OGC Testbed-15

Quebec Model MapML Engineering Report

Table of Contents

Publication Date: 2020-01-08

Approval Date: 2019-11-22

Submission Date: 2019-08-22

Reference number of this document: OGC 19-046r1

Reference URL for this document: http://www.opengis.net/doc/PER/t15-D023

Category: OGC Public Engineering Report

Editor: Scott Serich

Title: OGC Testbed-15: Quebec Model MapML Engineering Report

OGC Public Engineering Report

COPYRIGHT

WARNING

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

LICENSE AGREEMENT

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

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

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

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

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

indicate compliance with any LICENSOR standards or specifications.

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

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

Chapter 1. Subject

This OGC Testbed-15 Engineering Report (ER) describes the Map Markup Language (MapML) enabled client component implementation for the Quebec Lake-River Differentiation Model in the Machine Learning (ML) task of Open Geospatial Consortium (OGC) Testbed-15 (T-15). This ER presents the MapML parsing capabilities that were developed to illustrate the outputs of a ML model to delineate lake and river features from an undifferentiated waterbody vector dataset in Québec, Canada. Client data was accessed through an OGC Web Processing Service (WPS) interface in coordination with an OGC API - Features implementation.

Chapter 2. Executive Summary

The Testbed-15 (T-15) Map Markup Language (MapML) work built on the momentum documented in the OGC Testbed-14 MapML Engineering Report [http://www.opengis.net/doc/PER/t14-D012]. The T-15 ER includes a description of the MapML-enabled client component implementation for the Quebec Lake-River Differentiation machine-learning (ML) task. Client parsing capabilities enabled display of model outputs that delineated lake and river features from undifferentiated waterbodies.

MapML is a text format for encoding map information for the World Wide Web. The objective of MapML is to allow Web-based user agent software (browsers and others) to display and edit maps and map data without necessary customization.

MapML is unique relative to other maps on the web encodings but there is also some duplication. What makes MapML unique is that it takes Spatial Data on the Web Best Practices [https://www.w3.org/TR/sdw-bp/] and applies them for the direct benefit of HyperText Markup Language (HTML) users (as contrasted with web developers).

As described by the MapML use cases and requirements [http://maps4html.github.io/HTML-Map-Element-UseCases-Requirements]. MapML is an extension to HTML. If implemented, this implies that the browser understands map/layer semantics (however those elements are eventually named) as well as feature, property, or geometry semantics. MapML is intended to be user-oriented. This includes enabling users to create web pages in all manner of styles while having a solid foundation in HTML and Cascading Style Sheets (CSS) as well as JavaScript for progressive enhancement. Today, there is no built-in map/layer behavior in web browsers, nor is there feature/property/geometry semantics. MapML provides the ability to encode map/layer and feature/property/geometry semantics in a single format that will be read and interpreted by web browsers directly. MapML brings geographic information to the web browser, thereby making the web browser the user agent.

MapML maintains a different focus as compared to other geographic encodings such as GeoJSON, Geography Markup Language (GML), and Keyhole Markup Language (KML). These encodings require interpretation and/or processing and are not native in web browsers. MapML should not be constrained by other encodings if the user requires additional capabilities such as markup in coordinate strings and possibly other requirements. A major objective of MapML is to make the browser "understand" not only where the user is but also to understand where features are in relation to the user.

2.1. Document contributor contact points

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

Contacts

Name	Organization	Role
Scott Serich	Open Geospatial Consortium	Editor
Gil Heo	George Mason University	Contributor

Name	Organization	Role
Je ro me Jacovella-St-Louis	Ecere Corporation	Contributor
Peter Rushforth	Natural Resources Canada	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 ER:

- OGC: OGC 06-121r9, OGC® Web Services Common Standard (2010) [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- OGC: OGC 07-036r1, OpenGIS® Geography Markup Language (GML) Encoding Standard - with corrigendum (2016) [https://portal.opengeospatial.org/files/?artifact_id=74183&version=2]
- OGC: OGC 07-057r7, OGC® OpenGIS Web Map Tile Service Implementation Standard (2010) [http://portal.opengeospatial.org/files/?artifact_id=35326]
- OGC: OGC 17-069r3, OGC API - Features - Part 1: Core (2019) [https://www.opengeospatial.org/standards/ogcapi-features]
- OGC: OGC 12-007r2, OGC KML 2.3 (2015) [http://docs.opengeospatial.org/is/12-007r2/12-007r2.html]
- IETF: RFC 7946, The GeoJSON Format (2016) [https://tools.ietf.org/html/rfc7946]
- W3C: GeoLocation API (2016) [https://www.w3.org/TR/geolocation-API/]
- W3C: Hypertext Markup Language (HTML) (2019) [https://html.spec.whatwg.org/multipage/]

Chapter 4. Terms and definitions

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

- **Dnnn**

  Deliverable IDs used during the initiative to uniquely identify various components, engineering reports, etc. (for example, the *D106 Quebec Model MapML Client* component).

- **map**

  portrayal of geographic information as a digital image file suitable for display on a computer screen

- **portrayal**

  presentation of information to humans (source: ISO 19117)

4.1. Abbreviated terms

- API - Application Programming Interface
- CORS - Cross-Origin Resource Sharing
- CSS - Cascading Style Sheets
- DOM - Document Object Model
- EC2 - Elastic Compute Cloud
- GML - Geography Markup Language
- HTML - Hypertext Markup Language
- IANA - Internet Assigned Numbers Authority
- ISO - International Organization for Standardization
- JPEG - Joint Photographic Experts Group
- JSON - JavaScript Object Notation
- KML - Keyhole Markup Language
- MapML - Map Markup Language
- MVC - Model-View-Controller
- OGC - Open Geospatial Consortium
- OWS - OGC Web Services
- PNG - Portable Network Graphics
- SVG - Scalable Vector Graphics
- URL - Uniform Resource Locator
- WFS - Web Feature Service

- WMS - Web Map Service
- WMTS - Web Map Tile Service
- WPS - Web Processing Service

Chapter 5. Overview

Much of the OGC Testbed-15 Quebec Model MapML work was based on foundational work of the MapML W3C Community Group [https://www.w3.org/community/maps4html/] and the OGC Testbed-14 MapML work documented in the Testbed-14 Engineering Report [http://www.opengis.net/doc/PER/t14-D012].

Section 5 of this ER describes the background of the current work.

Section 6a describes the OGC API - Features implementation that served MapML data to the client.

Section 6b describes the Quebec MapML Client that rendered retrieved features to a browser user.

Chapter 6. Background

The overall architecture of the Testbed-15 Machine Learning Task was based on a broad set of scenarios ranging from image classification to dataset discovery. The objective of the Quebec Lake-River Differentiation Model subtask was to create and deploy a machine-learning model to differentