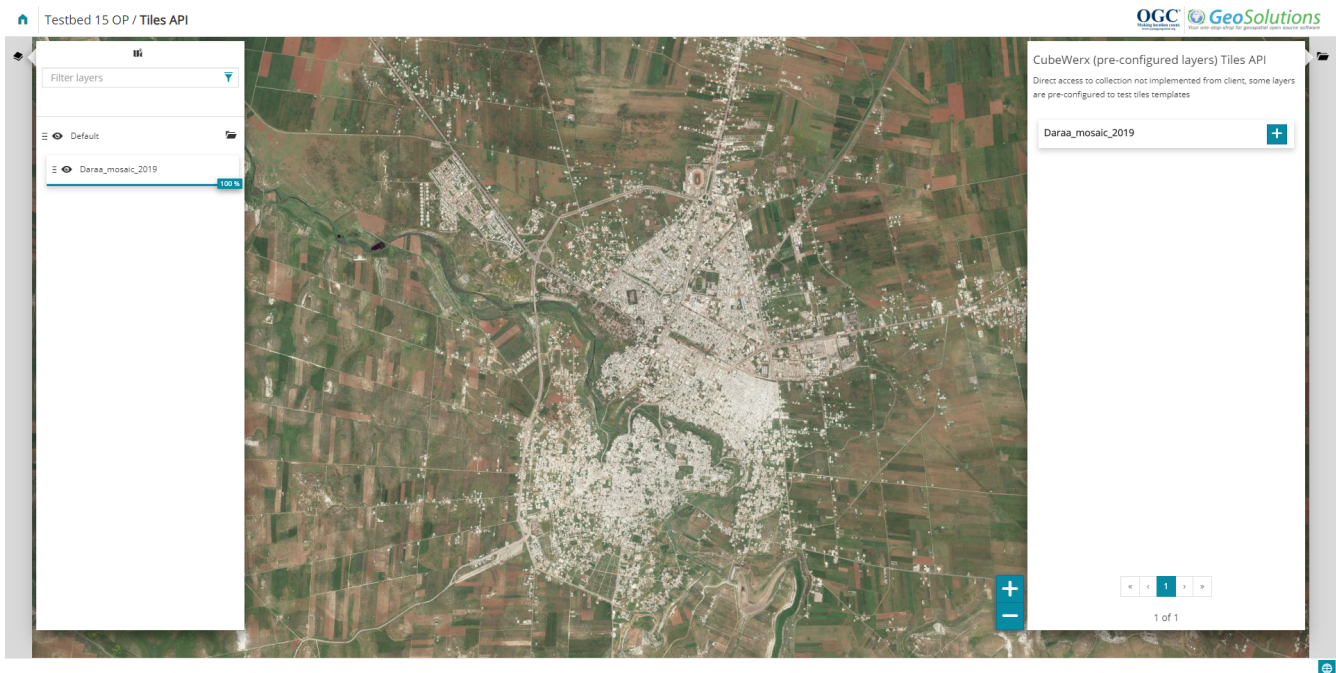


Figure 29. The Tiles API client uses the Testbed-15 participant CubeWerx tile service to display a satellite image of Daraa.

Discover data - Styles API

The Styles API client provides tools to explore styles services and list all styles available with related metadata.

A user can interact with following tools:

- Style Cards: cards with information related to the style and buttons to get metadata or remove the style. A user can select one or multiple styles to edit in the Visual Style Editor. [Figure 31](#)
- Edit button: Access to the style editor by clicking this button. [Figure 30](#)

This client has been tested with following services:

- GeoServer Styles API
- Interactive Instrument Styles API

GeoSolutions
Styles API Service
Tiles API Service (needed for Visual Style Editor)

Filter styles...

- Point of Interest**: The style supports datasets based on the TDS 6.1 specification. by OGC
- Nigth**: This topographic basemap style is designed to be used in situations with low ambient light. The style supports datasets based on the TDS 6.1 specification. by OGC
- Topographic**: A traditional topographic basemap style. The style supports datasets based on the TDS 6.1 specification. by OGC
- Night Hillshade**: A dark style for hillshade. by OGC
- Topographic Hillshade**: a bright hillshade style. by OGC

Figure 30. The Styles API client uses GeoServer endpoints - application list all the styles as cards with preview, title and description

The dialog box displays the following metadata for the 'Topographic' style:

- id: "vtp:base-topographic"
- title: "Topographic"
- links: [1 item]
- selected: false
- description: "A traditional topographic basemap style. The style supports datasets based on the TDS 6.1 specification"
- pointOfContact: "OGC"
- accessConstraints: "unclassified"
- scope: "style"
- stylesheets: [1 item]
 - Q: [5 keys]
 - title: "Stylesheet as SLD 1.0.0"
 - version: "1.0.0"
 - specification: "http://portal.opengeospatial.org/files/7artifact_id=1188"
- native: true
- link: [3 keys]
- layers: [9 items]
- loading: false

Figure 31. The Styles API client uses GeoServer endpoints to display metadata of a style in a dialog.

Interactive Instruments
 Styles API Service
 Tiles API Service (needed for Visual Style Editor)

Filter styles...

Edit Selected Style

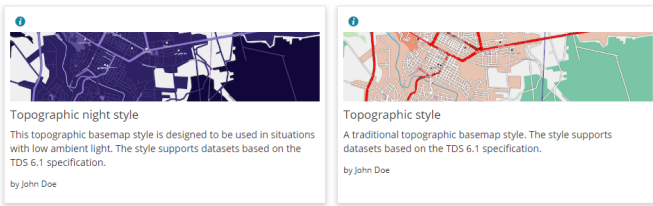


Figure 32. The Styles API client uses Interactive Instruments endpoints to list all the styles as cards with preview, title and description.

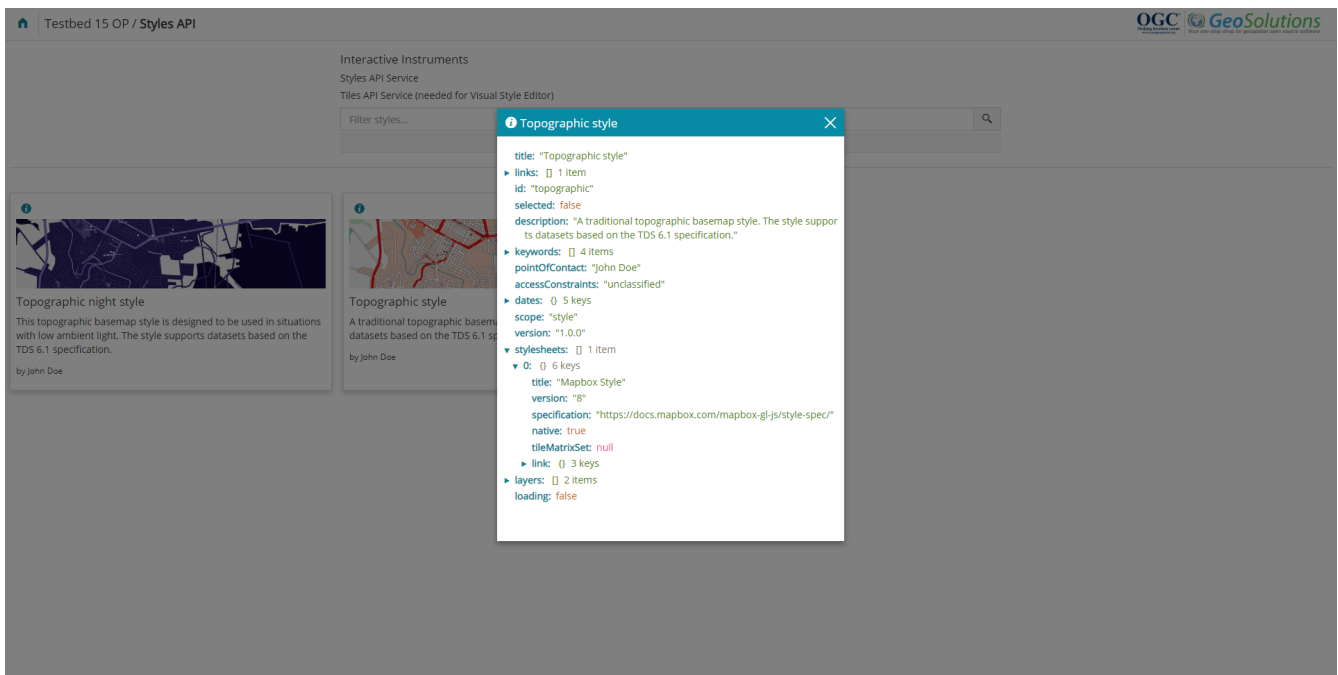


Figure 33. The Styles API client uses Interactive Instruments endpoints to visualize metadata of a style in a dialog.

Edit and Update Styles - Visual Style Editor

The Visual Style Editor provides a user interface to update styles and styles metadata, with the possibility to visualize the changes live. The Visual Style Editor also allows translating styles on the client side between SLD and Mapbox Style encodings (the conversions have some limits due to the different structure of styles and the JavaScript library in use). The Visual Style editor can be accessed after selecting one or more styles in the Styles API client. A user can select multiple styles. Multi selection provides a way to preview and edit styles from vector and raster sources at the same time.

A user can interact with following tools:

- List of layers: Layers are grouped by style and displayed in rendering order. The user can access more tools by selecting a layer or a group/style (left panel). [Figure 34](#)
- Style editor: The user can access to a visual UI for styling after selecting a layer. A text area style editor can be select for advanced styling. [Figure 34](#)
- Symbolizers component UI: For each symbolizer there is a correspondent panel with slider, inputs or color pickers. [Figure 35](#)
- Metadata/style update panel: A user can access this panel after selecting a group/style and then update metadata, background color and style at the same time. [Figure 37](#)
- Translate style client side: A user can test style conversions in the metadata panel and visualize the results in Mapbox Style in a Mapbox GL client to ensure the validity of conversion (note: sometimes the conversion results are incomplete due to different management of sprites/icon and other rules).[Figure 38](#)

The Visual Style Editor has been tested with following services:

- GeoServer Styles API
- Interactive Instrument Styles API

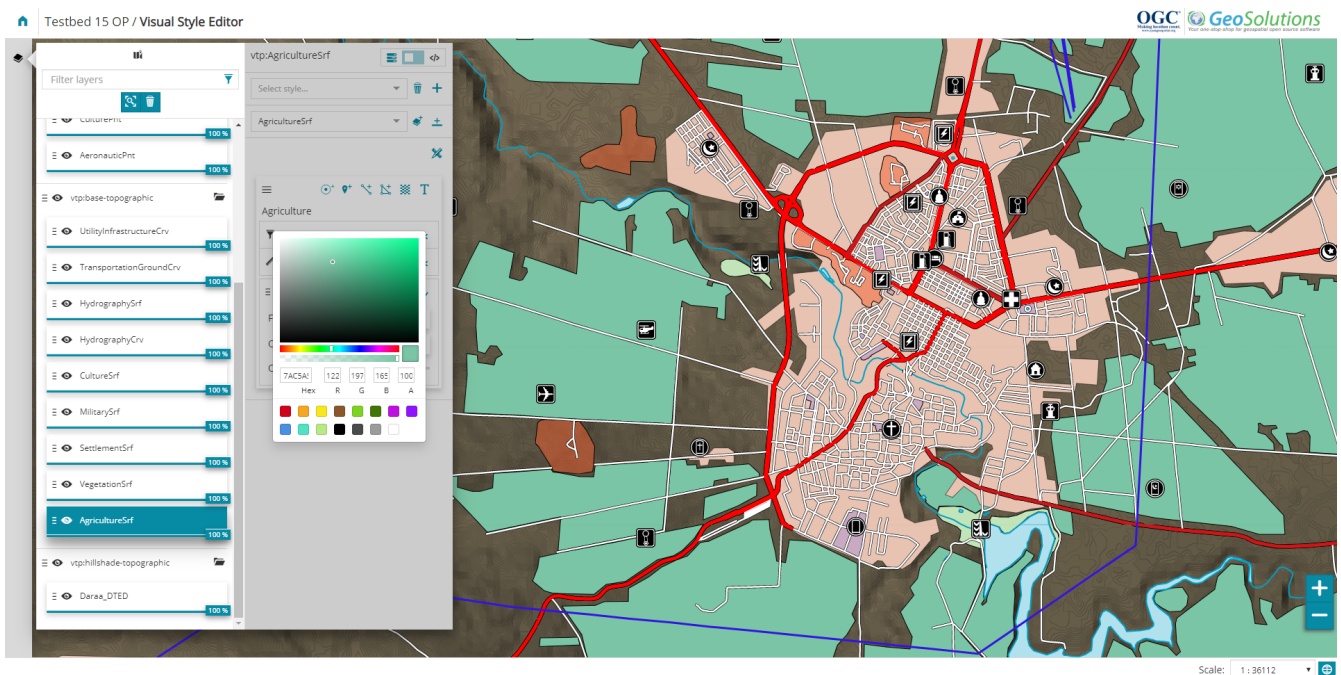


Figure 34. Visual Style Editor using GeoServer endpoints. The user is selecting a color for the agriculture surface layer with a color picker for the topographic style.

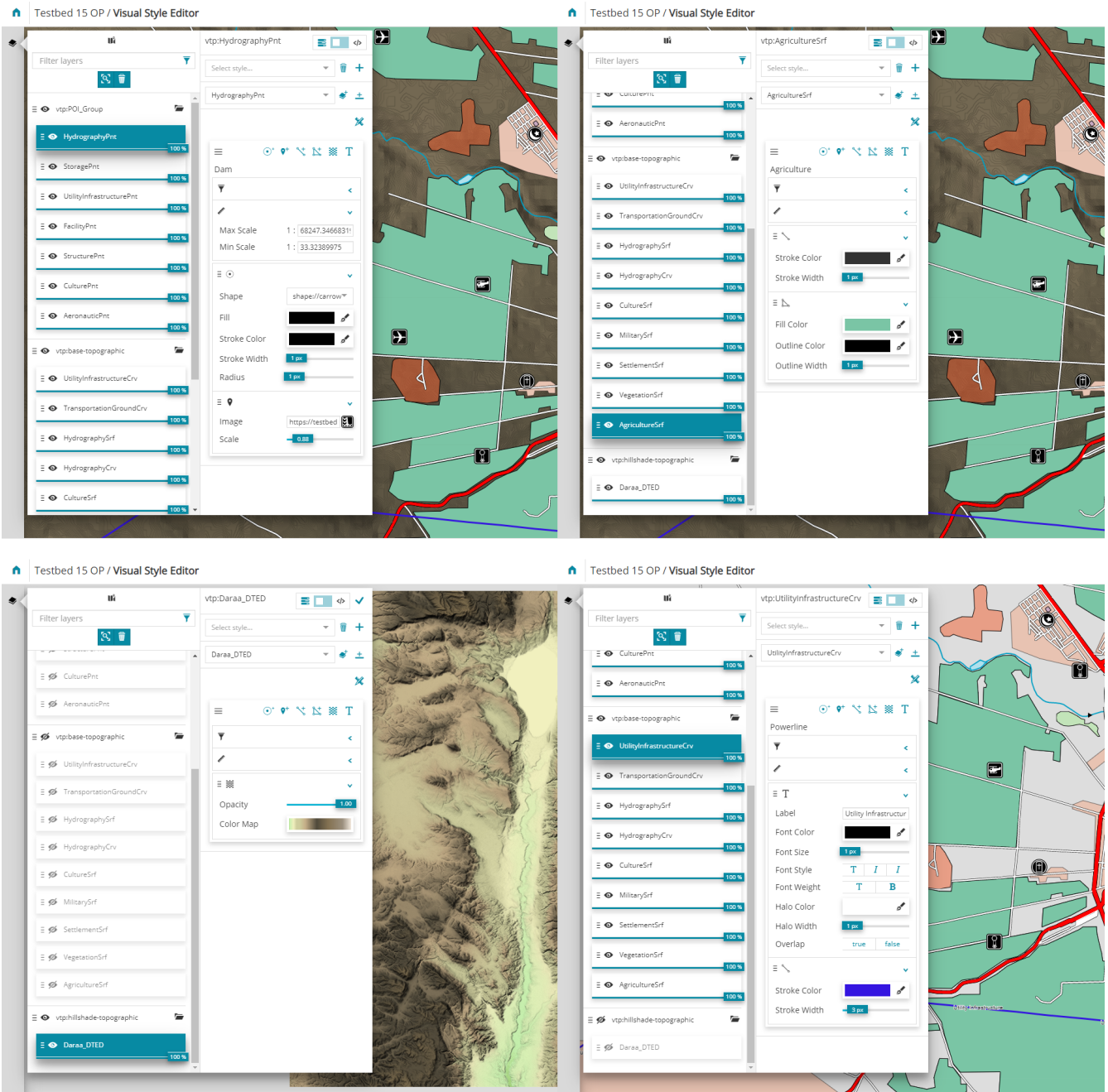


Figure 35. Visual Style Editor using GeoServer endpoints. All visual symbolizers component displayed side by side: point, mark, line, polygon, text and raster symbolizers.

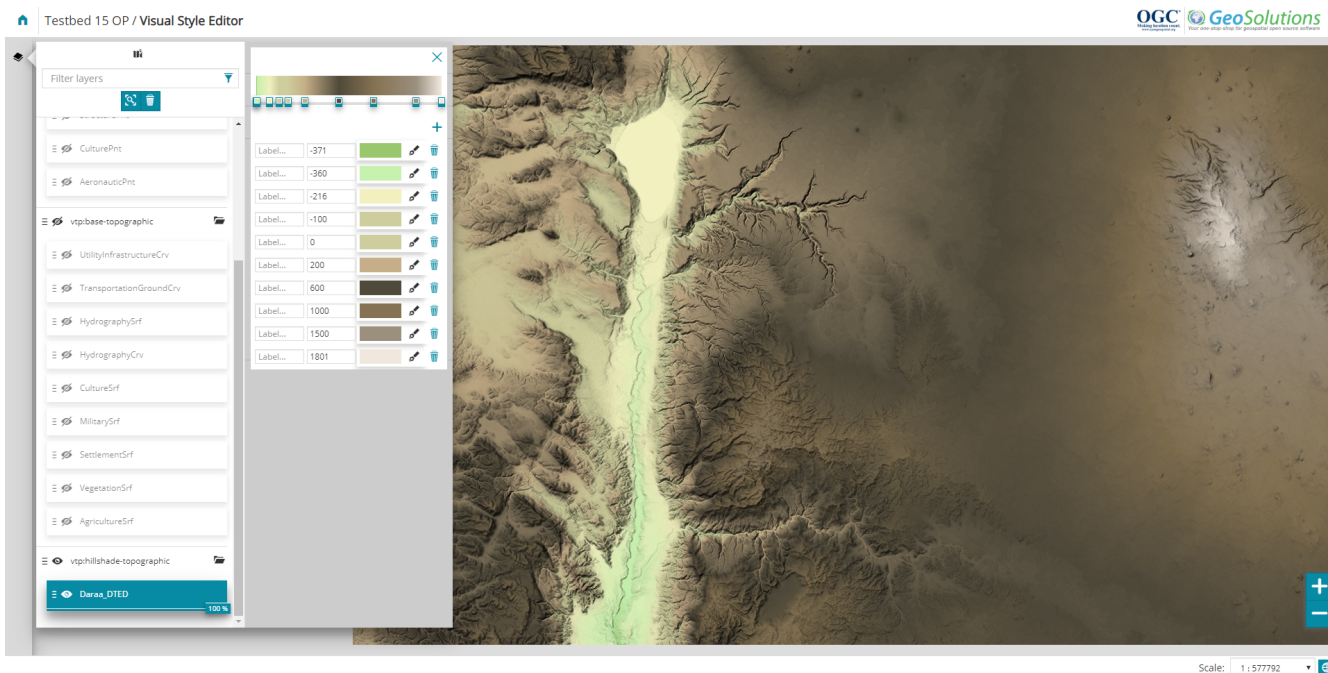


Figure 36. Visual Style Editor using GeoServer endpoints. Color map visual component being used to to update the raster color map (note: raster styles need to be manually updated by clicking on apply icon).

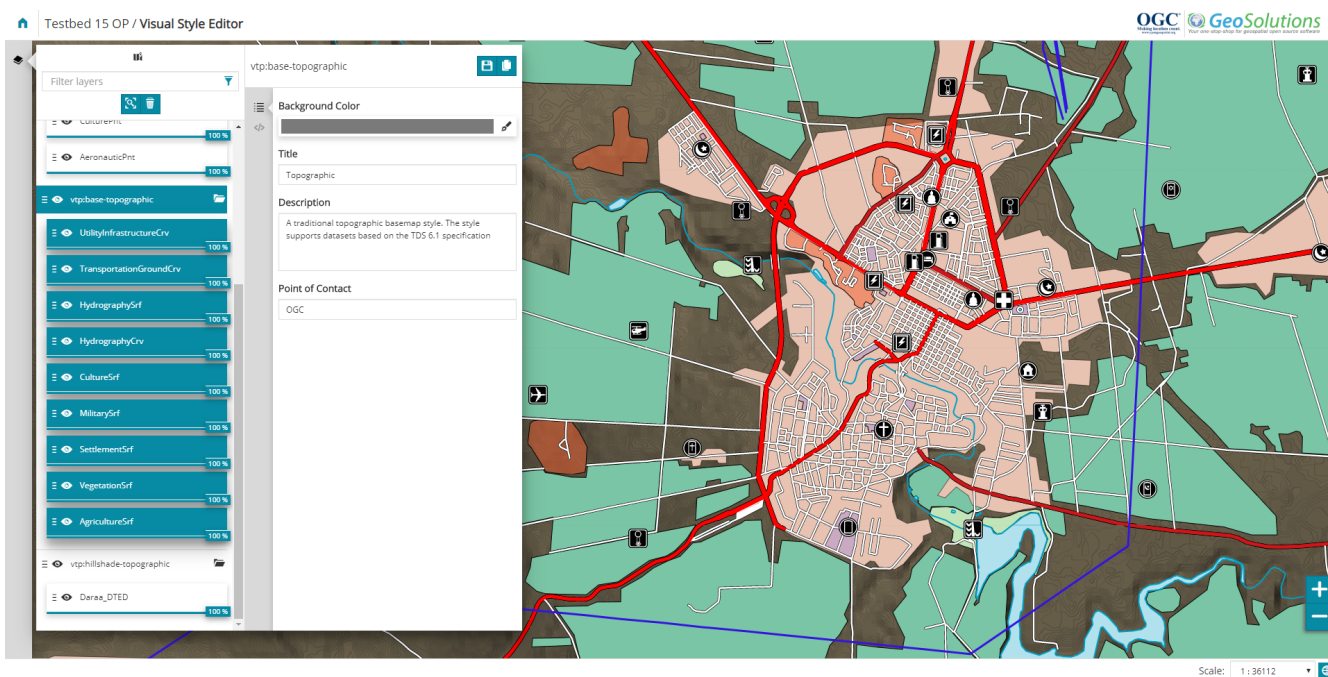


Figure 37. Visual Style Editor on GeoServer endpoints. A user is updating the style metadata, the background color picker is visible on the panel.

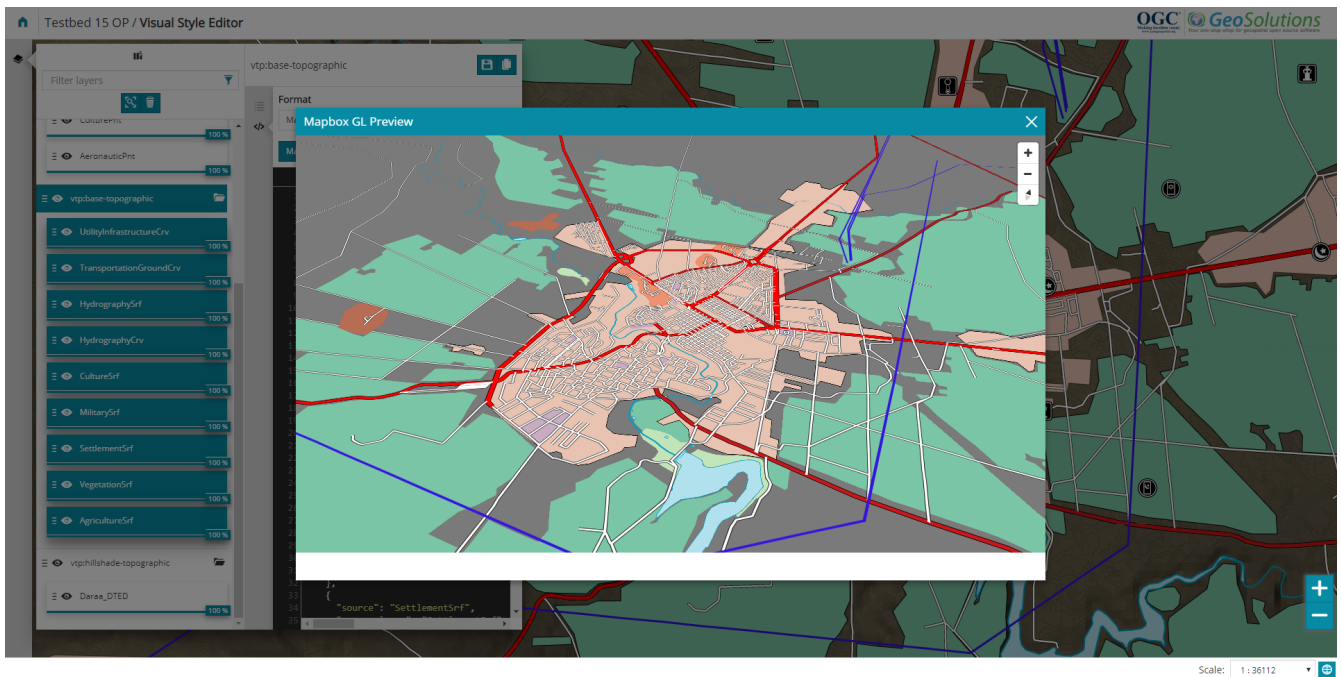


Figure 38. Visual Style Editor on GeoServer endpoints. Preview of style in a Mapbox GL client after translating a SLD to a Mapbox Style.

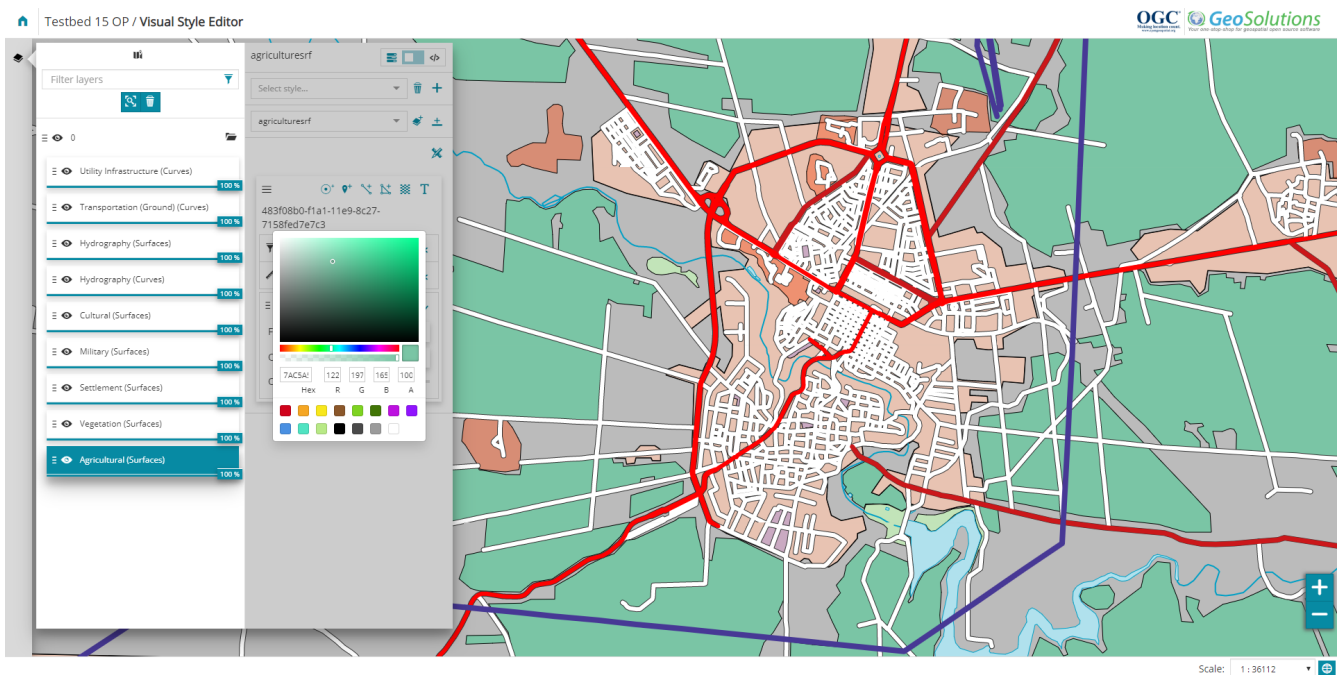


Figure 39. Visual Style Editor on Interactive Instruments endpoints. A user is selecting a color for the agriculture surface layer with the color picker for the topographic style.

Update Satellite Images - Images API

The draft OGC Images API client provides tools to explore pre-configured image data from the Images and Tiles API. A user can visualize on the same map the bounding boxes related to the mosaic images overlaid on tiled raster data from the same layer.

A user can interact with following tools:

- Images panel: Shows all the images listed in the mosaic. The user can add or remove images to the mosaic.

The Images API client has been tested with following services:

- GeoServer Images and Tiles API
- CubeWerx Images and Tiles API



Figure 40. Images API client using GeoServer endpoints. Images bounding boxes overlays on the tiles from the Daraa satellite image mosaic.

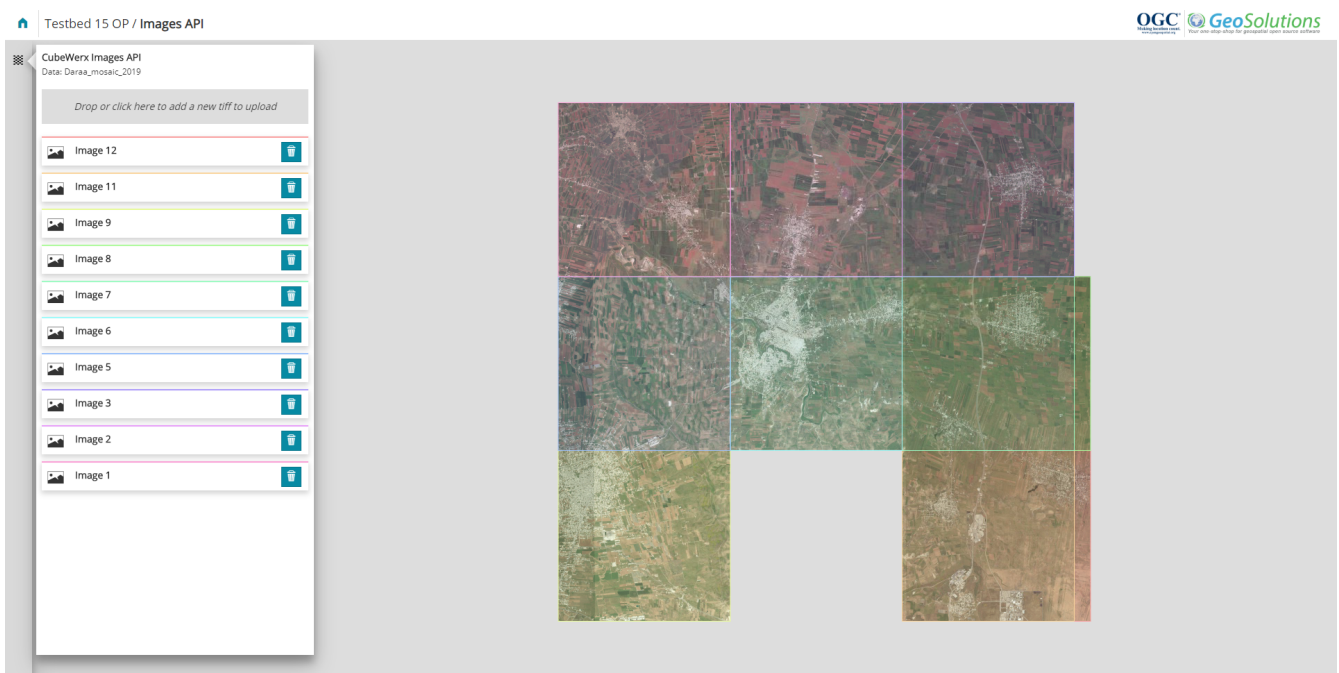


Figure 41. Images API client using CubeWerx endpoints. Images bounding boxes overlays the tiles of the Daraa satellite image mosaic, view captured after removing some images.

A.2.1. Links

Components and code

Name	Link
GeoServer OGC API modules source code	https://github.com/geoserver/geoserver/tree/a9f93393ab543e18ed0af34c2f66c18e0e16c907/src/community/ogcapi
GeoServer Testbed-15 instance home page	http://ows.geo-solutions.it/geoserver/web/
Client repository	https://github.com/geosolutions-it/VisualStyleEditor
Client demo https	https://testbed15.s3-eu-west-1.amazonaws.com/client/index.html/
Client demo http	http://testbed15.s3-eu-west-1.amazonaws.com/client/index.html/

Client libraries

Name	Link
MapStore repository	https://github.com/geosolutions-it/MapStore
OpenLayers repository	https://github.com/openlayers/openlayers
GeoStyler Mapbox parser	https://github.com/geostyler/geostyler-mapbox-parser
GeoStyler SLD parser	https://github.com/geostyler/geostyler-sld-parser
Geostyler OpenLayers parser	https://github.com/geostyler/geostyler-openlayers-parser

A.3. Implementation interactive instruments

A.3.1. Overview

interactive instruments provided a Web API for the Open Portrayal Framework scenario in Testbed-15. The Web API implements the following specifications and conformance classes:

- OGC API - Features: Core Part 1: HTML, GeoJSON, OpenAPI 3.0, Coordinate Reference Systems (by reference)
- Draft OGC API - Tiles: Core: Tiles from more than one collection, Tile Matrix Set, Multi-tiles, Collections Multi-tiles
- Draft Styles API: Core: Manage styles, Validation of styles, Resources, Manage resources, HTML, Mapbox Styles, SLD 1.0, Queryables, Style information

The API provided the following content:

- A test dataset derived from OpenStreetMap data in the region of Daraa, Syria, converted to the

Topographic Data Store schema of NGA. The dataset consists of 33 collections (feature types).

- Two topographic styles for the Daraa dataset using Mapbox Style as the stylesheet encoding, derived from styles developed by GeoSolutions in the [OGC Vector Tiles Pilot](https://www.opengeospatial.org/projects/initiatives/vt-pilot-2018) [https://www.opengeospatial.org/projects/initiatives/vt-pilot-2018>>].

The Web API uses Idproxy, available under the MIT license. The Daraa dataset was accessed from a PostgreSQL/PostGIS database.

A.3.2. New API building blocks

The following screenshots illustrate some of the new API building blocks in the HTML representation.

[Figure 42](#) shows the landing page of the Daraa Web API. The page includes new resources linked:

- "Access the data as tiled vector data" links to the [/tiles](#) path (shown in [Figure 43](#));
- "Styles to render the data" links to the [/styles](#) path (shown in [Figure 45](#));
- "Resources for rendering the data in maps" links to the [/resources](#) path providing thumbnails, symbols, sprites;
- "List of tile matrix sets implemented by this API" links to the [/tileMatrixSets](#) path providing tiling schemes (the tiling scheme WebMercatorQuad is shown in [Figure 44](#)).

Daraa

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

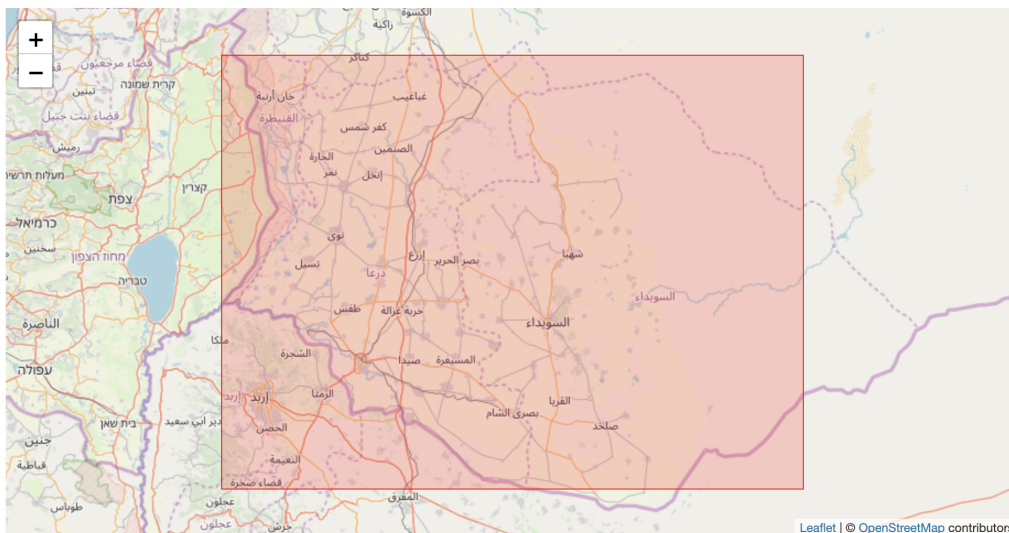
[Access the data](#)

[Access the data as vector tiles](#)

[Styles to render data](#)

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

Spatial Extent



Temporal Extent -

Expert information

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

Figure 42. The *landing page* [<https://services.interactive-instruments.de/t15/daraa>] of the Web API

Daraa

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

WebMercatorQuad

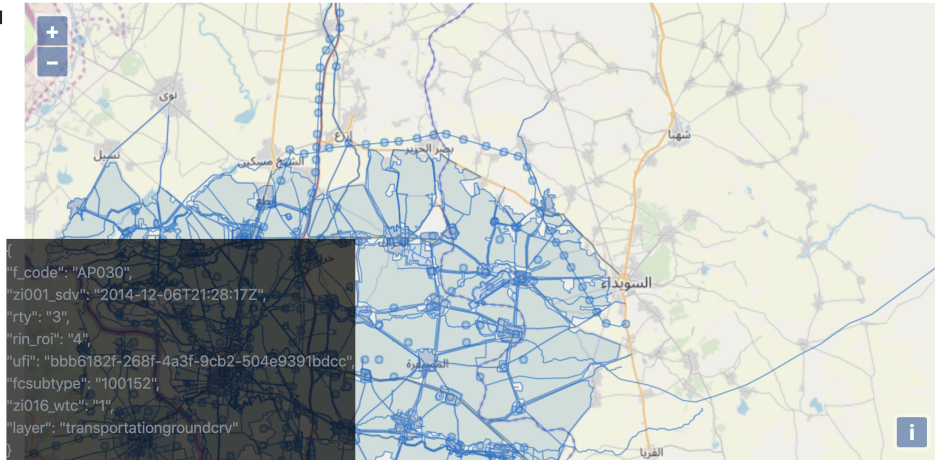


Figure 43. The [vector tiles](https://services.interactive-instruments.de/t15/daraa/tiles) [https://services.interactive-instruments.de/t15/daraa/tiles] in an OpenLayers client (without a style)

Google Maps Compatible for the World

Identifier WebMercatorQuad
<http://www.opengis.net/def/wkss/OGC/1.0/GoogleMapsCompatible>

Supported CRS <http://www.opengis.net/def/crs/EPSC/0/3857>

Bounding Box (-2.00375083427892E7 -2.00375083427892E7), (2.00375083427892E7 2.00375083427892E7)

Identifier	Scale Denominator	Top Left Corner		Tile Width	Tile Height	Matrix Width	Matrix Height
0	5.59082264028717E8	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	1.0	1.0
1	2.79541132014358E8	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	2.0	2.0
2	1.39770566007179E8	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	4.0	4.0
3	6.98852830035897E7	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	8.0	8.0
4	3.49426415017948E7	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	16.0	16.0
5	1.74713207508974E7	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	32.0	32.0
6	8735660.37544871	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	64.0	64.0
7	4367830.18772435	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	128.0	128.0
8	2183915.09386217	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	256.0	256.0
9	1091957.54693108	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	512.0	512.0
10	545978.773465544	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	1024.0	1024.0
11	272989.386732772	-2.00375083427892E7	2.00375083427892E7	256.0	256.0	2048.0	2048.0

Figure 44. The [WebMercatorQuad tiling scheme](https://services.interactive-instruments.de/t15/daraa/tileMatrixSets/WebMercatorQuad) [https://services.interactive-instruments.de/t15/daraa/tileMatrixSets/WebMercatorQuad] used for the tiles

Styles

The following styles to render feature data in maps are available via this API.

Styles	Title_ (0)
	Stylesheet in style encoding 'Mapbox Style' Style metadata Map showing the Style
	Topographic night style - Compusult - Updated (1) Stylesheet in style encoding 'Mapbox Style' Style metadata Map showing the Style
	Topographic night style (night) Stylesheet in style encoding 'Mapbox Style' Style metadata Map showing the Style
	Topographic style (topographic) Stylesheet in style encoding 'Mapbox Style' Style metadata Map showing the Style

Figure 45. Overview of the *available styles* [<https://services.interactive-instruments.de/t15/daraa/styles>] with links to the stylesheets, the style metadata and a simple client that visualizes the style (see [Figure 46](#))

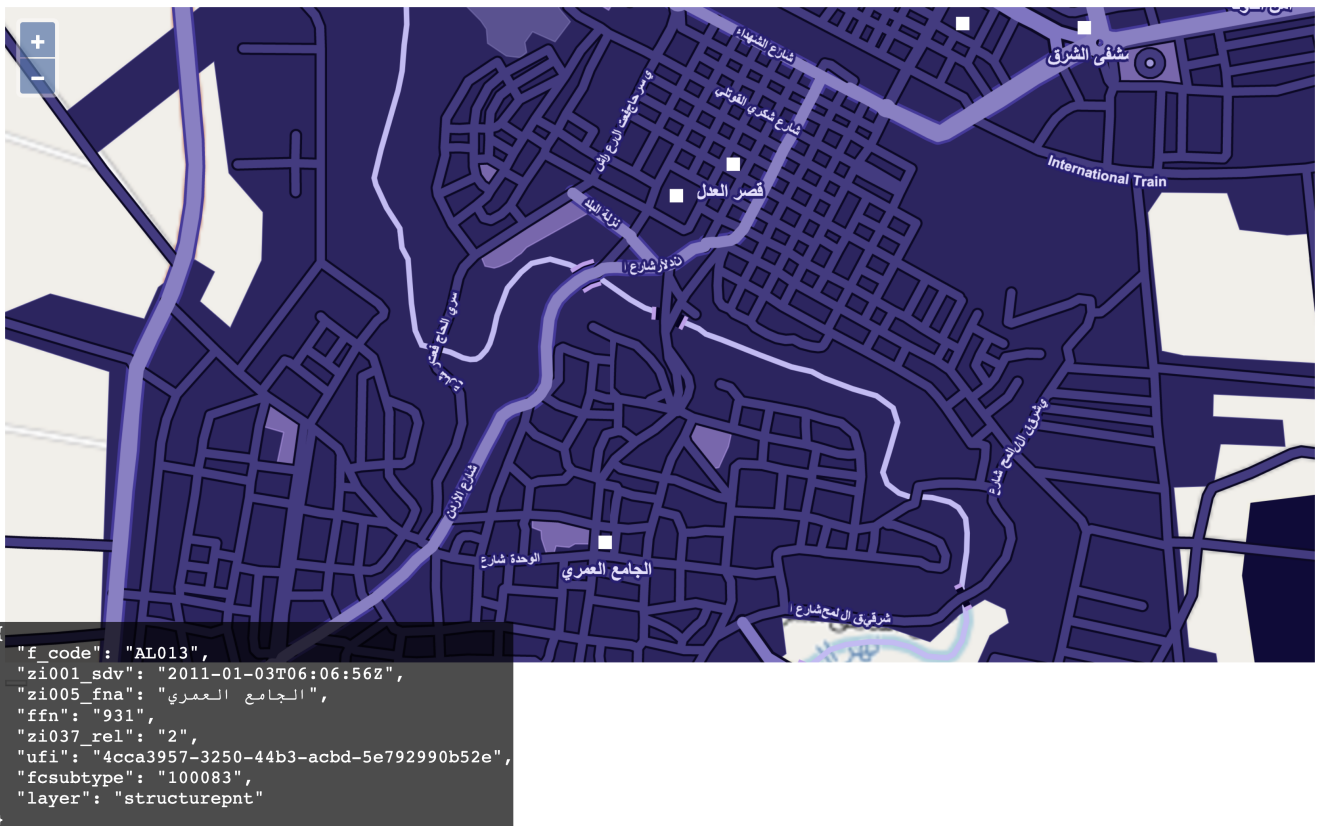


Figure 46. Visualization of the "night" style in an OpenLayers client [<https://services.interactive-instruments.de/t15/daraa/styles/night/map>]

Topographic night style

This topographic basemap style is designed to be used in situations with low ambient light. The style supports datasets based on the TDS 6.1 specification.

Version 1.0.0

Identifier night

keywordsTitle basemap
night
TDS
TDS 6.1
OGC API

Point of Contact John Doe

Preview



Dates 2019-01-01T10:05:00Z (Creation)
2019-01-01T11:05:00Z (Publication)
2019-02-01T11:05:00Z (Revision)
2019-02-01T11:05:00Z (Valid Until)
2019-02-01T11:05:00Z (Received On)

Additional Links None

Stylesheets Mapbox Style
• Specification: <https://docs.mapbox.com/mapbox-gl-js/style-spec/>
• Version: 8
• Native: No

Figure 47. The *style metadata* [<https://services.interactive-instruments.de/t15/daraa/styles/night/metadata>] of the "night" style

Queryables

"Queryables" are attributes that can be used in search or selection expressions, for example, in queries or styling rules.

f_code	Data type: string
zi005_fna	Data type: string
zi001_sdv	Data type: dateTime
rtv	Data type: string
rin_roi	Data type: string
trs	Data type: string

Figure 48. The *queryables* [<https://services.interactive-instruments.de/t15/daraa/collections/transportationgroundcrv/queryables>] for the "Transportation (Ground) (Curves)" collection

A.4. Implementation (Ecere)

Ecere developed a number of client and server components for the Testbed-15 Open Portrayal Framework, all based on its cross-platform GNOSIS Software Development Kit and written in the [eC programming language](http://ec-lang.org) [<http://ec-lang.org>].

A.4.1. Service components

Ecere deployed a new service endpoint. Based on the GNOSIS Map Server, OGC APIs were implemented for Testbed-15 in this endpoint.

The endpoint presents all supported OGC API capabilities as parts of a single resource tree.

The modular OGC API building blocks implemented for Testbed-15's Open Portrayal Framework include:

- (Vector) Features
- Tiles
- Styles (retrieval only)

The service currently offers the data in the EPSG:4326 (plate carree, WGS84 spheroid) coordinate reference system (CRS). Using the draft OGC Tiles API specification, the data is provided in the tiling scheme's CRS, except for vector data formats mandating EPSG:4326 (e.g. GeoJSON). A number of tiling schemes based on both spherical Mercator and EPSG:4326 are currently supported.

The data is available in different data formats:

- GeoJSON (vector)
- Mapbox Vector Tiles (vector)
- GNOSIS Map Tiles (vector, imagery, coverage — tiles only)
- 16-bit PNG (coverage with a specific range intended to accommodate digital elevation models, until the service offers GeoTIFF support)
- PNG and JPEG (imagery)
- MapML (vector, support for which was implemented during Testbed-15 as the focus of a work package within the Machine Learning thread)

Support for GML and GeoECON, available in earlier iterations of the GNOSIS Map Server, will also be re-enabled at the endpoint presented above.

Data layers served

A number of data layers are accessible from this service, including data from Natural Earth, NASA BlueMarble (next generation), and regional sections of OpenStreetMap. The data layers specific to the Testbed's Open Portrayal Framework is organized within a *vtp* (for the Vector Tiles Pilot where they were first used) group of collections. Specifically, the following sub-groups of layers are used for Testbed-15's experiments:

- [Daraa2](http://maps.ecere.com/geoapi/collections/vtp/Daraa2) [http://maps.ecere.com/geoapi/collections/vtp/Daraa2] (vector layers originating from OpenStreetMap data in Daraa, Syria, following NGA's Topographic Data Store schema)
- [Imagery](http://maps.ecere.com/geoapi/collections/vtp/Imagery) [http://maps.ecere.com/geoapi/collections/vtp/Imagery] (satellite imagery over Daraa)
- [Daraa_DTED](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/Daraa_DTED) [http://maps.ecere.com/geoapi/collections/vtp/Daraa2/Daraa_DTED] (Digital Terrain Elevation Data of Daraa)

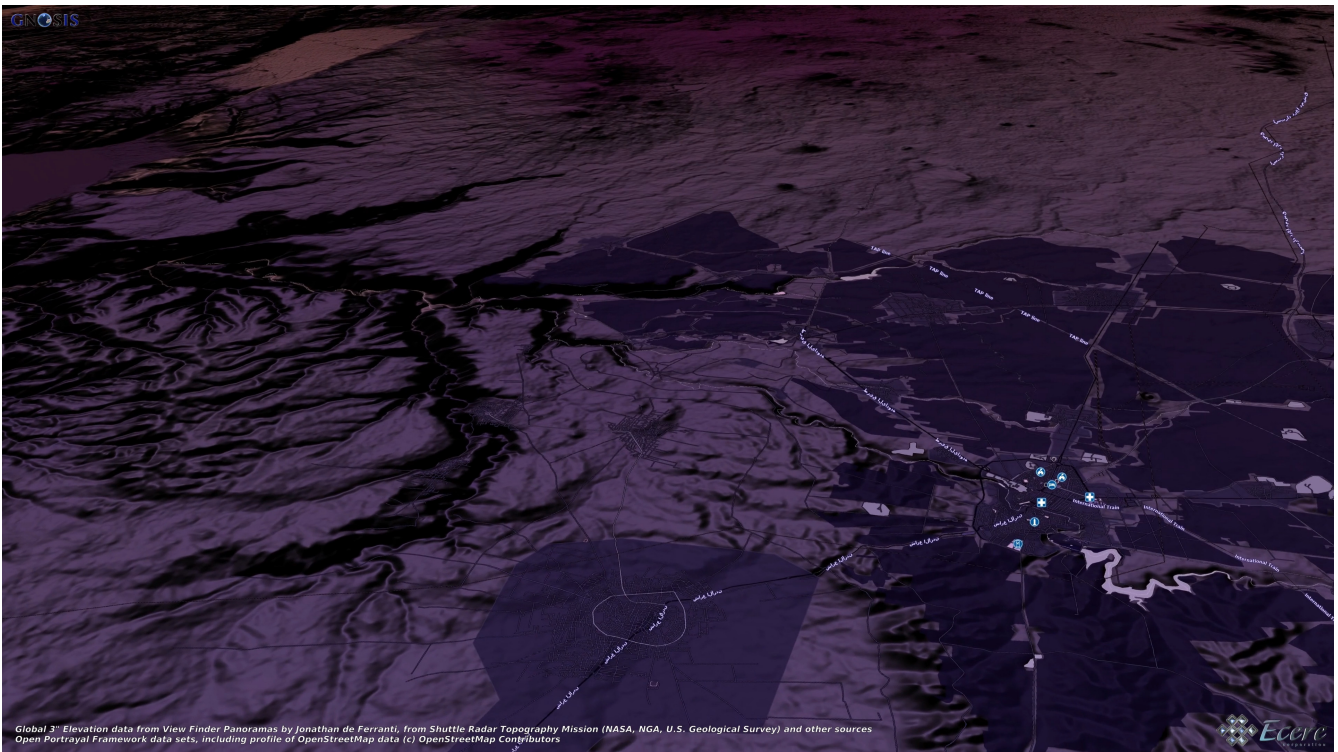


Figure 49. Daraa vector layers rendered in night style over hill-shaded elevation data visualized in GNOISIS Cartographer



Figure 50. Daraa vector layers rendered in topographic style over imagery visualized in GNOISIS Cartographer

The GNOISIS Map Server currently provides groups of collections as recursive layers within the /collections/ path. The service does not yet follow a standard approach for managing large quantities of organized collections from a single service end-point. A standardized approach has not yet been defined, as the only proposed extension for this so far has been a separate /themes resource. However, the separate /themes resource does not address things such as the large payload of listing collections, or managing multi-layer tiles. At the time of writing this report, the

{collectionID} thus contains the invalid '/' characters. The colon (:) separator might end up being used as an alternative in order to conform with the OGC API specifications.

For the moment, direct links to these groups of collections fall under the /collections/ resource with {collectionID} containing slashes, e.g. <http://maps.ecere.com/geoapi/collections/vtp/Daraa2> .

OGC API modules

The following modular OGC API capabilities are available for all applicable data layers served at the deployed endpoint.

Common

Table 5. Ecere OGC API - Common resources

Resource path	Description
/	Landing page.
/api	API description (<i>NOTE: currently missing</i>)
/conformance	API conformance (<i>NOTE: currently missing</i>)
/collections	The list of available collections
/collections/{collectionId}	The description of a specific collection
/collections/{collectionId}/items	The data of a specific collection

Example direct paths to collections listing and description:

<http://maps.ecere.com/geoapi/collections/> (List of all collections served)

<http://maps.ecere.com/geoapi/collections/vtp/Daraa2/> (List of all collections served within the Daraa2 group)

<http://maps.ecere.com/geoapi/collections/vtp/Daraa2/AgricultureSrf> (Agricultural surfaces collection description)

Vector features

Table 6. Ecere OGC API - Features resources

Resource path	Description
/collections/{collectionId}/items	The vector features for a specific collection
/collections/{collectionId}/schema	WFS-style schema listing attributes and their types

Example direct paths to vector features:

<http://maps.ecere.com/geoapi/collections/vtp/Daraa2/AgricultureSrf/items> (Agricultural surfaces vector data)

<http://maps.ecere.com/geoapi/collections/vtp/Daraa2/AgricultureSrf/schema> (Agricultural surfaces attributes schema)

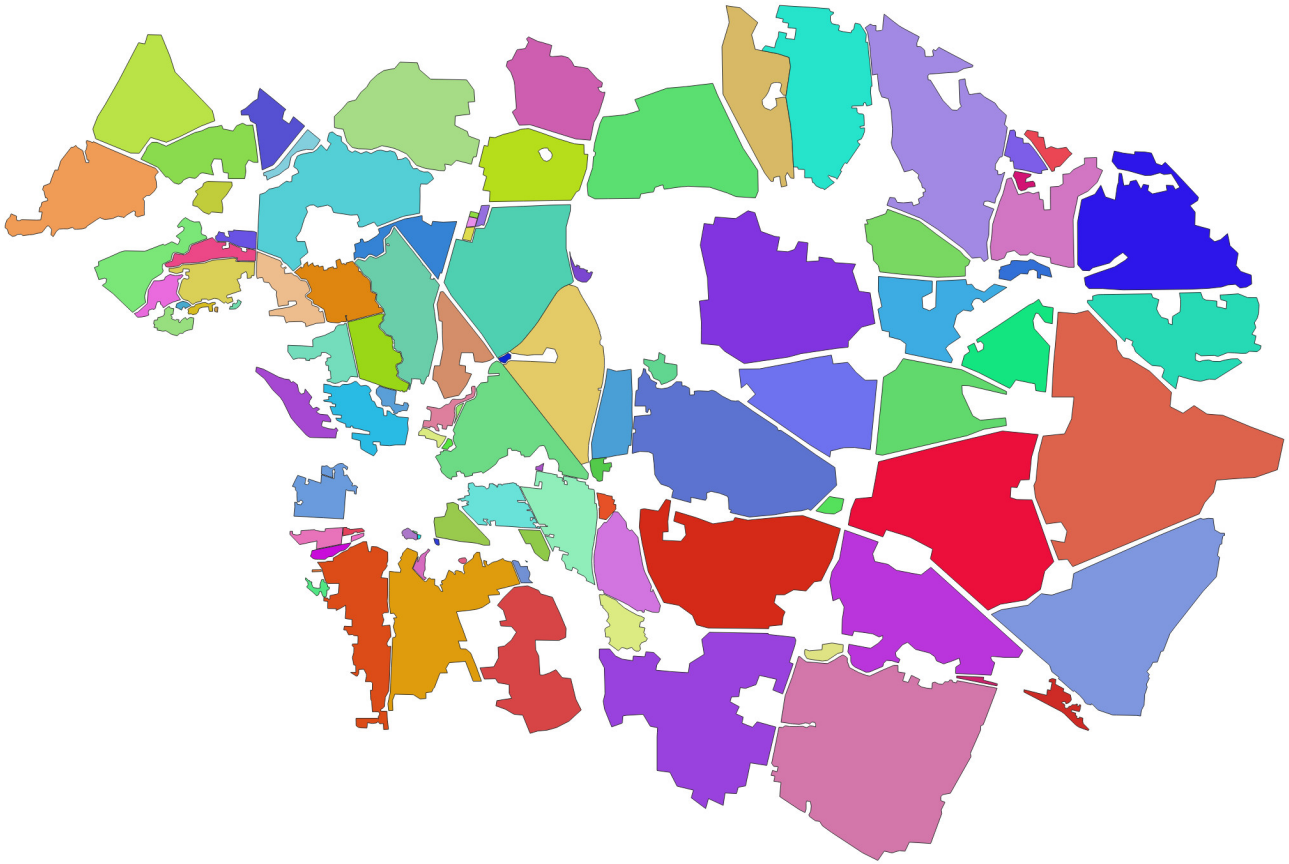


Figure 51. *Agricultural surfaces polygons returned as GeoJSON by above request*

The features end-point supports a number of parameters:

- *f* (specify encoding)
- *bbox* (an intersecting bounding box)
- *clipBox* (a clipping bounding box)
- *zoomLevel* (a zoom level to specify a desired generalization level, based on the selected tiling scheme, or defaulting to the [GNOSIS Global Grid](http://docs.opengeospatial.org/is/17-083r2/17-083r2.html#106) [<http://docs.opengeospatial.org/is/17-083r2/17-083r2.html#106>])
- *tilingScheme* (the tiling scheme defining the scale set to which *zoomLevel* is referring)
- *time* (for temporal datasets)
- *properties* (specify the list of properties to include; geometry is excluded if this parameter is specified, and it does not include 'geometry')

The intersecting bounding box will return the entire feature, identical with traditional WFS behavior.

On the other hand, the clipping bounding box can be used for requesting only a portion of a high resolution very large single feature, such as for example a single polygon for the whole of Canada with highly detailed coastline.

The zoom level parameter offers a generalized version of the data without requiring the use of tiles.

(A)RGB Imagery

Table 7. Ecere OGC API - (A)RGB Imagery resources

Resource path	Description
/collections/{collectionId}/items	The data for the whole layer (as JPEG or PNG)

The following parameters are supported:

- *f* (specify encoding)
- *bbox* or *clipBox* (a clipping bounding box)
- *zoomLevel* (a zoom level to specify a desired generalization level, based on the selected tiling scheme, or defaulting to the GNOSIS Global Grid)
- *tilingScheme* (the tiling scheme defining the scale set to which *zoomLevel* is referring)
- *width* and/or *height* in pixels
- *time* (for temporal datasets)

Example direct paths to imagery:

<http://maps.ecere.com/geoapi/collections/vtp/Imagery/Imagery1/items> (imagery data)



Figure 52. Daraa imagery returned by above request

Coverage

Table 8. Ecere OGC API - Coverage resources

Resource path	Description
/collections/{collectionId}/items	The data for the whole layer (as 16-bit PNG, eventually as GeoTIFF)

The following parameters are supported:

- *f* (specify encoding)
- *bbox* or *clipBox* (a clipping bounding box)
- *zoomLevel* (a zoom level to specify a desired generalization level, based on the selected tiling scheme, or defaulting to the GNOSIS Global Grid)
- *tilingScheme* (the tiling scheme defining the scale set to which *zoomLevel* is referring)
- *width* and/or *height* in pixels
- *time* (for temporal datasets)

Example direct paths to coverage (elevation data):

http://maps.ecere.com/geoapi/collections/vtp/Daraa2/Daraa_DTED/items (elevation, currently served as 16-bit PNG)



Figure 53. Daraa DTED elevation data returned by above request (16-bit PNG)

Tiles

Table 9. Ecere OGC API - Tiles resources

Resource path	Description
/tiles	The list of all tiling schemes

/tiles/{tilingSchemeId}	The description of a specific tiling scheme
/collections/{collectionId}/tiles/{tilingSchemeId}/{level}/{row}/{col}	The data for a specific tile from a collection
/collections/{collectionGroup}/tiles/{tilingSchemeId}/{level}/{row}/{col}	Multi-layer data for a specific tile from a group of collections

Vector tiles are available for either individual layers, or as multi-layer Mapbox Vector Tiles for a group of vector collections.

The Tiles API supports the *Tile Matrix Set* extension to describe the tiling scheme.

Most of the tiling schemes supported are defined in the OGC Web Map Tile Service (WMTS) standard. The supported tiling schemes include:

- [GlobalCRS84Scale](http://maps.ecere.com/geoapi/tiles/GlobalCRS84Scale) [http://maps.ecere.com/geoapi/tiles/GlobalCRS84Scale] (EPSG:4326, scale set with round scale denominators, defined in WMTS 1.0 Appendix E.1)
- [GlobalCRS84Pixel](http://maps.ecere.com/geoapi/tiles/GlobalCRS84Pixel) [http://maps.ecere.com/geoapi/tiles/GlobalCRS84Pixel] (EPSG:4326, scale set with round pixels per degrees, defined in WMTS 1.0 Appendix E.2)
- [GoogleCRS84Quad](http://maps.ecere.com/geoapi/tiles/GoogleCRS84Quad) [http://maps.ecere.com/geoapi/tiles/GoogleCRS84Quad] (EPSG:4326, quad-tree/power of 2 scale set defined in WMTS 1.0 Appendix E.3)
- [CRS84Quad2L0Tiles](http://maps.ecere.com/geoapi/tiles/CRS84Quad2L0Tiles) [http://maps.ecere.com/geoapi/tiles/CRS84Quad2L0Tiles] (EPSG:4326, same as GoogleCRS84Quad, but offset by one level to avoid intricacies)
- [GNOSISGlobalGrid](http://maps.ecere.com/geoapi/tiles/GNOSISGlobalGrid) [http://maps.ecere.com/geoapi/tiles/GNOSISGlobalGrid] (EPSG:4326, [variable tile width matrix](http://docs.opengeospatial.org/is/17-083r2/17-083r2.html#14) [http://docs.opengeospatial.org/is/17-083r2/17-083r2.html#14], defined in [OGC TMS 1.0 Appendix H.2](http://docs.opengeospatial.org/is/17-083r2/17-083r2.html#106) [http://docs.opengeospatial.org/is/17-083r2/17-083r2.html#106])
- [GoogleMapsCompatible](http://maps.ecere.com/geoapi/tiles/GoogleMapsCompatible) [http://maps.ecere.com/geoapi/tiles/GoogleMapsCompatible] (EPSG:3857, spherical Mercator quad-tree scale set defined in WMTS 1.0 Appendix E.4)

The support for CRS84Quad2L0Tiles (known as WorldCRS84Quad in the new Tile Matrix Set standard), a variant of WMTS 1.0 "GoogleCRS84Quad" well-known scale set skipping level 0, was added during the initiative.

Example direct paths to tiles:

<http://maps.ecere.com/geoapi/collections/vtp/>

[Daraa2/tiles/GoogleMapsCompatible/11/827/1229.mvt](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/tiles/GoogleMapsCompatible/11/827/1229.mvt) [http://maps.ecere.com/geoapi/collections/vtp/Daraa2/tiles/GoogleMapsCompatible/11/827/1229.mvt] (multi-layer Mapbox Vector Tile)

[Daraa2/AgricultureSrf/tiles/GoogleMapsCompatible/11/827/1229](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/AgricultureSrf/tiles/GoogleMapsCompatible/11/827/1229) [http://maps.ecere.com/geoapi/collections/vtp/Daraa2/AgricultureSrf/tiles/GoogleMapsCompatible/11/827/1229] (Agricultural surfaces vector tile)

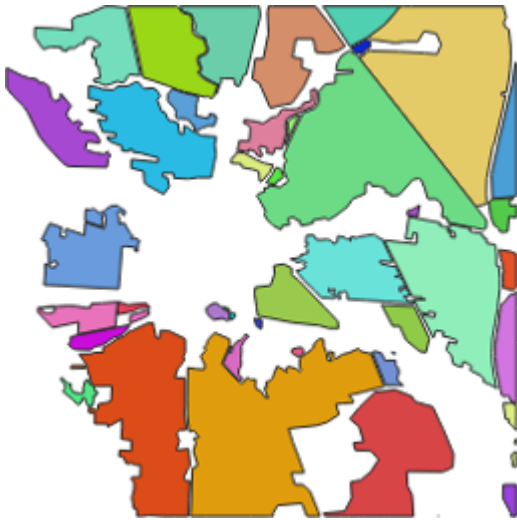


Figure 54. *Agricultural surfaces vector tile*

[Daraa2/Daraa_DTED/tiles/GoogleMapsCompatible/11/827/1229](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/Daraa_DTED/tiles/GoogleMapsCompatible/11/827/1229) [http://maps.ecere.com/geoapi/collections/vtp/Daraa2/Daraa_DTED/tiles/GoogleMapsCompatible/11/827/1229] (Elevation data tile)



Figure 55. *Elevation data tile (16-bit PNG)*

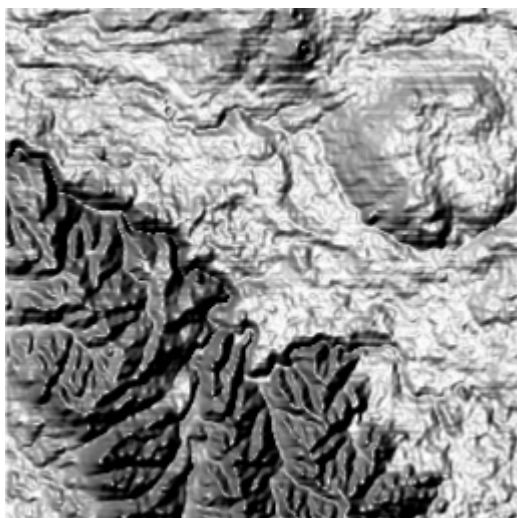


Figure 56. *Elevation data tile (hillshaded with QGIS)*

[Imagery/Imagery1/tiles/GoogleMapsCompatible/11/827/1229](http://maps.ecere.com/geoapi/collections/vtp/Imagery/Imagery1/tiles/GoogleMapsCompatible/11/827/1229) [<http://maps.ecere.com/geoapi/collections/vtp/Imagery/Imagery1/tiles/GoogleMapsCompatible/11/827/1229>] (Imagery tile)



Figure 57. *Imagery tile*

Styles

Table 10. *Ecere OGC API - Styles resources*

Resource path	Description
/collections/{collectionId}/styles	The list of all styles associated with this collection
/collections/{collectionId}/styles/{styleId}	The style data for a given style

The style sheets are available in GNOSIS CMSS, SLD/SE and Mapbox GL JSON styles. The server can either serve the different encodings (one or more) directly if available for each style, or generate the missing ones on-the-fly. Translation from one style encoding to another poses numerous challenges, as there are major differences in approach from one styling language and renderer to another. The importing and exporting capabilities from the different encodings to the native GNOSIS CMSS are continuously being improved (cf. [styles conversion section of this ER](#)).

The association between the styles and the data layers is done by either storing style sheets with the source data store itself, or through the concept of a *stylable layer set*. The stylable layer set is a unique resource identifier which both the layers and styles can be associated with. One or more *stylable layer set* can be specified within the styles and layers metadata.

Example direct paths to styles:

<http://maps.ecere.com/geoapi/collections/vtp/>

[Daraa2/styles](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles) [<http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles>] (list of all styles)

[Daraa2/styles/night](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/night) [<http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/night>] (night style)

[Daraa2/styles/topographic](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/topographic) [<http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/topographic>] (night style)

[Daraa2/styles/overlay](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/overlay) [<http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/overlay>] (overlay style)

[Daraa2/styles/night.sld](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/night.sld) [<http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/night.sld>] (night style, SLD/SE encoding)

[Daraa2/styles/night.json](http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/night.json) [http://maps.ecere.com/geoapi/collections/vtp/Daraa2/styles/night.json] (night style, Mapbox GL encoding)

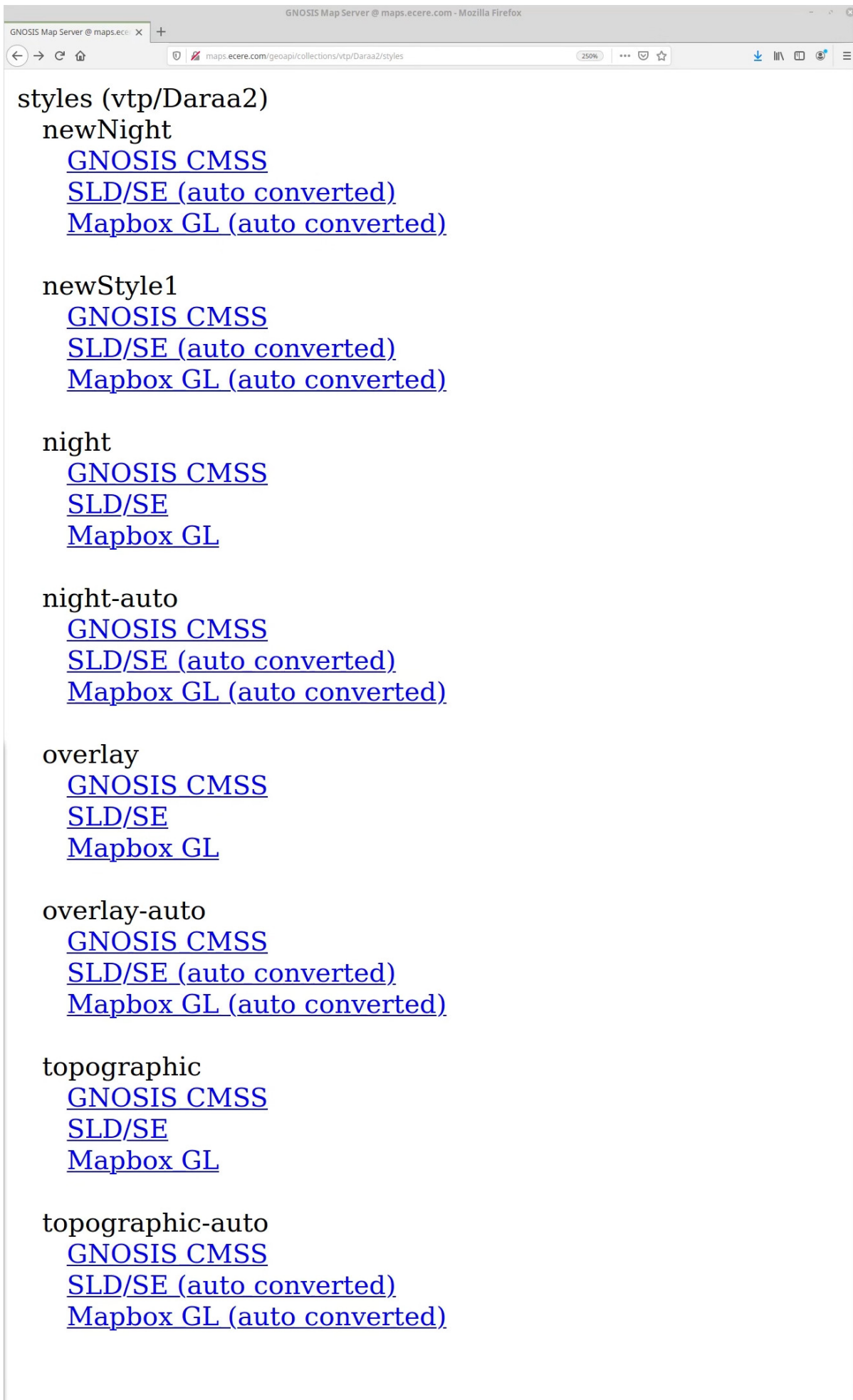


Figure 58. Sample basic HTML representation of styles for Daraa collections



Figure 59. 3D perspective view of Daraa vector layers layers rendered in topographic style over imagery in GNOSIS Cartographer

Example of styles for other data sets:

Styles for Natural Earth:

<http://maps.ecere.com/geoapi/collections/>

[NaturalEarth/styles](http://maps.ecere.com/geoapi/collections/NaturalEarth/styles) [<http://maps.ecere.com/geoapi/collections/NaturalEarth/styles>] (list of all styles for Natural Earth)

[NaturalEarth/styles/default](http://maps.ecere.com/geoapi/collections/NaturalEarth/styles/default) [<http://maps.ecere.com/geoapi/collections/NaturalEarth/styles/default>] (Default Natural Earth style, default CMSS encoding)

[NaturalEarth/styles/default.sld](http://maps.ecere.com/geoapi/collections/NaturalEarth/styles/default.sld) [<http://maps.ecere.com/geoapi/collections/NaturalEarth/styles/default.sld>] (Default Natural Earth style, SLD/SE encoding)

[NaturalEarth/styles/default.json](http://maps.ecere.com/geoapi/collections/NaturalEarth/styles/default.json) [<http://maps.ecere.com/geoapi/collections/NaturalEarth/styles/default.json>] (Default Natural Earth style, Mapbox GL encoding)

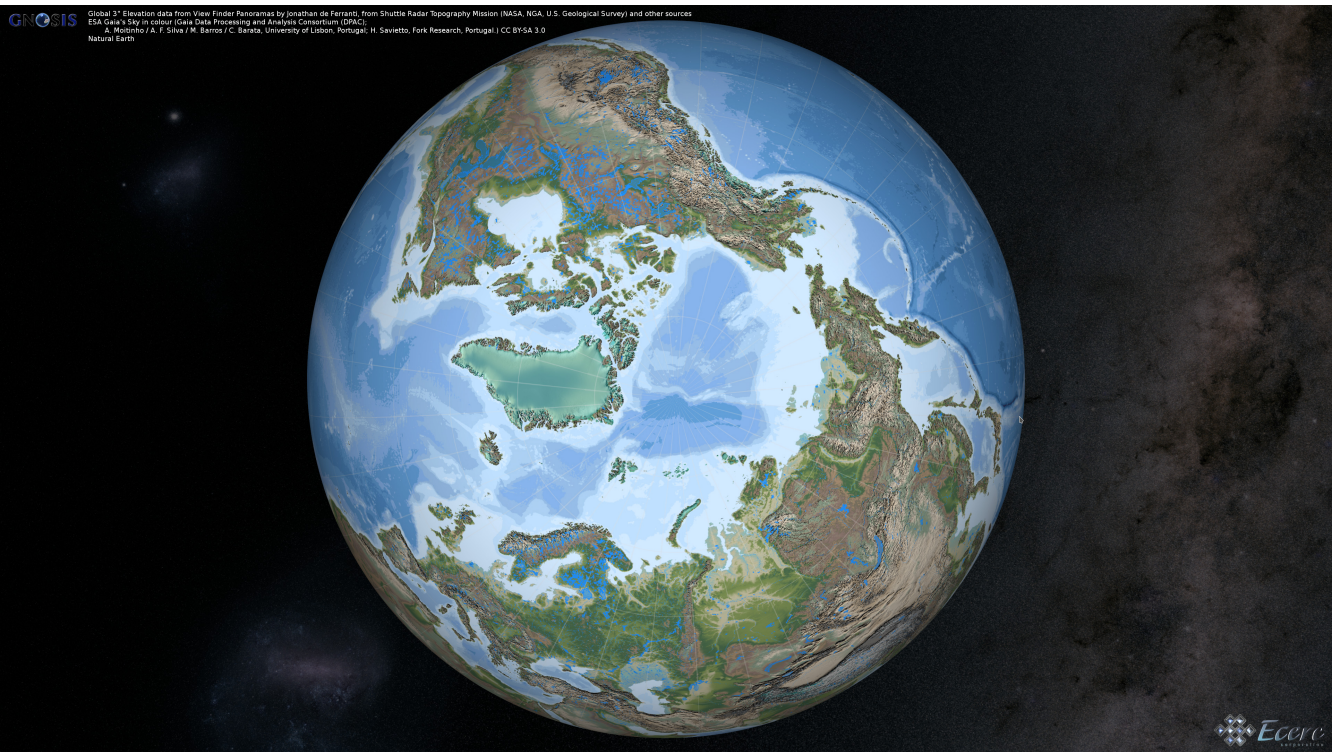


Figure 60. *Styled Natural Earth layers and global elevation data*

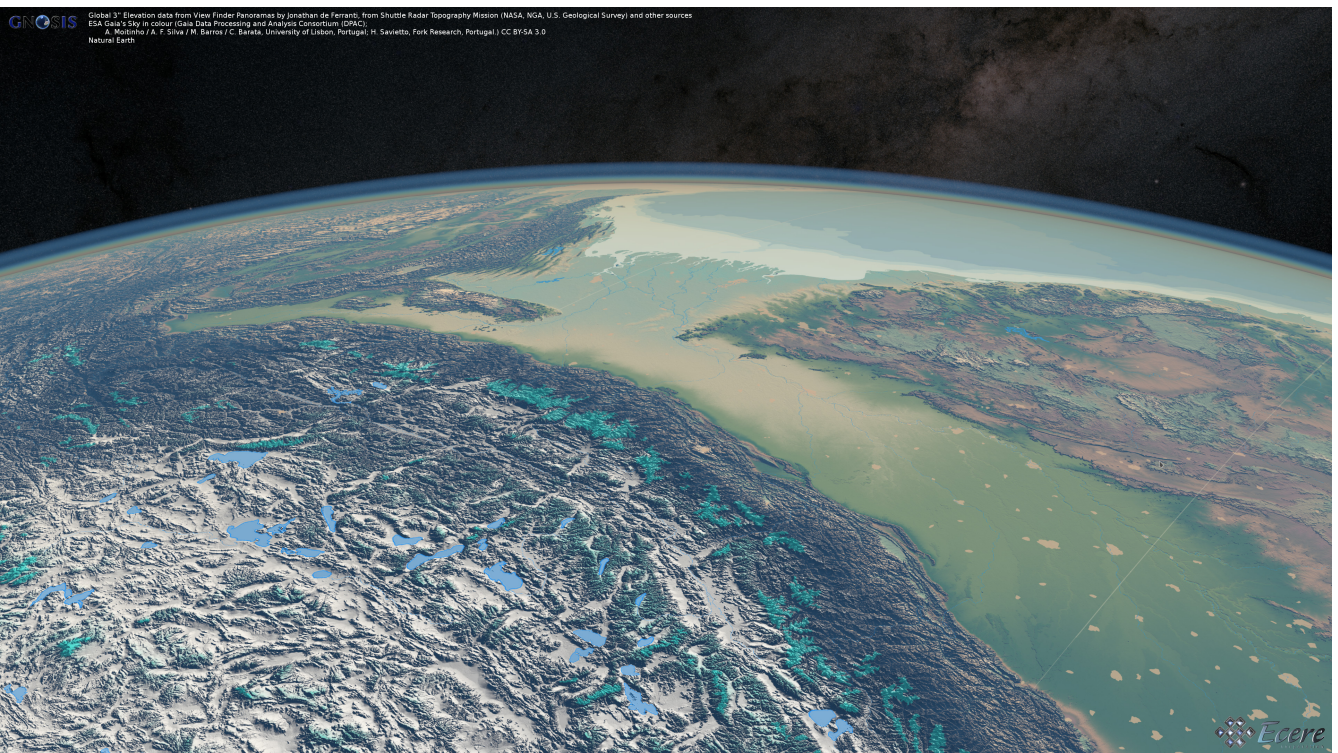


Figure 61. *Styled Natural Earth layers and global elevation data*



Figure 62. *Styled Natural Earth layers and global elevation data*

Styles for OpenStreetMap:

[osm/styles](http://maps.ecere.com/geoapi/collections/osm/styles) [http://maps.ecere.com/geoapi/collections/osm/styles] (list of all styles for OpenStreetMap)

[osm/styles/default](http://maps.ecere.com/geoapi/collections/osm/styles/default) [http://maps.ecere.com/geoapi/collections/osm/styles/default] (Default style for OpenStreetMap, default CMSS encoding)

[osm/styles/default.sld](http://maps.ecere.com/geoapi/collections/osm/styles/default.sld) [http://maps.ecere.com/geoapi/collections/osm/styles/default.sld] (Default style for OpenStreetMap, SLD/SE encoding)

[osm/styles/default.json](http://maps.ecere.com/geoapi/collections/osm/styles/default.json) [http://maps.ecere.com/geoapi/collections/osm/styles/default.json] (Default style for OpenStreetMap, Mapbox GL encoding)

[osm/styles](http://maps.ecere.com/geoapi/collections/osm/styles) [http://maps.ecere.com/geoapi/collections/osm/styles] (list of all styles for OpenStreetMap)

[osm/styles/default](http://maps.ecere.com/geoapi/collections/osm/styles/default) [http://maps.ecere.com/geoapi/collections/osm/styles/default] (Default style for OpenStreetMap, default CMSS encoding)

[osm/styles/default.sld](http://maps.ecere.com/geoapi/collections/osm/styles/default.sld) [http://maps.ecere.com/geoapi/collections/osm/styles/default.sld] (Default style for OpenStreetMap, SLD/SE encoding)

[osm/styles/default.json](http://maps.ecere.com/geoapi/collections/osm/styles/default.json) [http://maps.ecere.com/geoapi/collections/osm/styles/default.json] (Default style for OpenStreetMap, Mapbox GL encoding)

Full-color styles for Ordnance Survey OpenMapLocal layers:

<http://maps.ecere.com/geoapi/collections/OpenMapLocal/Road/styles>

<http://maps.ecere.com/geoapi/collections/OpenMapLocal/RailwayTrack/styles>

This list only highlights some examples out of many other available layers.

A.4.2. GeoPackage producer

The GeoPackage exporter was adapted to work with the latest version of GNOSIS Cartographer and GNOSIS SDK. Support for associating data layers and styles was implemented through a shared *Stylable Layer Set* using the new semantic annotations extension. Support for exporting and visualizing GeoPackage data layers using the tiled 2D gridded coverages extension was implemented. Work towards supporting the Well Known Binary for non-tiled vector features was also performed.

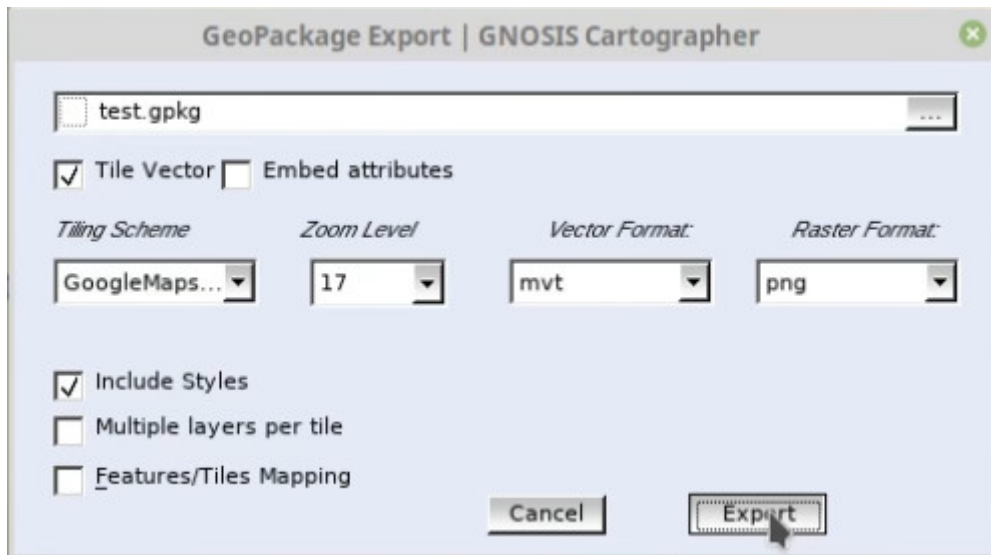


Figure 63. *GeoPackage exporting options in GNOSIS Cartographer*

The vector data is tiled using internal functionality from the GNOSIS SDK, and can be sourced from any data store that supports this functionality. For the experiment, a GeoPackage was generated from a GNOSIS data store of the Daraa2 OSM/TDS data set, including multiple styles and symbology for the points of interest.

A.4.3. Client components

GNOSIS Cartographer was used to demonstrate the client capabilities. GNOSIS Cartographer leverages the cross-platform GNOSIS Software Development Kit which directly makes use of OpenGL, OpenGL ES or WebGL. This utilizes the acceleration of Graphical Processing Units when deploying to a desktop, mobile or web platform. It supports 3D views and client-side rendering.

GNOSIS Cartographer had to be ported to a new and complete re-write of the GNOSIS SDK, which itself saw a significant amount of continued development. Improvements to the new version featured a completely redesigned styling engine. The objective was to support a fully dynamic styling system where complex expressions are valid for any symbolizer value, and better management of requests to fetch numerous remote resources, such as tiles.

Support was implemented for applying two distinct colors to monochrome icons (used for symbolizing points of interest).

GeoPackage client

The GeoPackage client was tested with GeoPackages produced by Ecere, Compusult and Image Matters.

It associates data layers with styles based on a shared *Stylable Layer Set* semantic annotation.

Applications written with the GNOSIS SDK present and style data layers exactly the same way, regardless of the type of data store (e.g. GNOSIS data store, GeoPackage or OGC API).



Figure 64. Visualizing Image Matters GeoPackage in GNOSIS Cartographer

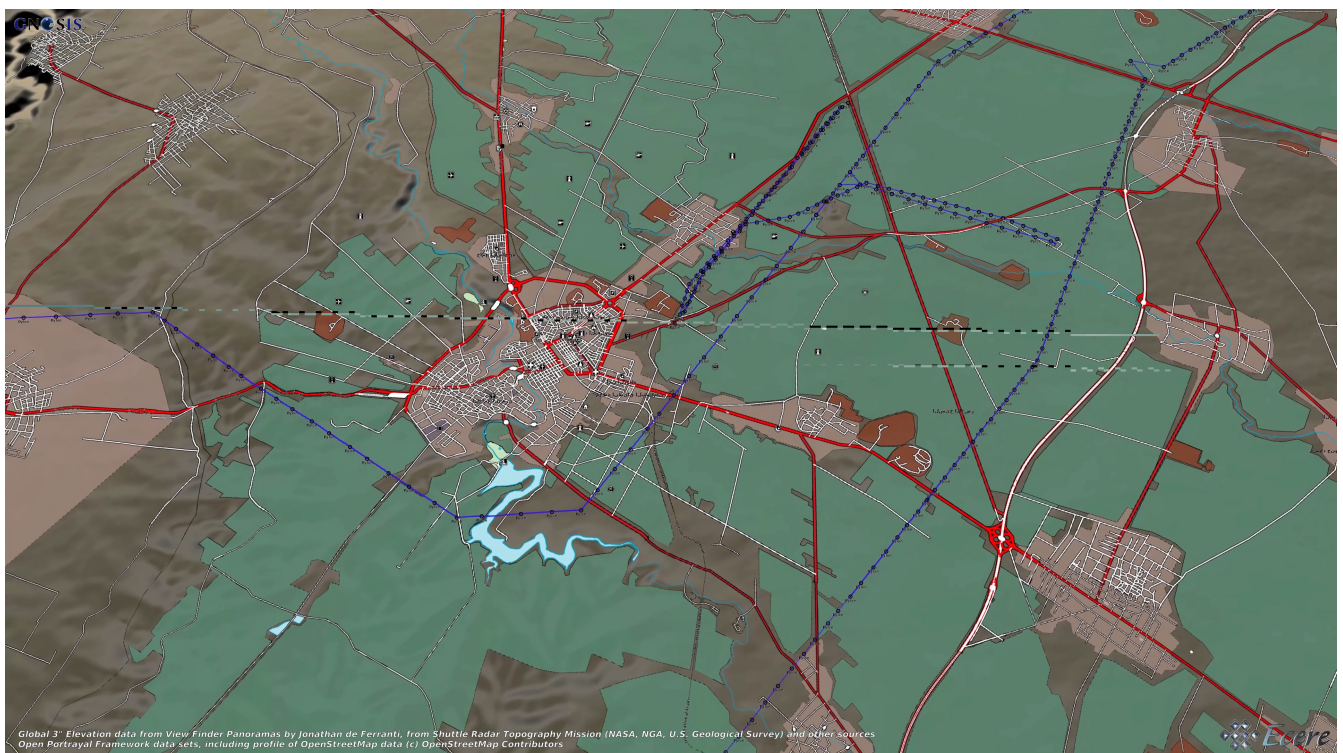


Figure 65. Visualizing CompuSult GeoPackage in GNOSIS Cartographer

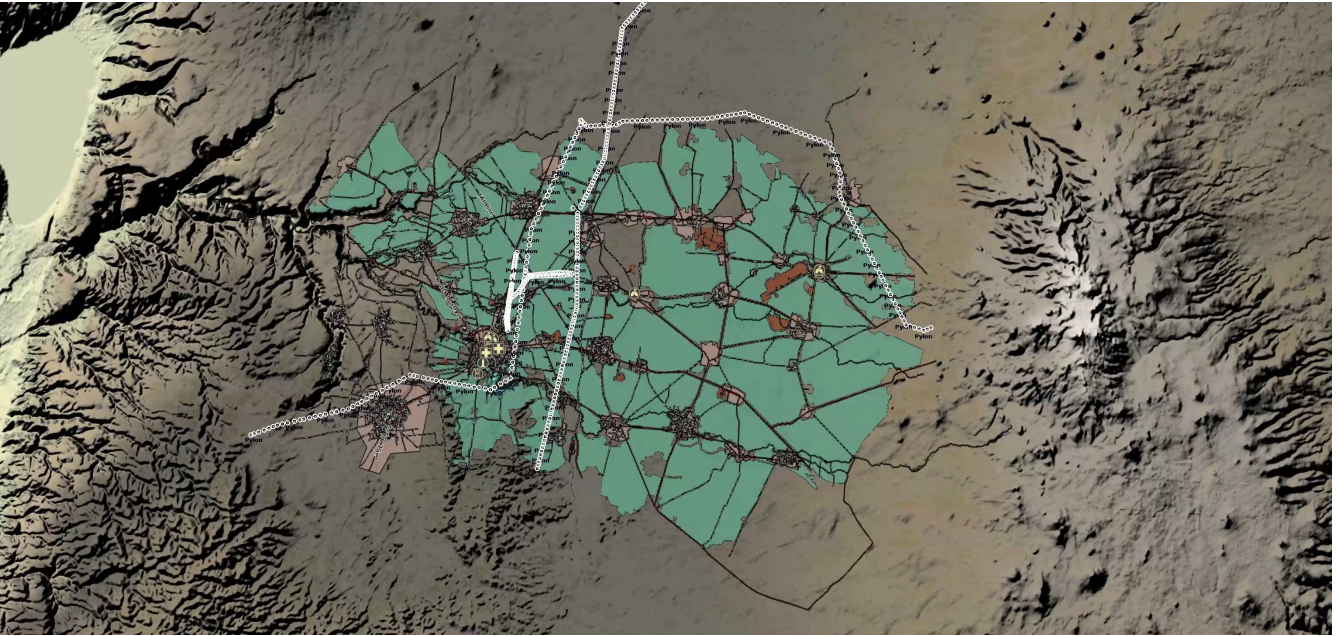


Figure 66. Visualizing Ecere GeoPackage in GNOSIS Cartographer

Mobile OGC API client

For the mobile client, a simple Android application was built with the Ecere GUI toolkit and GNOSIS SDKs. This Android application was used to demonstrate the visualization of data retrieved from OGC APIs and client-side styling.

The mobile OGC API client was tested with the Ecere and GeoSolutions end-points.

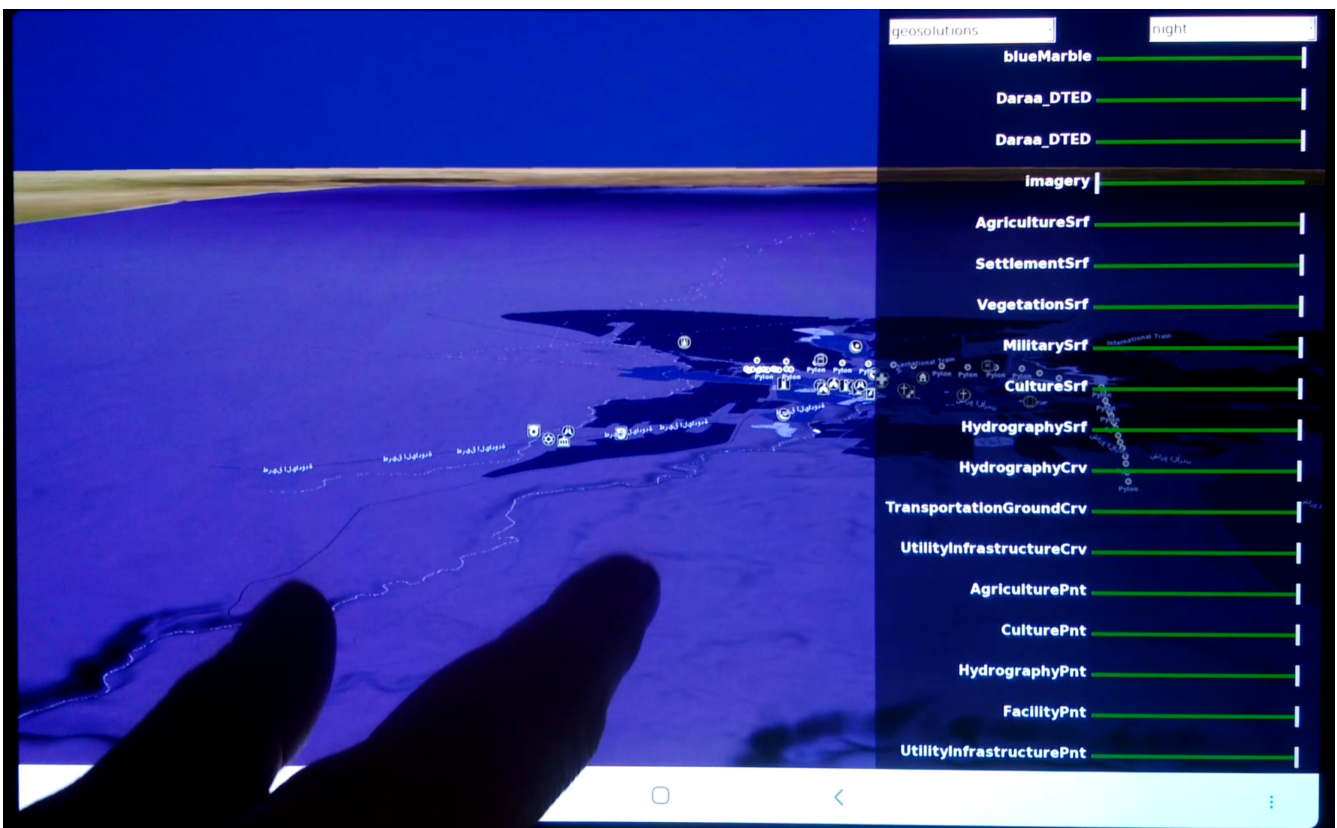


Figure 67. Accessing GeoSolutions Features, Tiles and Styles API in mobile GNOSIS client for Android



Figure 68. Accessing Ecere Features, Tiles and Styles API in mobile GNOSIS client for Android

The interactive instruments OGC API was also tested using the desktop client.

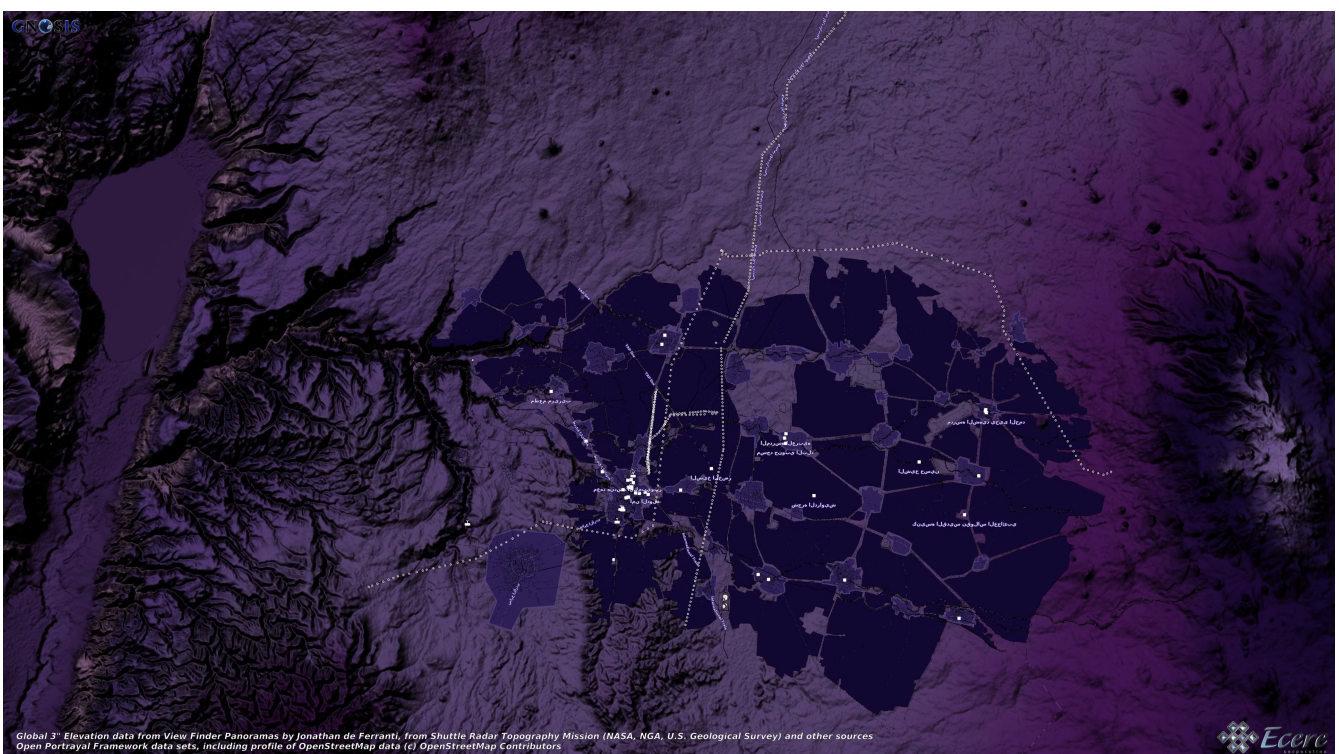


Figure 69. Accessing interactive instruments Features, Tiles and Styles API in GNOSIS Cartographer



Figure 70. Accessing Ecere Features, Tiles and Styles API in GNOSIS Cartographer

The Tiles and Styles APIs were exercised in the Technology Integration Experiments.

Visual Style Editor

A major effort for the initiative was to re-write the style editor within GNOSIS Cartographer to work with the GNOSIS CMSS styling model, which is fundamentally different and more flexible than the previous model.

Additionally, support for importing from SLD/SE and Mapbox GL styles to CMSS, and exporting from CMSS to both SLD/SE and Mapbox GL styles had to be developed. This functionality is also leveraged by the GNOSIS Map Server to provide on-the-fly styles translation.

The styles editor can directly display data from a local GNOSIS data store or GeoPackage (offline), or a remote OGC API (online). It can also list, retrieve, update, add, and delete styles from these data stores. Styles modifications are being reflected in real time in the 3D geospatial view.

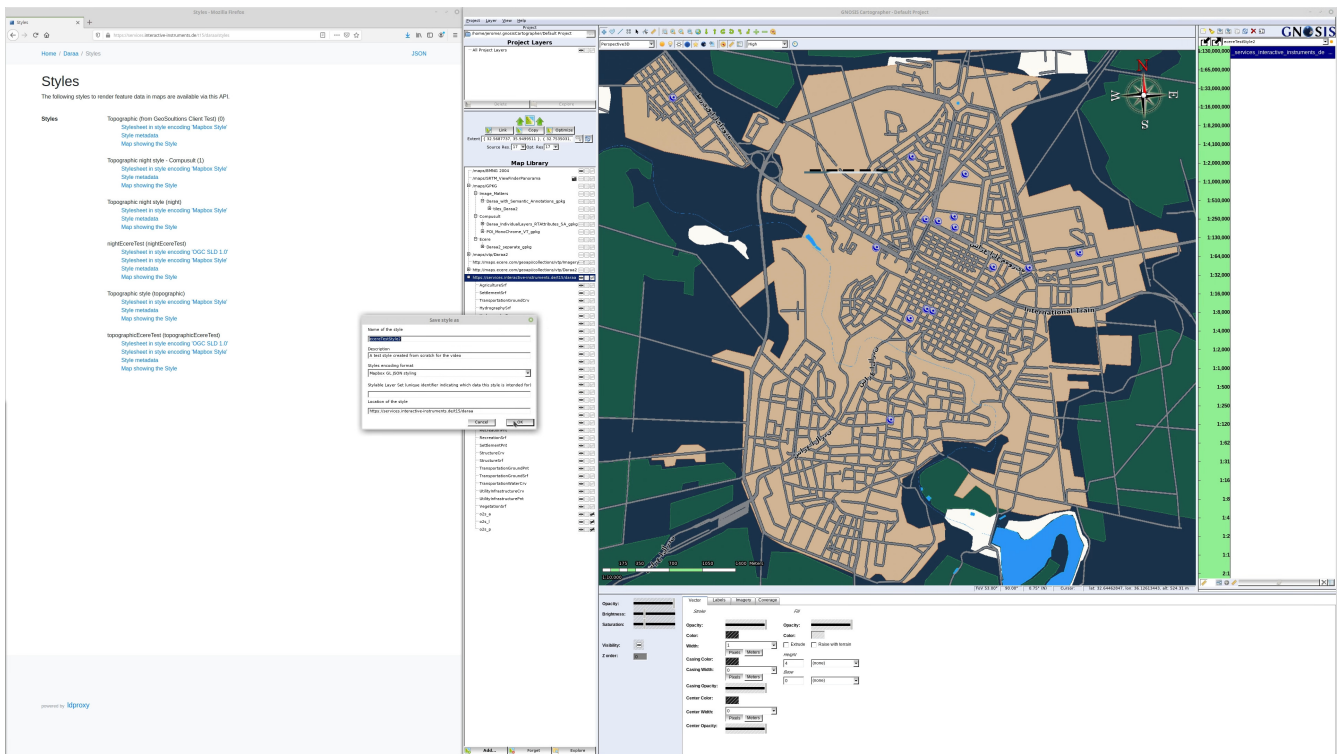


Figure 71. Accessing the interactive instruments Styles API to directly edit styles stored remotely

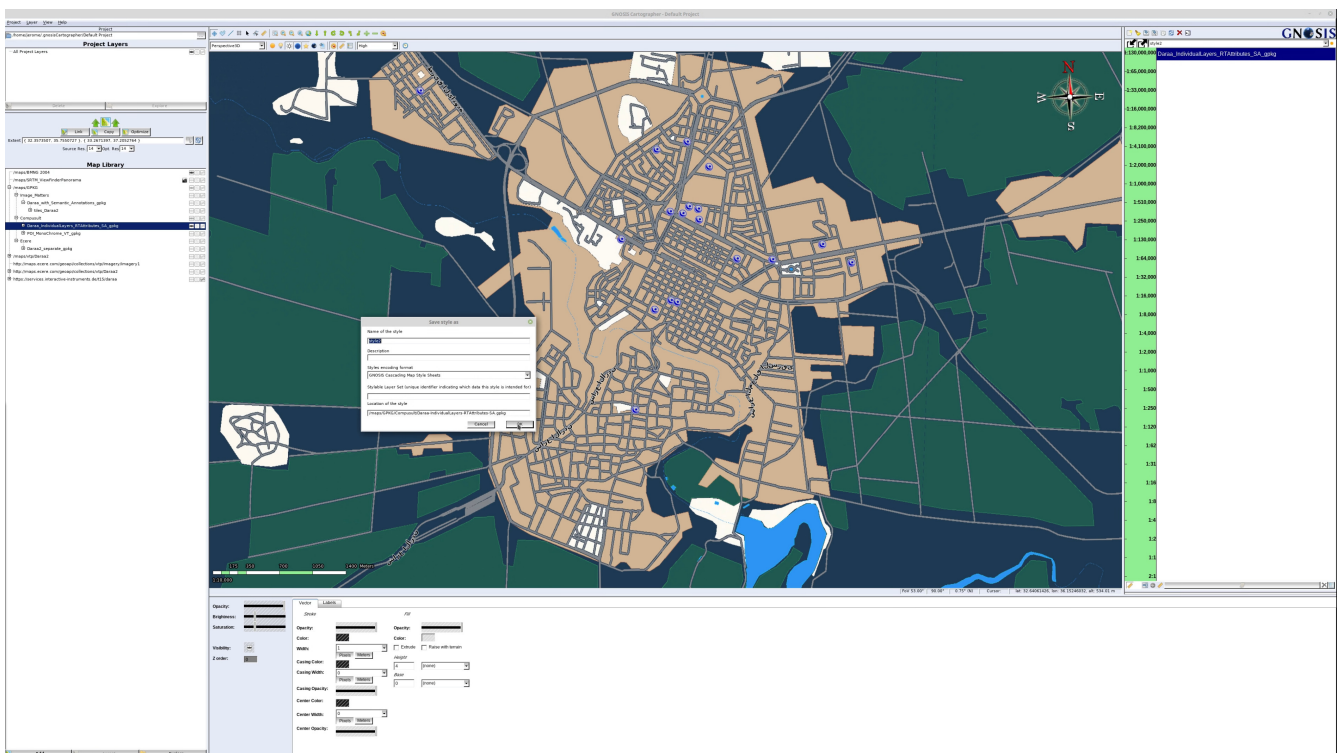


Figure 72. Editing styles inside a GeoPackage using the same interface

The layers panel provides a hierarchical list of data layers available for visualization. Data stores can be added to the *library layers*, which can in turn be converted to a GNOSIS data store or linked to be used as *project layers*. The rules panel presents a list of all styling rules pertaining to a selected data layer.

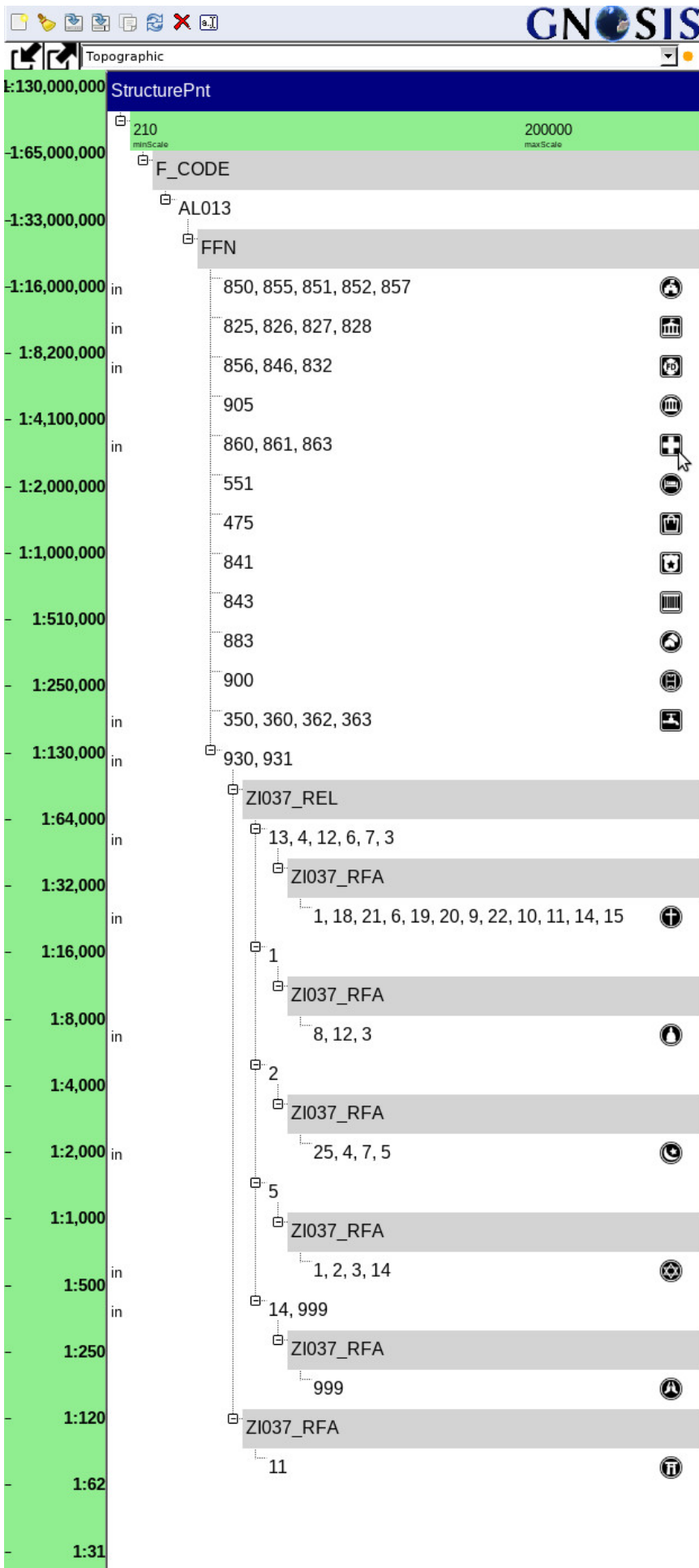


Figure 73. GNOSIS Cartographer's rule panel presenting a summarized view of all styling rules applied to the structure points layer

The styling panel allows to define the symbolizer specific to the selected styling rule.

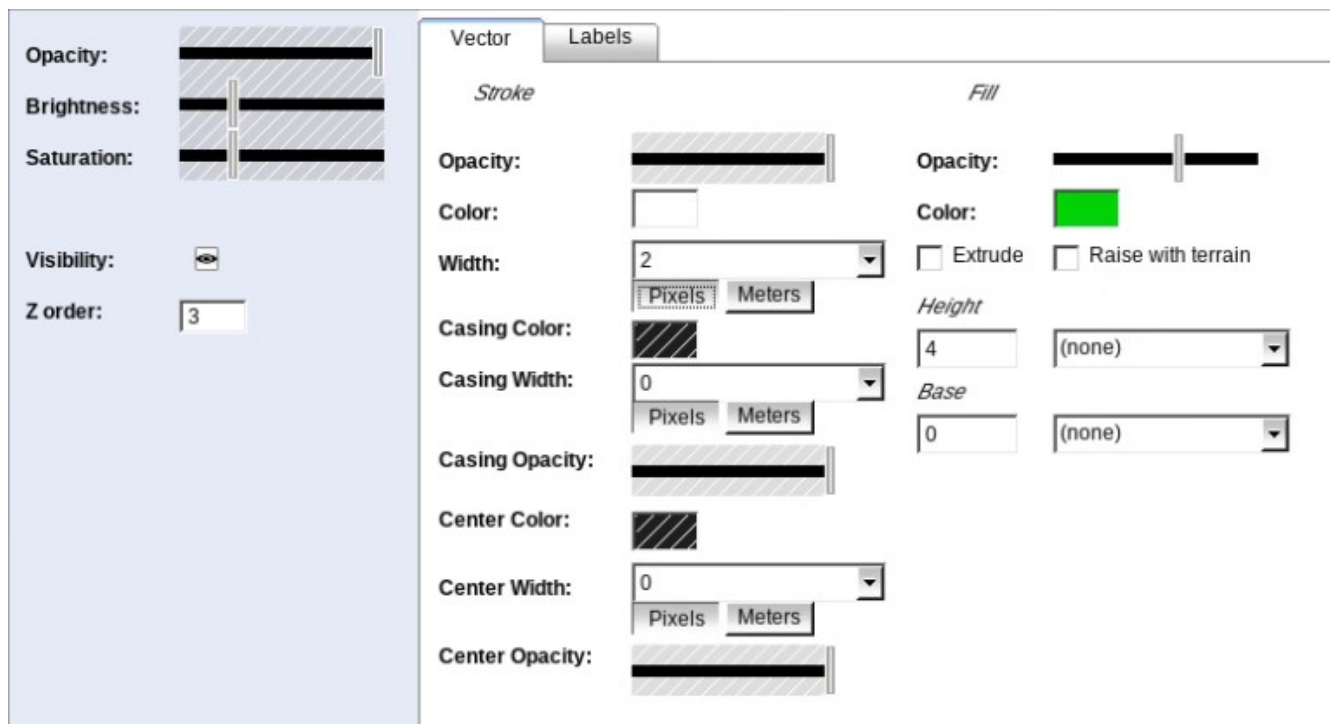


Figure 74. GNOSIS Cartographer's styling panel for symbolizing polygon geometry

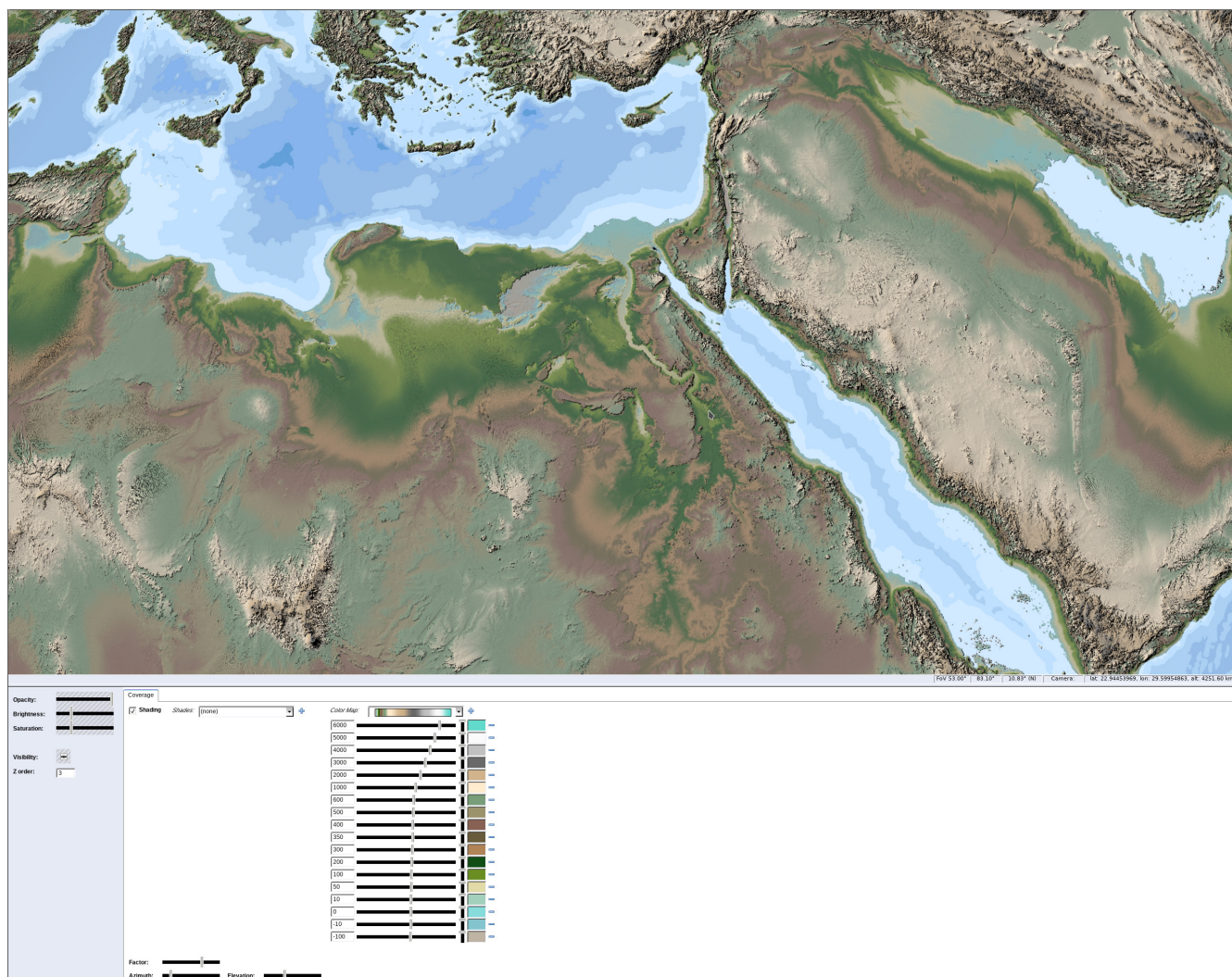


Figure 75. GNOSIS Cartographer's styling panel for symbolizing elevation data

A styles inspector panel is also being developed to assist the user in understanding how styles end up being applied and in helping to design their styles. Support was also implemented for editing more advanced styling aspects, such as scale-based selectors, complex expressions and expression-based symbolizer values. Further improvements to the functionality and useability of the styles editor are planned.

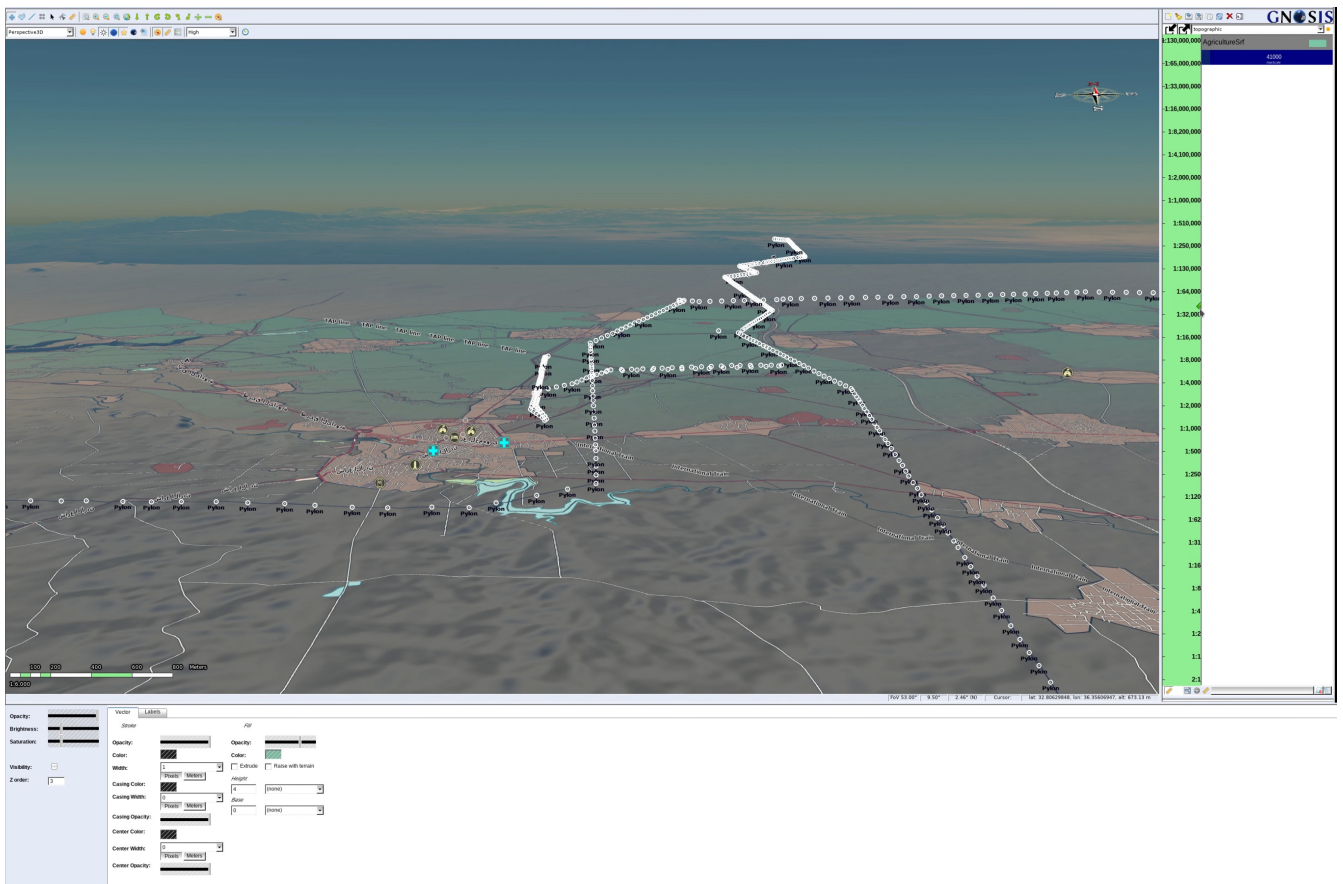


Figure 76. Scale selector allows to define scale-dependent rules



Figure 77. GNOSIS Cartographer's styling rule selector editor

Sample CMSS style created with the Visual Style Editor

```
#Daraa_DTED
{
  hillShading =
  {
    factor = 5.5,
    sun = { azimuth = 155, elevation = 37.5 }
  };
  colorMap = [
    { 0.0, 0x608849 },
    { 900.0, 0xE2DBA7 },
    { 1300.0, 0xFCC575 },
    { 1900.0, 0xFEAA86 },
    { 2500.0, 0xFAFAFA }
  ];
}

#AgricultureSrf
{
  fill = { color = green, opacity = 0.75 };
  stroke = { color = forestGreen, width = 2.0 };
}
```

```

#SettlementSrf
{
  fill = { color = tan };
  stroke = { color = peru, width = 2.0 };
}

#TransportationGroundCrv
{
  stroke = { color = gray, width = Meters { 33.0 }, casing = { color = black, width =
1.0 };
  [viz.sd > 55000]
  {
    stroke.casing.width = 0;
  }

  [!(ZI005_FNA = 'No Information')]
  {
    label =
    {
      elements = [
        Text {
          text = ZI005_FNA, alignment = { center, middle },
          font =
          {
            face = 'Arial Unicode MS', size = 14.0, bold = true,
            color = black, outline = { size = 2.0, color = white, opacity =
1.0 }
          }
        }
      ];
    }
  }
}

#HydrographySrf
{
  fill = { color = dodgerBlue };
  stroke = { color = deepSkyBlue, width = 2.0 };
}

#HydrographyCrv
{
  stroke = { color = dodgerBlue };
}

#StructurePnt
{
  [F_CODE = 'AL013']
  {
    label =

```

```

{
  elements = [
    Image { image = { path = 'Church_Islamic.png' },
      tint = mediumBlue, blackTint = silver, scaling = 0.8 }
  ];
};
}

```



Figure 78. Sample rendering of the above style in GNOSIS Cartographer

A.5. Implementation CubeWerx

A.5.1. Available services and layers

CubeWerx Inc. set up a server for Testbed-15 at <https://tb15.cubewerx.com/cubewerx/cubeserv/op>. This server implements the following classic OGC web services:

Table 11. OGC web services provided by CubeWerx server

service type	service versions	URL of capabilities document
WMTS	1.0.0, 1.1.0, 1.1.1, 1.3.0	https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/WMTSCapabilities.xml
WMS	1.0.0, 1.1.0, 1.1.1, 1.3.0	https://tb15.cubewerx.com/cubewerx/cubeserv/op?SERVICE=WMS&VERSION=1.3.0&REQUEST=GetCapabilities
WCS	1.0.0, 1.1.1, 1.1.2, 2.0.1	https://tb15.cubewerx.com/cubewerx/cubeserv/op?DATASTORE=Daraa&SERVICE=WCS&REQUEST=GetCapabilities&ACCEPTVERSIONS=2.0.1
WFS	1.0.0, 1.1.0, 1.1.1, 1.1.3, 2.0.0, 2.0.2, 2.5.0, 3.0.0	https://tb15.cubewerx.com/cubewerx/cubeserv/op?datastore=Daraa&SERVICE=WFS&REQUEST=GetCapabilities&ACCEPTVERSIONS=1.1.0

These services serve the following layers:

Table 12. Layers served by CubeWerx server

layer ID	Style options	TileMatrixSet options	Format options	InfoFormat options	available through
Daraa_mosaic_2019	default	smmerc, wgs84, 3395	jop, jpg, png	gml, json, html	WMTS, WMS, WCS
Daraa_vectors	Topographic, Overlay, Night	smmerc, wgs84, 3395	jop, jpg, png, mvt, geojson	gml, json, html	WMTS, WMS, WFS

NOTE

The jop format stands for "JPEG or PNG". It will serve either a JPEG or a PNG on a tile-by-tile basis, optimizing tile size while still providing transparency where needed.

A.5.2. Tiles API endpoints

Individual tiles for the layers listed above can be requested through the following URL template:

```
https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/tiles/{Layer}/{Style}/{TileMatrixSet}/{TileMatrix}/{TileRow}/{TileCol}.{Format}
```

Both, tiled raster data and (where applicable) tiled vector data are supported. These endpoints also support style-defined background colors and the TRANSPARENT parameter as described in the [Background Color](#) chapter. For the Daraa_vectors layer, the default background color for the Night

style is #1E193E (a dark blue), and the default background color for the Topographic style is #CCCCCC (a light grey).

Information about the feature(s) at a specific pixel of a tile can be requested through the following URL template:

```
https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/featureInfo/{Layer}/{Style}/  
{TileMatrixSet}/{TileMatrix}/{TileRow}/{TileCol}/{J}/{I}.{InfoFormat}
```

A.5.3. Styles API endpoints

The set of endpoints representing full stylesets (i.e., all styles) of a layer (as an SLD document) is indicated by the following URL template:

```
https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/layers/{Layer}.sld
```

In addition to the HTTP GET method for retrieving the stylesets, HTTP PUT can be used to push a new styleset for a layer (completely replacing its previous style definitions), and HTTP DELETE can be used to delete the styleset of a layer (effectively removing the layer). HTTP HEAD and OPTIONS are also supported.

The set of endpoints representing individual styles of a layer (as an SLD document) is indicated by the following URL template:

```
https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/layers/{Layer}/{Style}.sld
```

In addition to the HTTP GET method for retrieving the style, HTTP PUT can be used to add a new style or update an existing style, and HTTP DELETE can be used to delete a style. HTTP HEAD and OPTIONS are also supported.

When a styleset is updated or a style is added, updated or removed, subsequent map and tile requests will reflect the change. However, it may take a few seconds for the change to take effect.

The following SLD versions are supported (for both GET and PUT): 1.0.0, 1.0.20, 1.1.0 and 1.1.20. To request a specific SLD version, supply an SLD_VERSION parameter with the GET request.

A.5.4. Image API endpoints

The set of source images (as a STAC Collection) comprising the Daraa_mosaic_2019 layer can be requested with the following URL:

[https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/collections/Daraa_mosaic_2019/
images](https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/collections/Daraa_mosaic_2019/images)

A new source image can be added to the layer by making an HTTP POST request to the above URL. The body of the POST request can be a [GeoTIFF](https://www.opengeospatial.org/standards/geotiff) [https://www.opengeospatial.org/standards/geotiff] image containing all necessary metadata or a ZIP file containing a source image bundled with any

auxiliary files that are necessary. The image must be of a compatible type.

The description of a single source image (as a [STAC](https://stacspec.org/) [https://stacspec.org/] Item) can be retrieved through the following URL template:

```
https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/collections/Daraa_mosaic_2019/images/{imageId}
```

Typically a client does not have to compose these URLs, since they are reported by the STAC Collection.

The STAC Item representing an image references the actual GeoTIFF image via a "main" asset, and any auxiliary files via "sidecar" assets. Additionally, a JPEG thumbnail of the image is made available through a "thumbnail" asset.

A source image can be deleted from the layer (and removed from the server) by making an HTTP DELETE request to its STAC-Item URL.

Each of these endpoints also supports the HTTP HEAD and OPTIONS request methods.

When a source image is added or removed, subsequent map, tile and coverage requests will reflect the change. However, it may take a few seconds for the change to take effect.

The Daraa_mosaic_2019 layer is initially populated with a subset of the images available at "ftp://ftp.opengis.org". The TIE can POST new images and/or DELETE existing ones from this starting point.

A.6. Implementation Compusult

A.6.1. Overview

Compusult provided a WEB API client for the Open Portrayal Framework scenario in Testbed-15 for a desktop/mobile devices. Compusult also provided a web-based GeoPackage producer.

The Web API client supports the following specifications:

- OGC API - Core, OpenAPI 3.0
- Tiles API - Tile Matrix Set, Tiled Data, Tiled Maps
- Styles API - Core, Use Styles, Manage styles, Manage resources, Fetch Resource, Mapbox Styles, SLD 1.0, SLD 1.1, Queryables, Style information
- Image API - Core, Images, Manage Images

The GeoPackage producer supports these following specifications:

- Data - Shapefile, File GeoDatabase, TIFF, DTED
- Styles - Mapbox Styles, SLD 1.0, SLD 1.1
- GeoPackage - Version 1.2

- GeoPackage Extensions - Related Tables, Simple Attributes, Semantic Annotations, Metadata, Tiled-Gridded-Elevation

The GeoPackage producer provides the following content:

- A test dataset derived from OpenStreetMap data in the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA. The dataset consists of 33 collections (feature types) in Mapbox/GeoJSON vector tile format.
- A test POI dataset derived from OpenStreetMap data in the region of Daraa, Syria, converted to the Topographic Data Store schema of NGA, in Mapbox/GeoJSON vector tile format and GeoPackage simple features.
- A test DTED in the region of Daraa, Syria, in GeoPackage tiled-gridded-elevation.
- Two topographic/night/overlay styles for the Daraa dataset using Mapbox Style and SLD 1.1 as the stylesheet encoding, derived from styles developed by GeoSolutions in the Vector Tiles Pilot.

A.6.2. Scenario Details

Step 1

The client supplies OpenAPI endpoints which support the new OGC APIs:

- CubeWerx : <https://tb15.cubewerx.com/cubewerx/cubeserv/op/wmts/1.0.0/WMTSCapabilities.xml>
- Ecere : <http://maps.ecere.com/geoapi/collections/vtp/Daraa2>
- Interactive Instruments : <https://services.interactive-instruments.de/t15/daraa/api/>
- GeoSolutions : <http://maps.geo-solutions.it/geoserver/ogc/styles>, <http://maps.geo-solutions.it/geoserver/ogc/tiles/>

The desktop/mobile application discovers the collections `/collections` (shown in [Figure 79](#)), and loads the referenced styles `/styles` (shown in [Figure 80](#)) using the Tiles and Styles API.

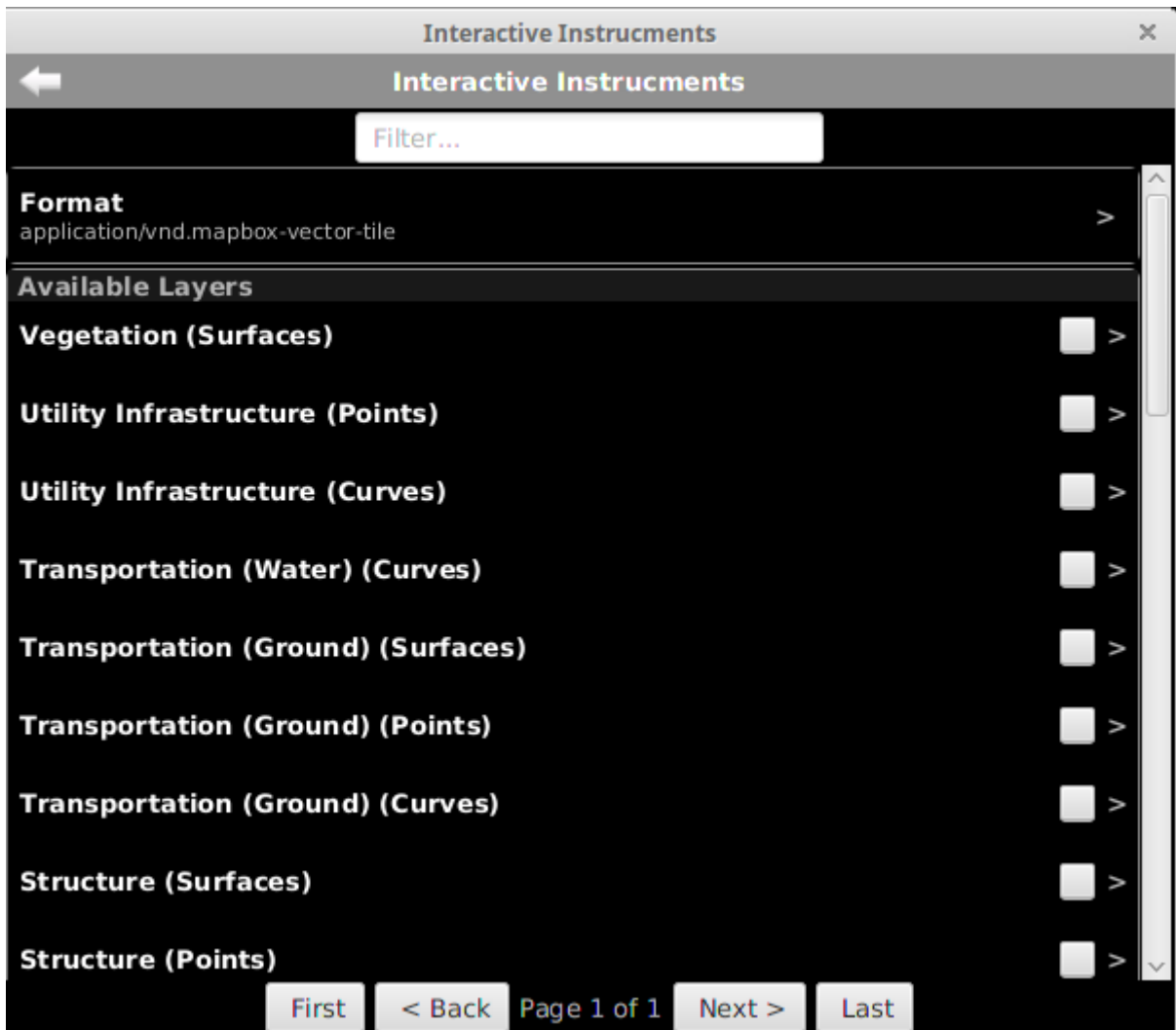


Figure 79. Discover OGC Common API Collections

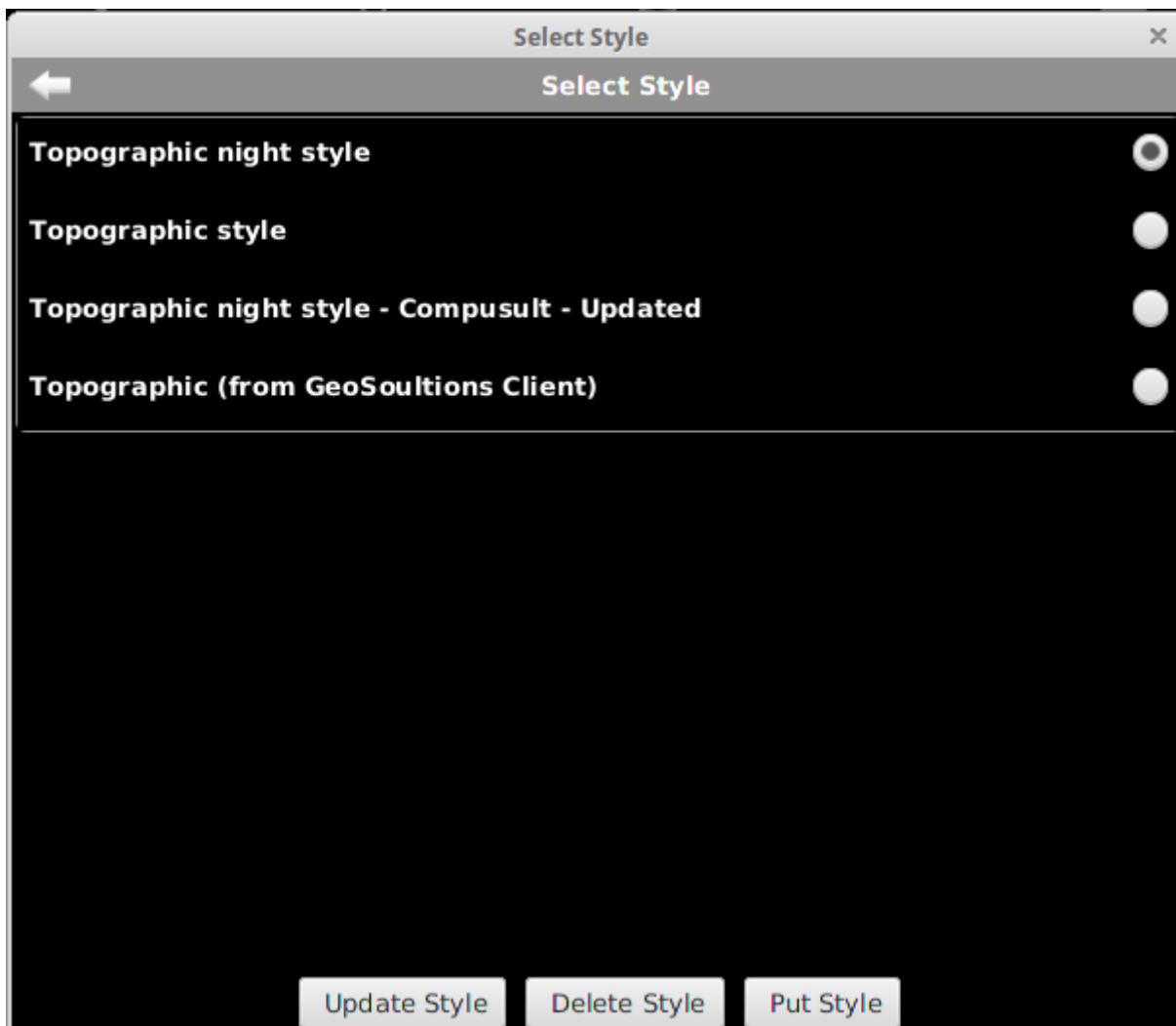


Figure 80. Discover OGC Styles API styles

Step 2

The client uses the `/tiles` and/or `/maps` collection resources to determine the available formats (shown in Figure 81), and requests the tiled-data or pre-rendered content in a specified format. If a raw format is selected, the chosen stylesheet is retrieved `/styles/{styleId}` and applied to the tiled-data (shown in Figure 82).

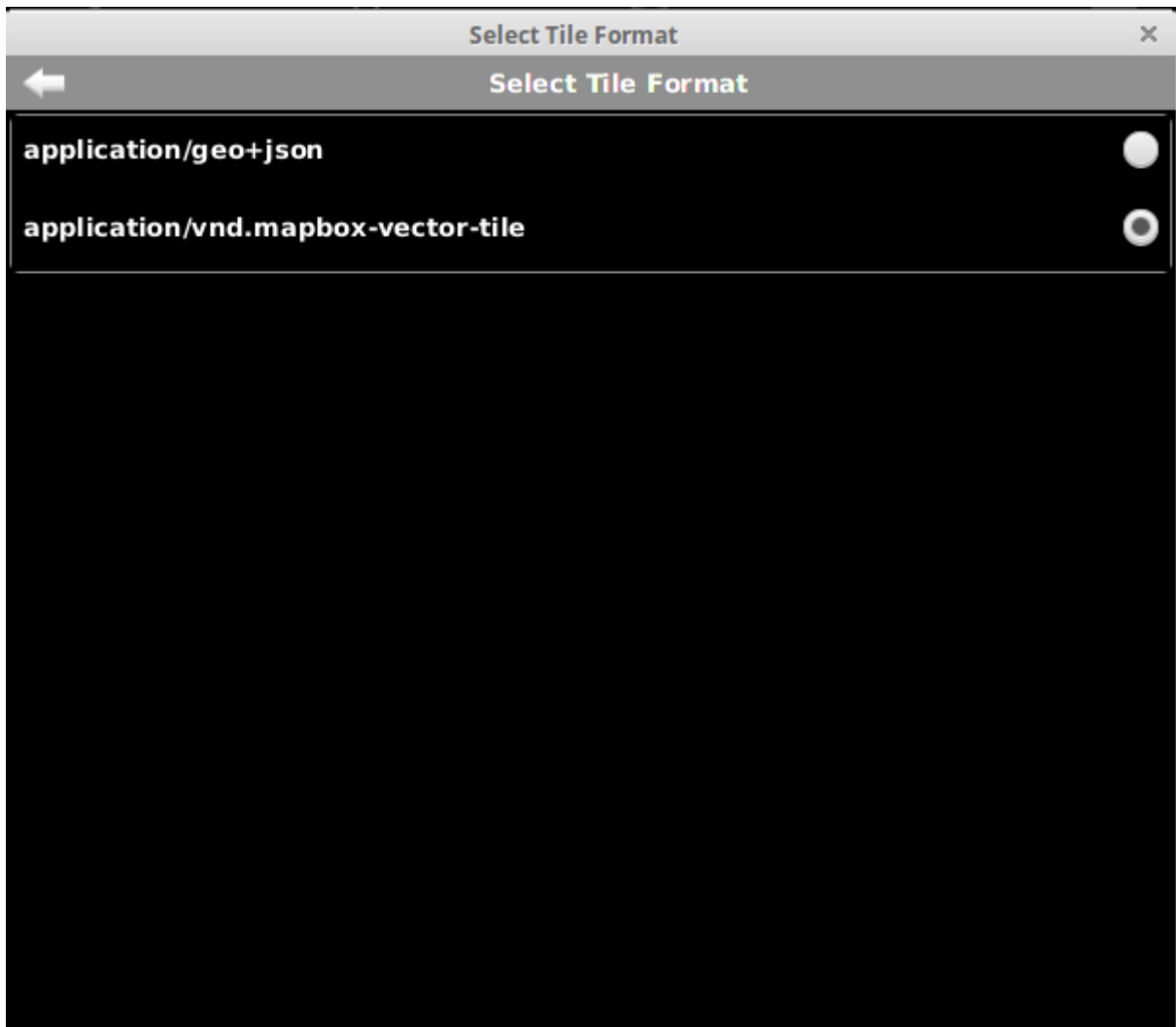


Figure 81. Discover collection formats from Tiles and Maps API

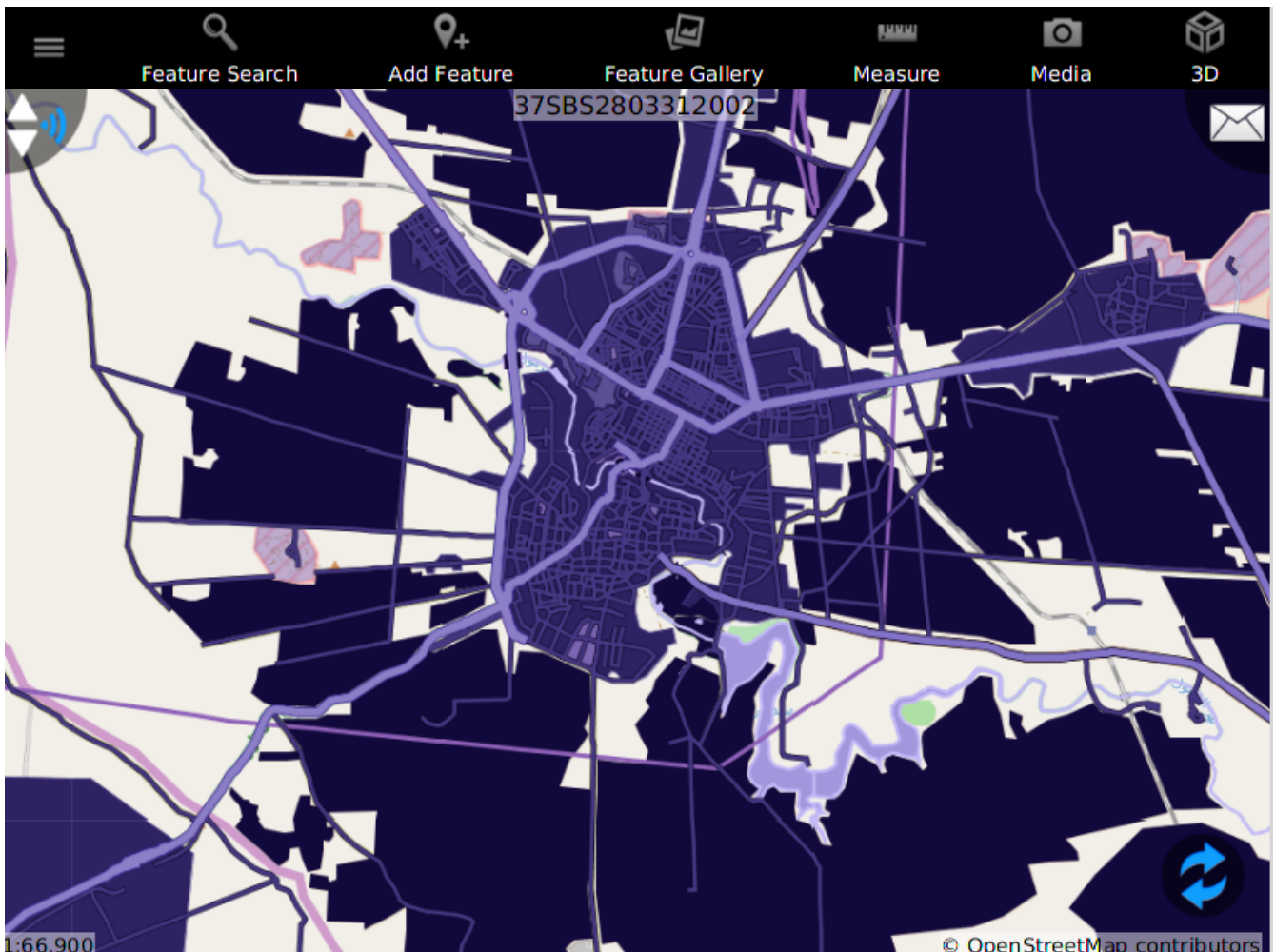


Figure 82. Render collection data in specified format(mapbox) using specified stylesheet(night)

Step 4

The client uses the `/styles`, `/styles/{styleId}` and `/styles/{styleId}/metadata` (show in Figure 83) endpoints which allows the user to access the Styles API to perform the following functions :

- View all styles (show in Figure 84)
- Add a new style (show in Figure 85)
- Update a style (show in Figure 86)
- Delete a style (show in Figure 86)
- Download a style (show in Figure 86)
- Preview a style (show in Figure 87)
- View meta-data for a style (show in Figure 88)
- Update meta-data for a style (show in Figure 86)

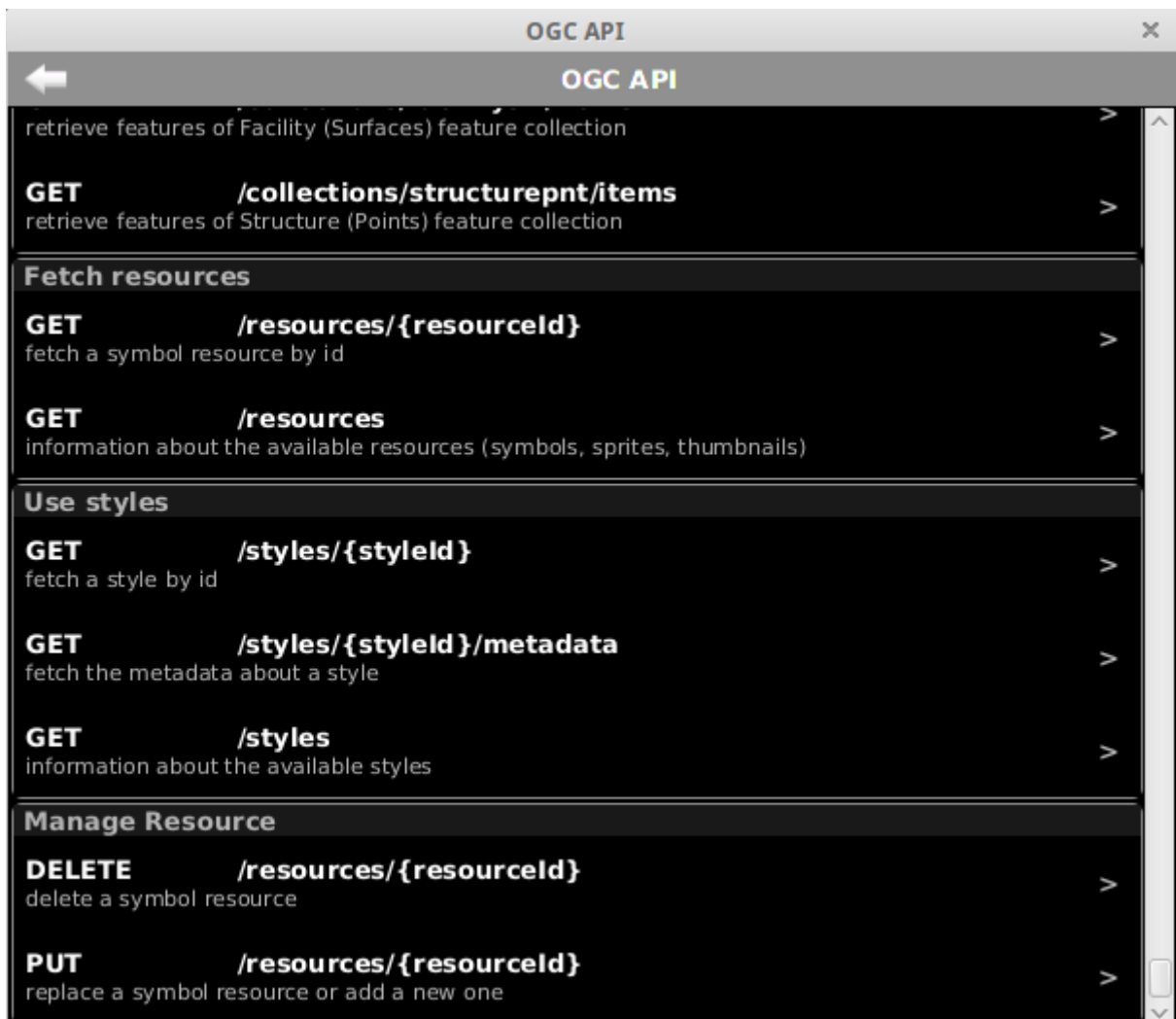


Figure 83. Access Style operations from Styles API

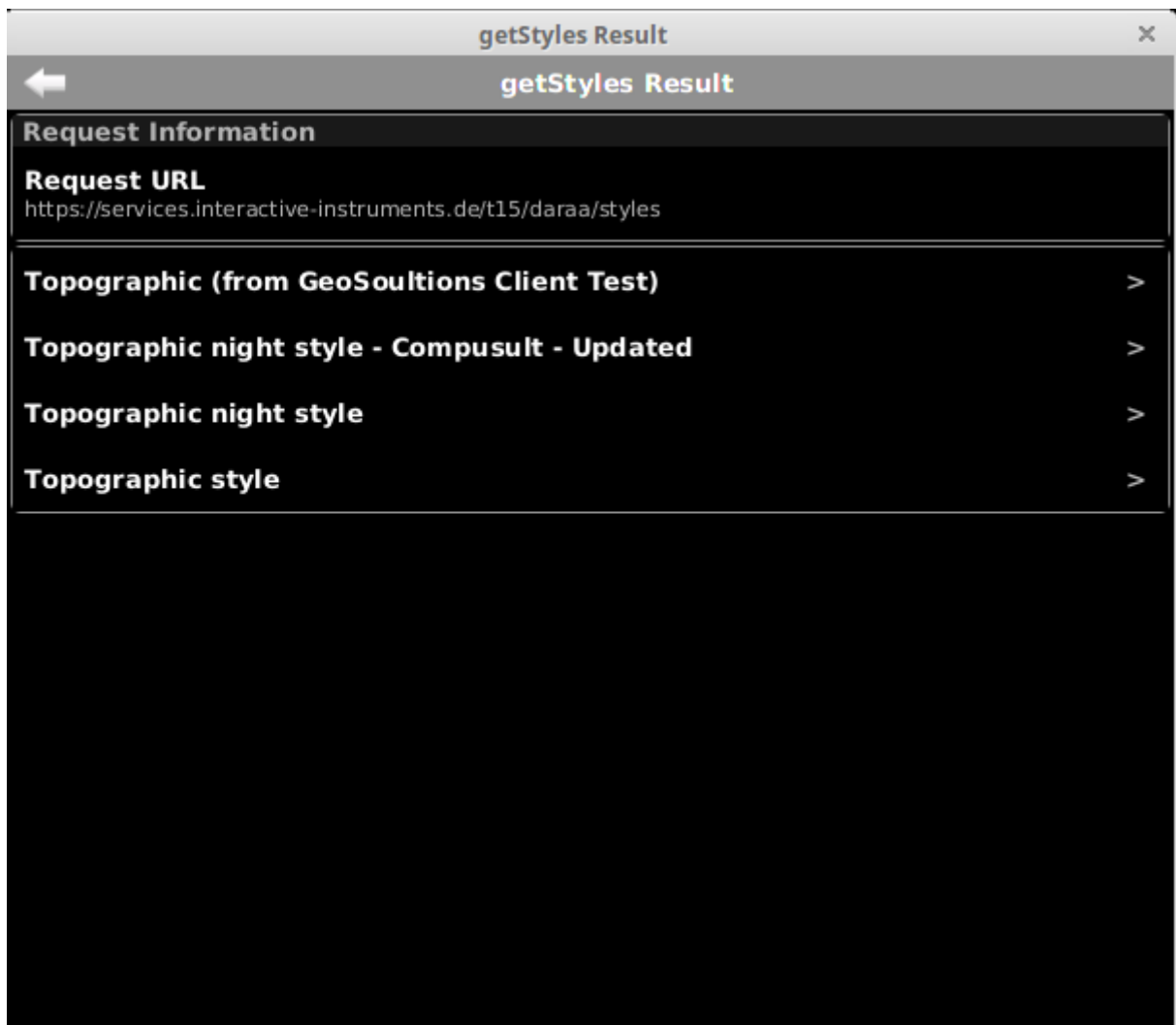


Figure 84. View available styles from Styles API

POST /styles

POST /styles Describe

Adds a style to the style repository.
If a new style is created, the following rules apply:

- * If the style submitted in the request body includes an identifier (this depends on the style encoding), that identifier will be used. If a style with that identifier already exists, an error is returned.
- * If no identifier can be determined from the submitted style, the server will assign a new identifier to the style.
- * A minimal style metadata resource is created at `/styles/{styleId}/metadata``. Please update the metadata using a PUT request to keep the style metadata consistent with the style definition.
- * The URI of the new style is returned in the header ``Location``.

Parameters

f - string (query)
 The format of the response. If no value is provided, the standard http rules apply, i.e., the accept header shall be used to determine the format. Pre-defined values are: 'json' (JSON); the response to other values is determined by the server.
 Available values : json

Request Body

application/vnd.mapbox.style+json
 Type : object

application/vnd.ogc.sld+xml;version=1.0
 Type : string

Execute Operation

Figure 85. Add a new style using Styles API



Figure 86. Available style operations for a style using Styles API

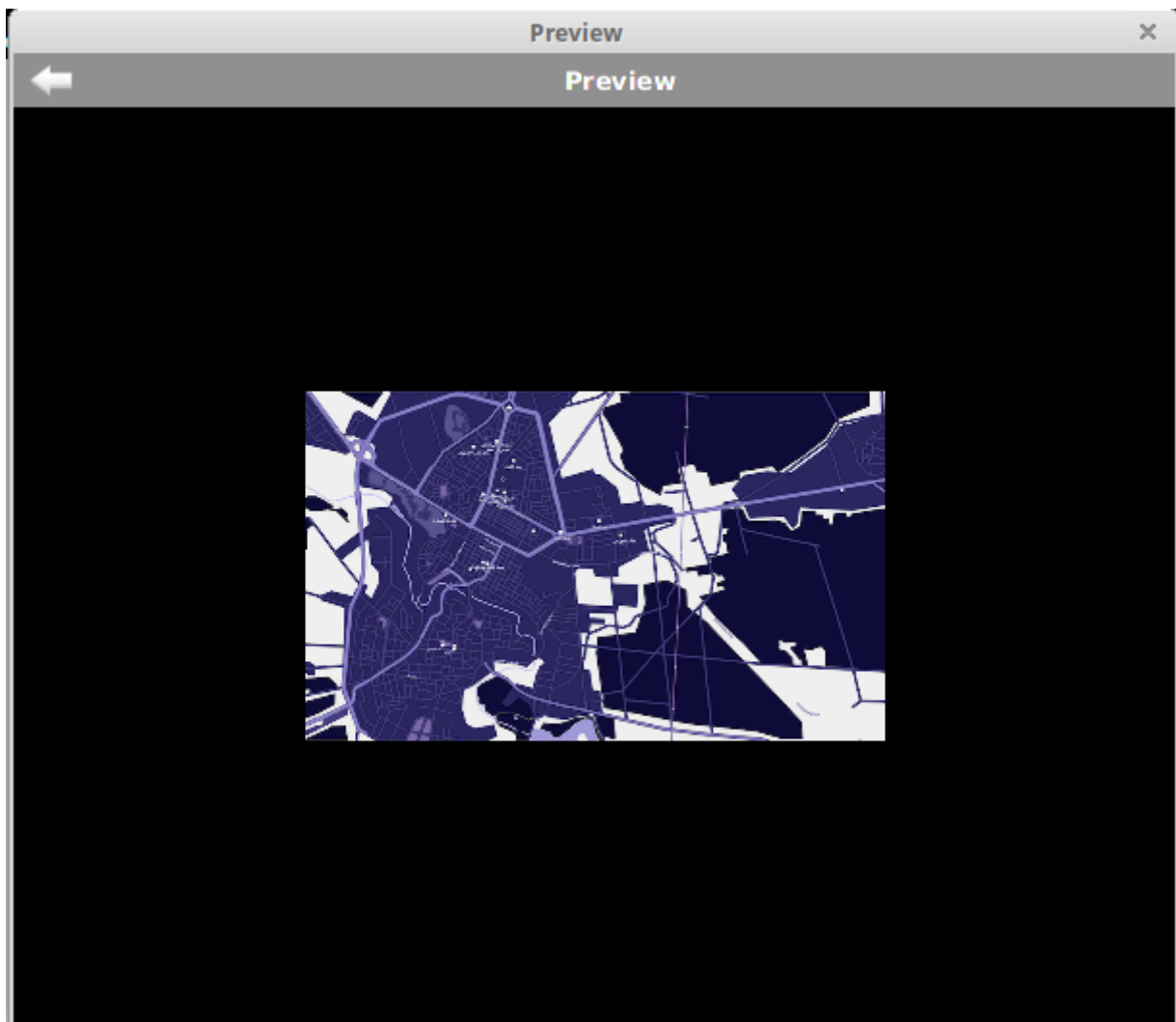


Figure 87. Preview a style using Styles API

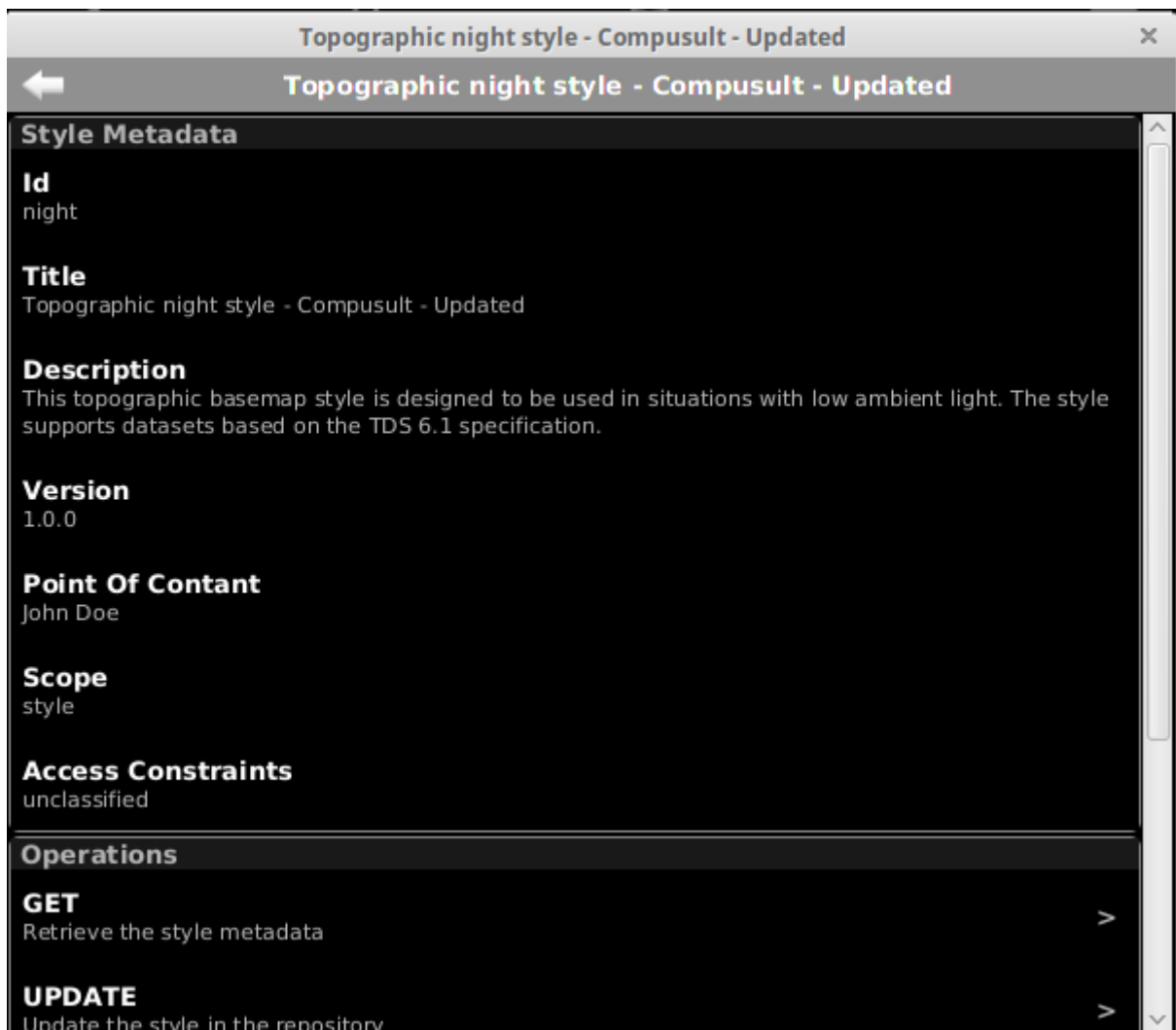


Figure 88. View style meta-data using Styles API

Step 6

Once a new style has been created, or a style has been updated the client will refresh the style content using the Styles API and update the content accordingly (show in [Figure 89](#)).

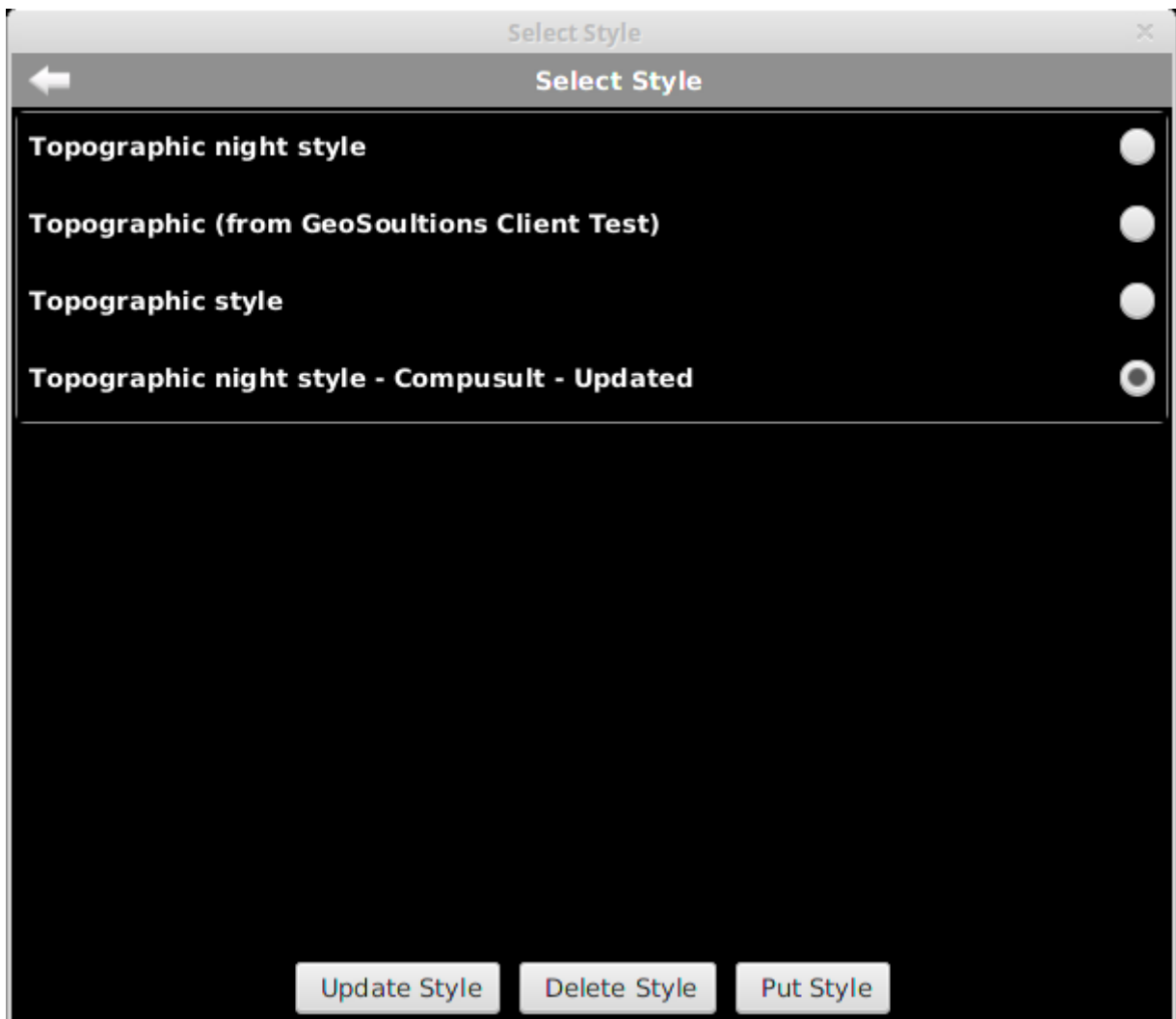


Figure 89. Newly added Compusult style using Styles API

Step 7

The client supports the Images API (shown in [Figure 90](#)) which allows for addition/modification/deletion of imagery content. Users are given the option to browse the image resources `/images` and perform the following operations:

- View all images (show in [Figure 91](#))
- Add a new image (show in [Figure 92](#))
- View image meta-data (show in [Figure 93](#))
- Modify an image (show in [Figure 93](#))
- Delete an image (show in [Figure 93](#))
- Download an image (show in [Figure 93](#))
- Preview an image (show in [Figure 94](#))

OGC API		
Default		
GET Get properties	/collections/{collectionId}/images/properties	>
DELETE Delete property	/collections/{collectionId}/images/properties/{propertyName}	>
GET Get property	/collections/{collectionId}/images/properties/{propertyName}	>
PUT Update property	/collections/{collectionId}/images/properties/{propertyName}	>
GET Get Image List	/collections/{collectionId}/images	>
POST Add image to list	/collections/{collectionId}/images	>
DELETE Delete image	/collections/{collectionId}/images/{imageId}	>
GET Get image description	/collections/{collectionId}/images/{imageId}	>
PUT Update image	/collections/{collectionId}/images/{imageId}	>

Figure 90. Images API Operations

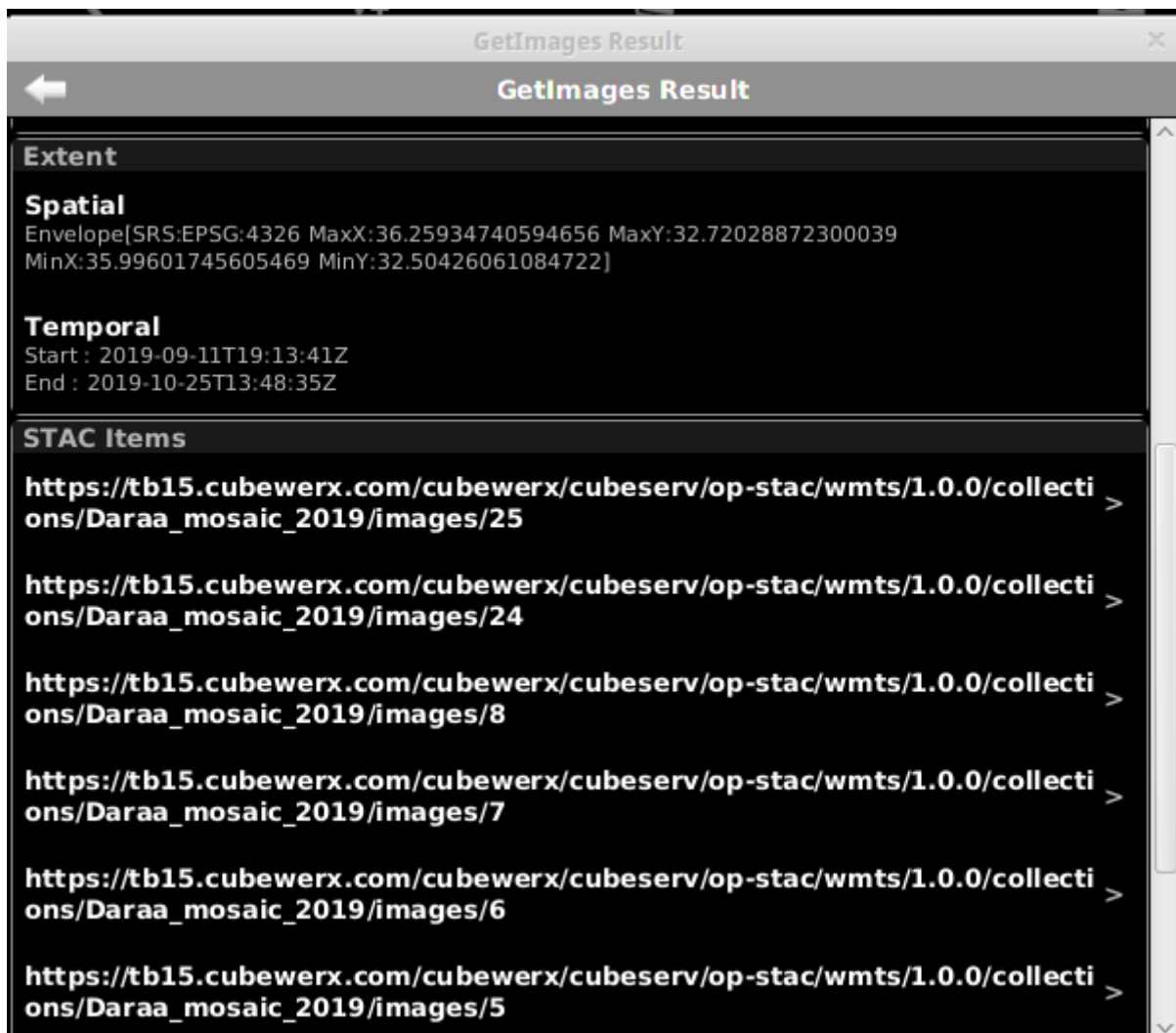


Figure 91. View images using Images API

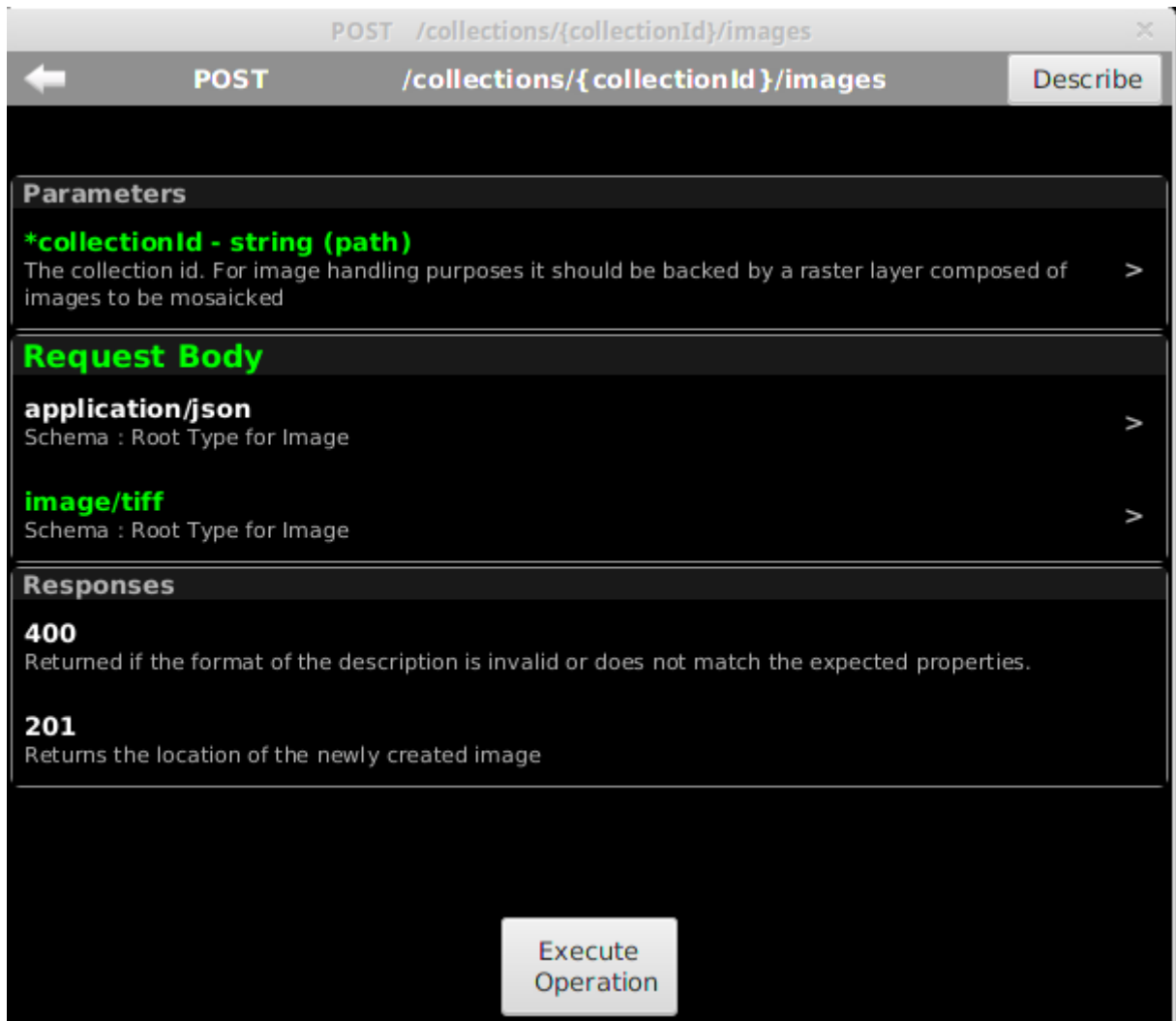


Figure 92. Add image using Images API

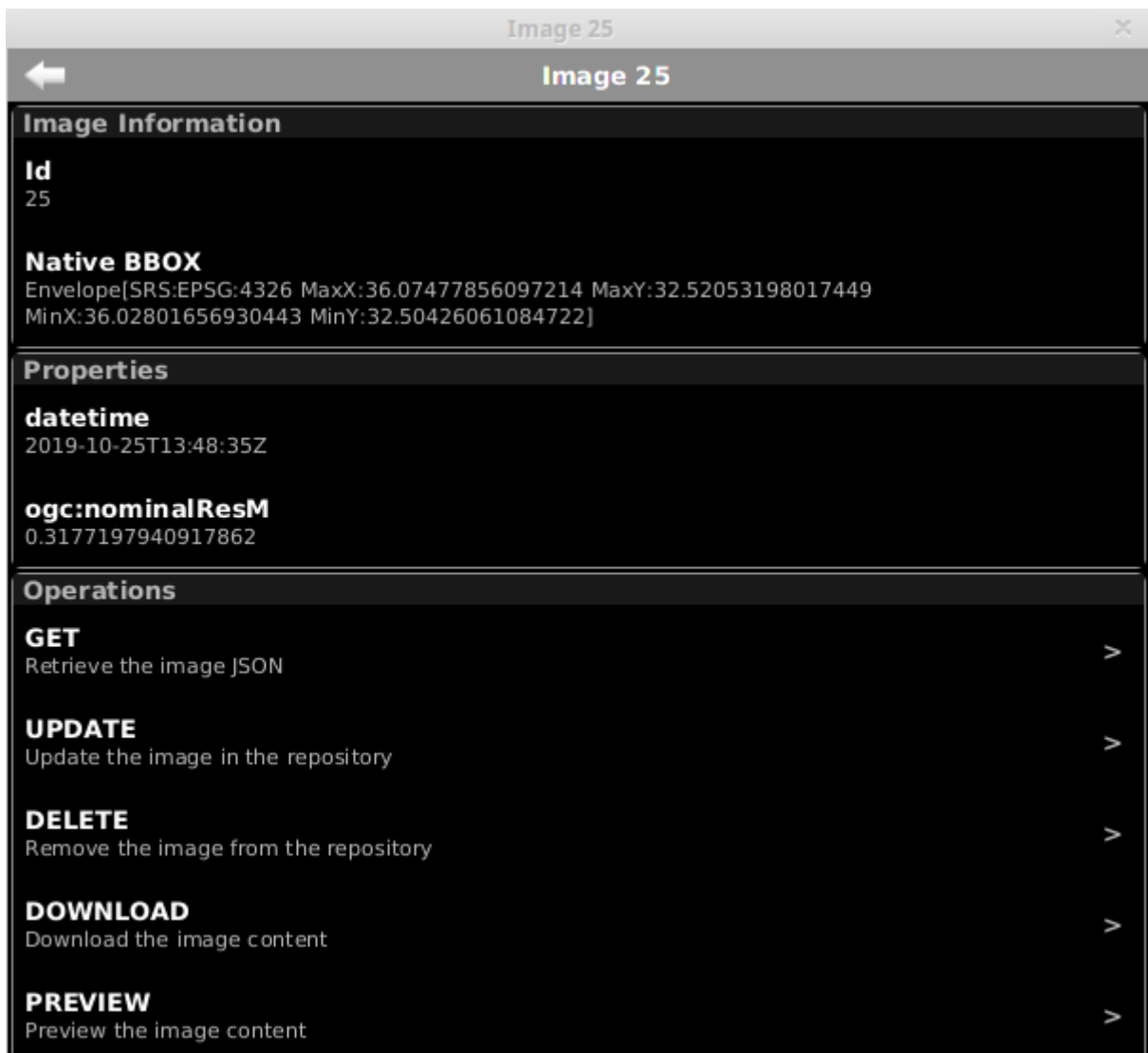


Figure 93. View image meta-data and operations using Images API

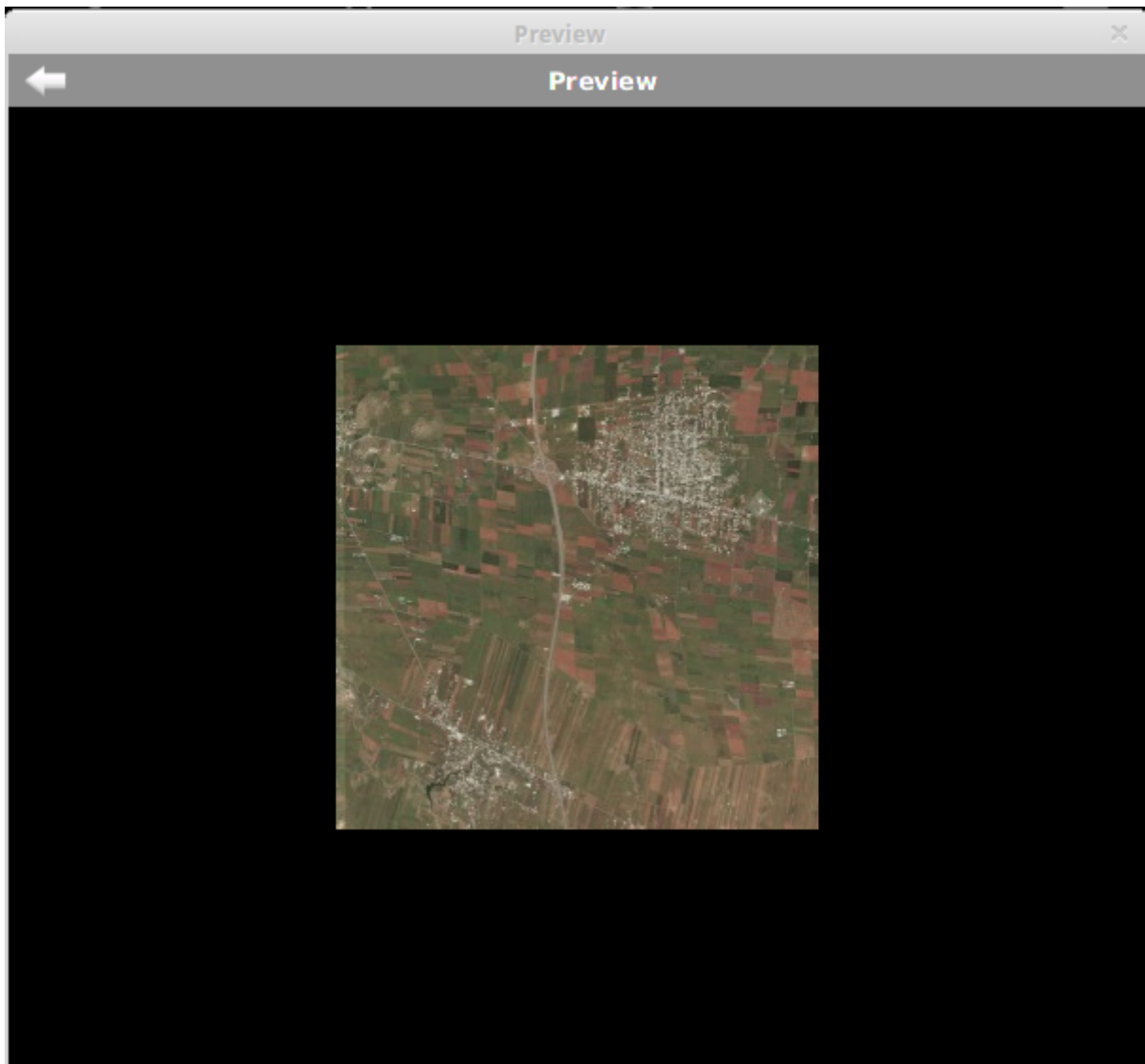


Figure 94. Preview image using Images API

Step 8

Once the Images API has been used to modify its content, the client re-requests the associated tiles to render the new server content (show in [Figure 95](#) and [Figure 96](#)).



Figure 95. CubeWerx WMTS before update

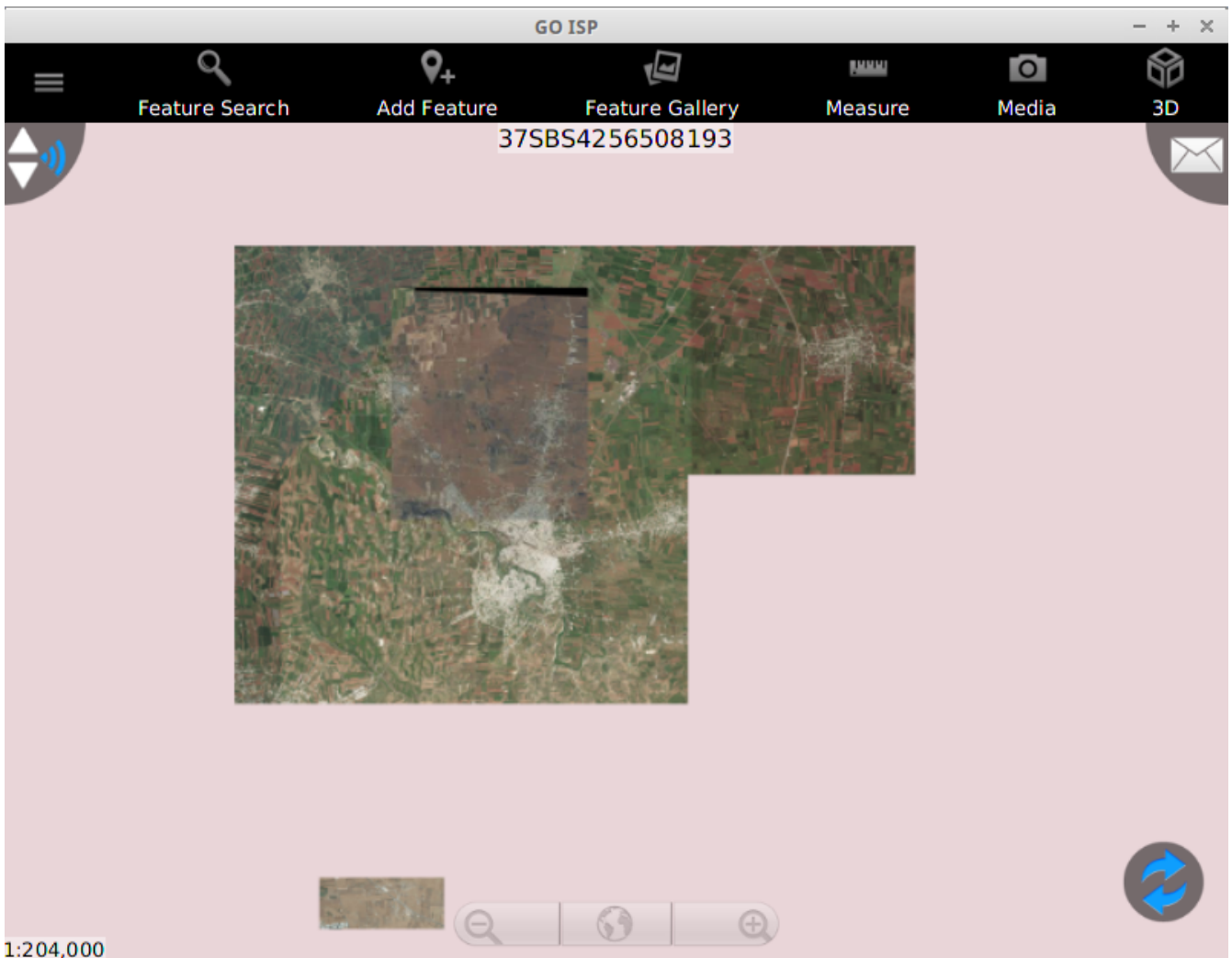


Figure 96. CubeWerx WMTS after performing updates (deletion)

Step 10

The web-based client provides GeoPackage producer support which allows the client to provide raw vector, raster and elevation data along with associated stylesheets and symbology to provide a meta-data and style compliant 1.2 GeoPackage. The following types of GeoPackages can be created :

- Simple Features : Vector
- Vector Tiles : Vector
- Tiles : Raster
- Elevation : Raster

When uploading content the user is given the ability to select the type of content to produce (show in [Figure 97](#)). After creation, the GeoPackage is added to a portfolio, which can then be loaded onto the mobile software for offline use (show in [Figure 98](#)).

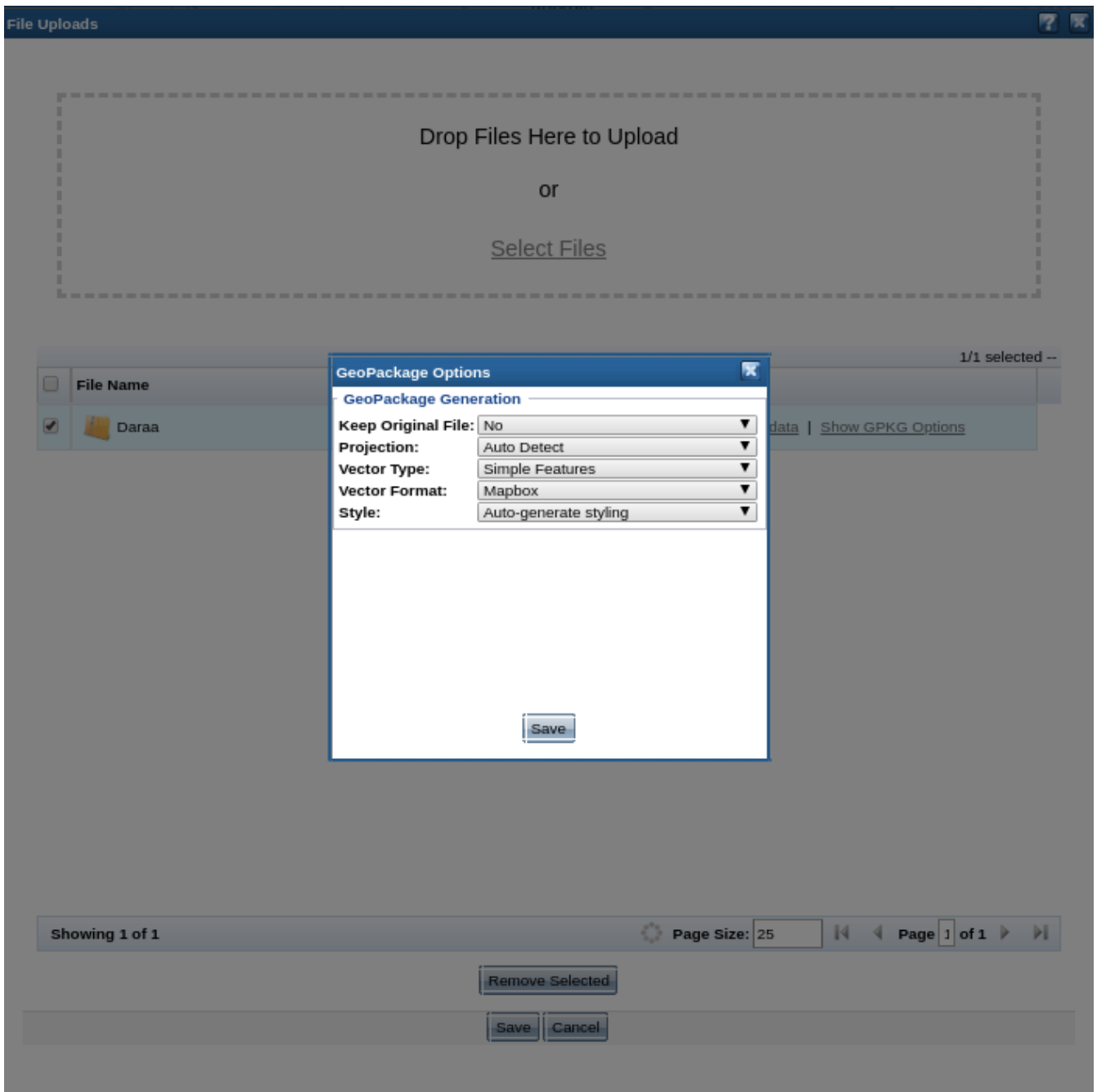


Figure 97. GeoPackage producer configuration

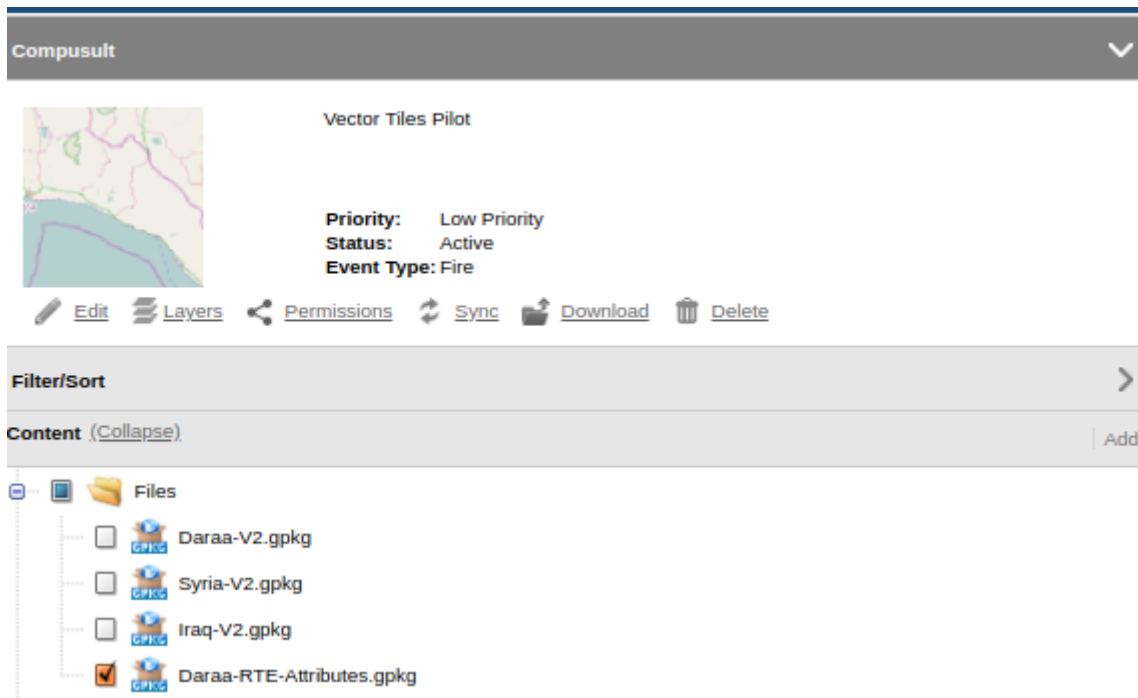


Figure 98. GeoPackage added to a portfolio for offline use (Daraa Vector Tiles)

Step 11

The mobile client (GOMobile) loads produced GeoPackages and services for offline usage and uses the semantic annotations extension to retrieve style and symbol references to apply to the raw data (show in [Figure 99](#), [Figure 100](#), [Figure 101](#), [Figure 102](#), [Figure 103](#)).

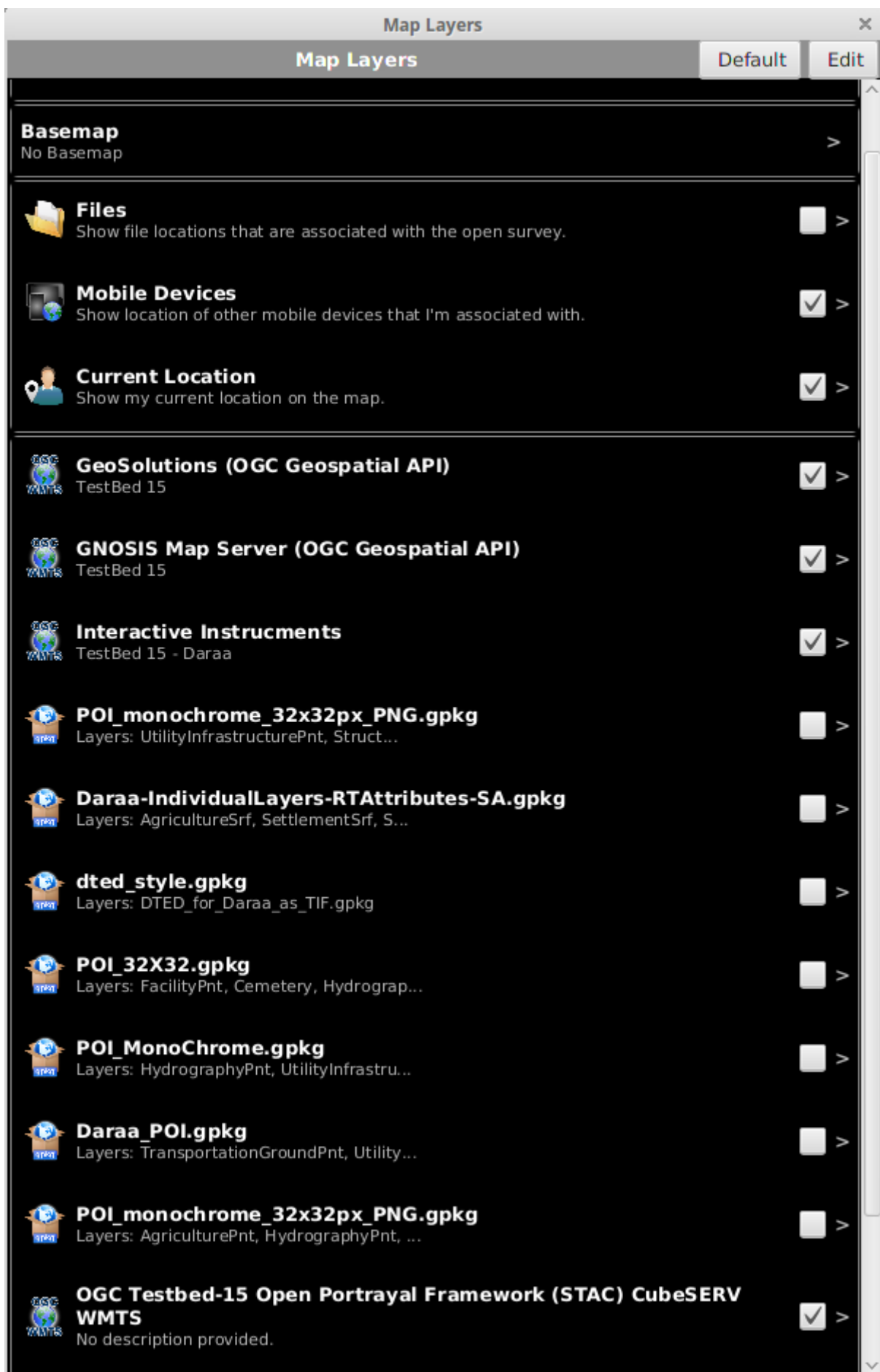


Figure 99. GOMobile GeoPackage/Service list

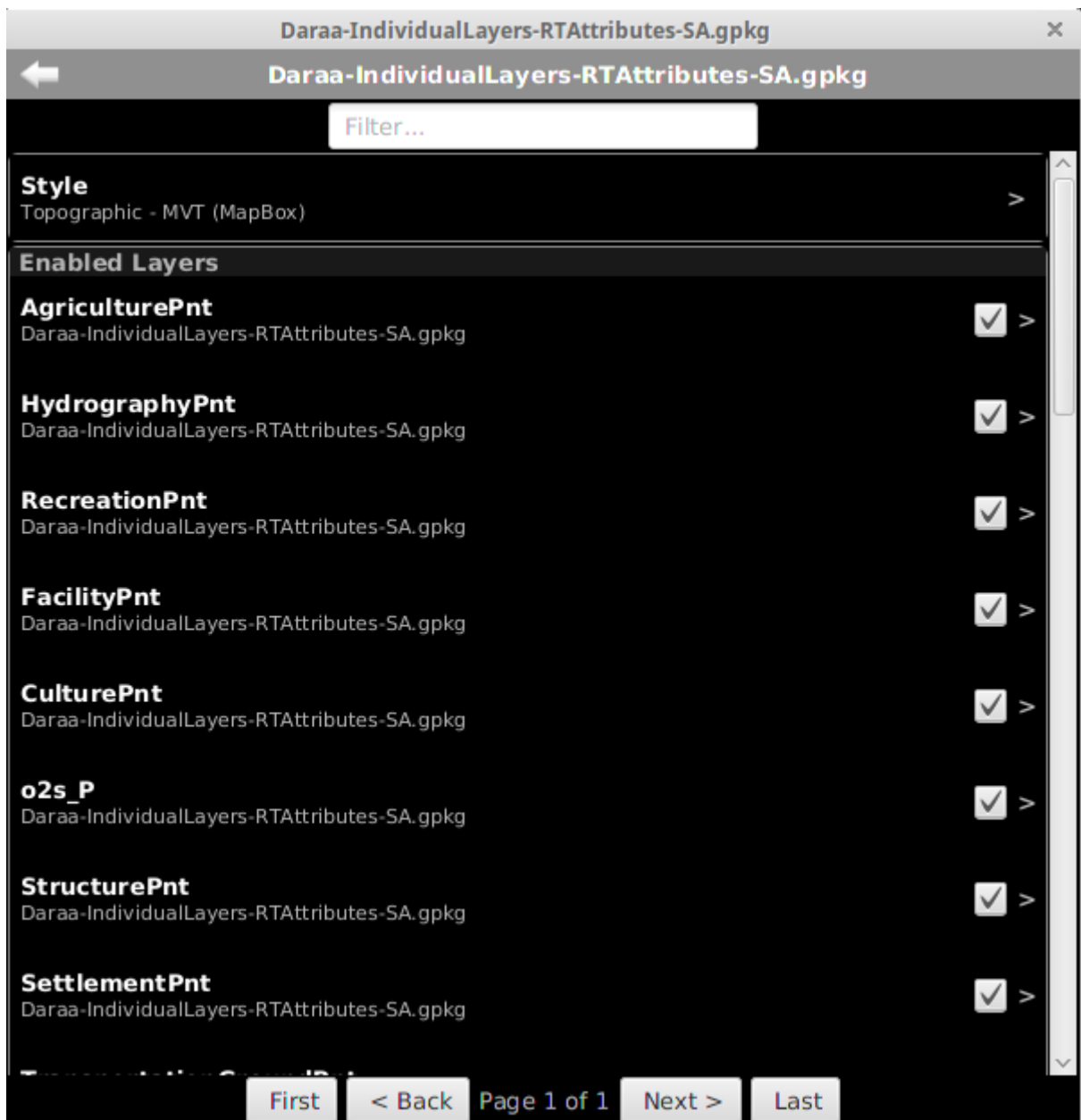


Figure 100. GOMobile Daraq Layer List

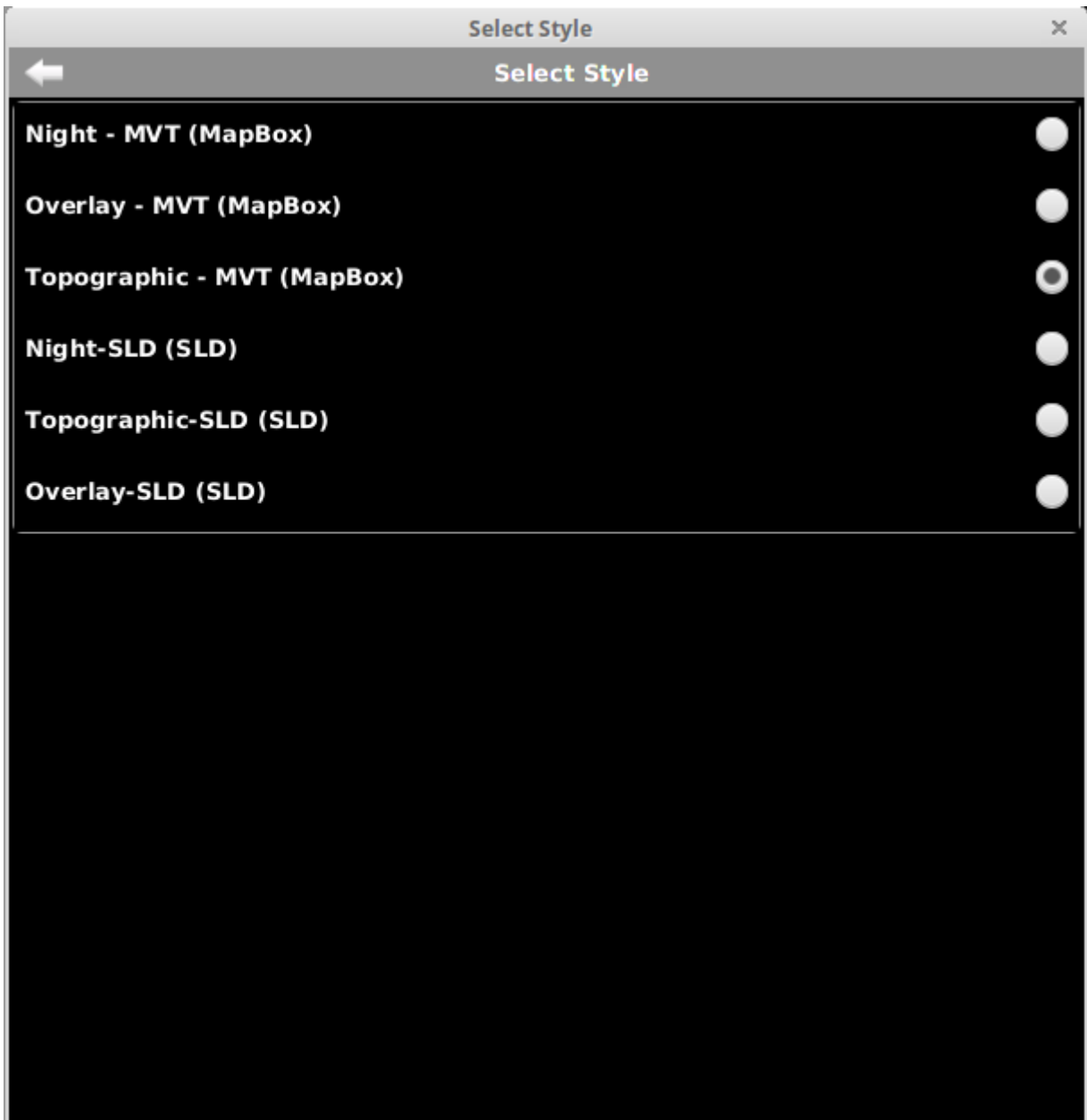


Figure 101. GOMobile Daraa Styles

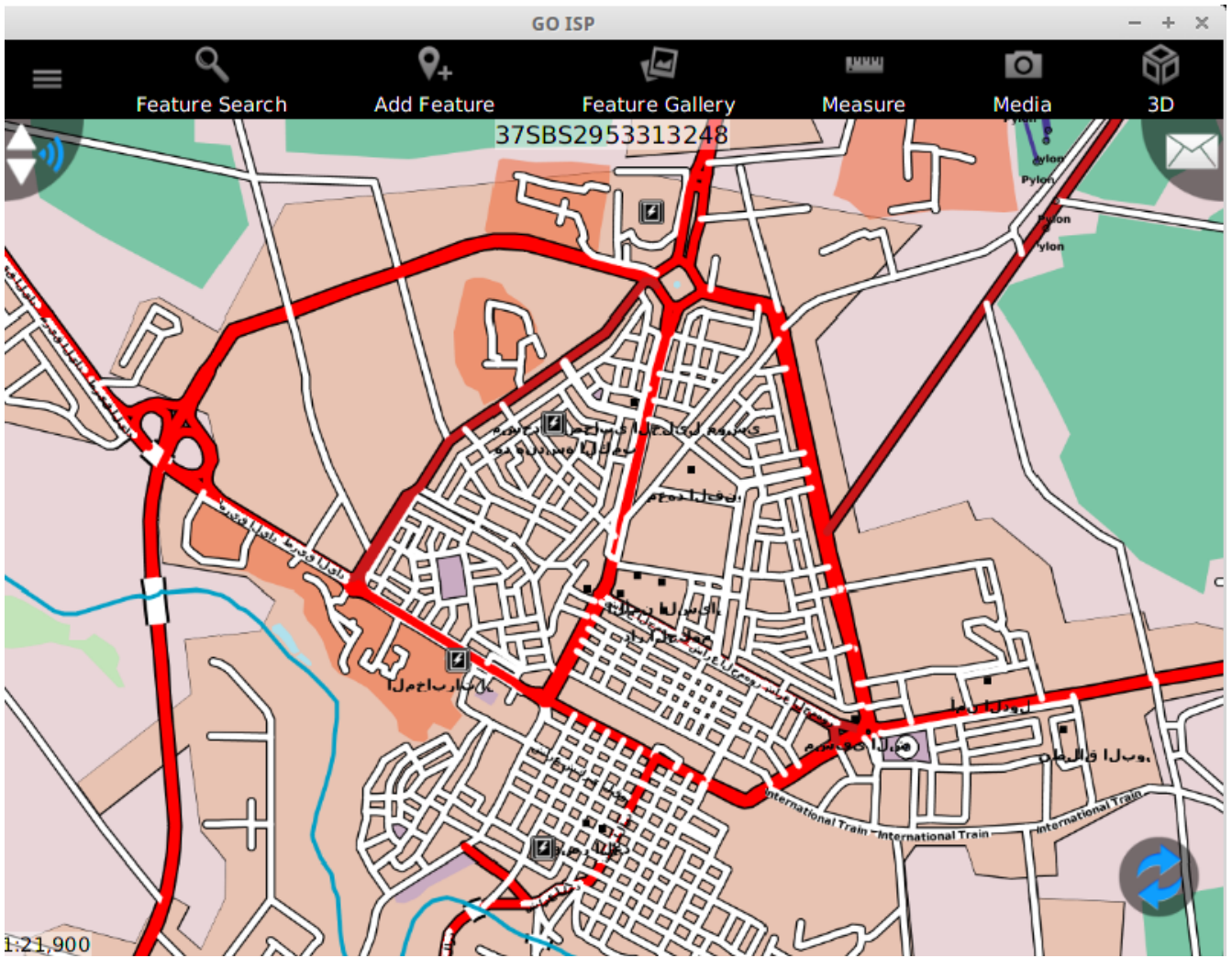


Figure 102. GOMobile POI (Simple Features) and Daraa (Vector Tiles)

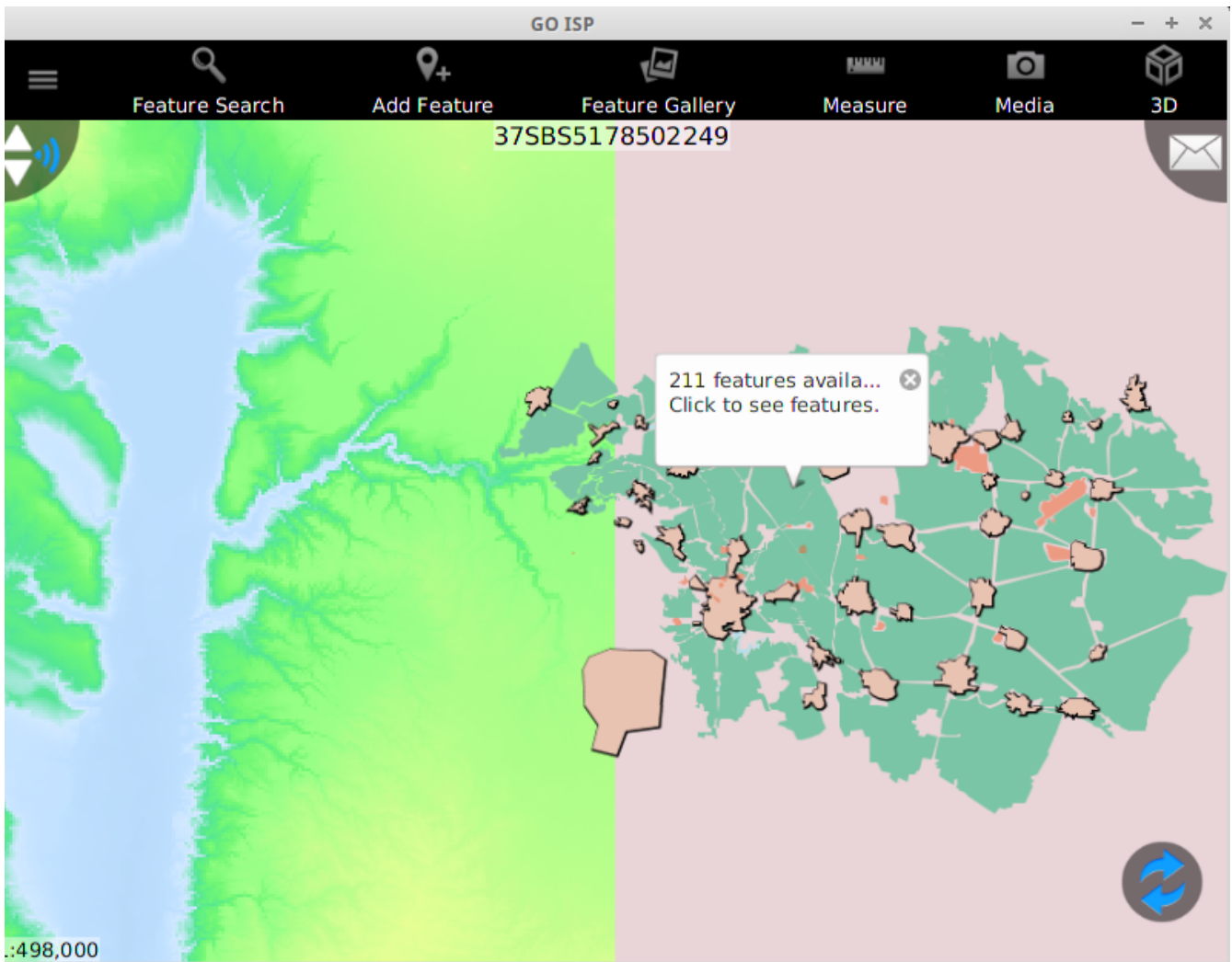


Figure 103. GOMobile DTED (Tiled-Gridded-Elevation) and Daraq (Vector Tiles)

A.7. Implementation Image Matters

Image Matters (IM) implemented a GeoPackage Provisioner and a mobile client to suit scenario steps 10 and 11. The GeoPackage Provisioner was the main focus of this testbed.

The GeoPackage Provisioner takes a JSON-formatted REST API request that includes a bounding box, a coordinate reference system, and an array of layers within services, and creates a GeoPackage that contains every tile from those layers within the bounding box. The GeoPackage also includes applicable Style information. This GeoPackage is then placed in cloud storage, and a download link is provided in the response.

Currently, the GeoPackage Provisioner supports only Web Mercator (EPSG:3857) and WGS-84 (EPSG:4326). As the Provisioner is still being tested, for this Testbed only the GeoSolutions tile server implementation was supported. However, if another service is appropriately interoperable, the Provisioner will function properly as well.

The generated GeoPackage(s) can then be downloaded for offline use. The IM mobile client has a mechanism for downloading GeoPackages that have been distributed. The mobile client user can also view GeoPackages using a map engine.

These GeoPackages are optimized for use on mobile devices. Tiled vector data with Style encodings

generally take much less storage than tile images. However, tiled vector data are intensive to render in real time. The mobile client renders the tiled vector data and caches them temporarily as images. This is done in order to utilize the best qualities of both approaches. This ensures smooth performance on devices with limited memory capacity. Desktop devices are powerful enough that such caching is generally unnecessary.

Appendix B: Revision History

Table 13. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
September 30, 2019	M. Klopfer	0.1	all	initial version
October 27, 2019	C. Reed	0.2	all	provide suggested edits and comments
October 28, 2019	M. Klopfer	0.3	all	draft version
October 31, 2019	M. Klopfer	1.0	all	final version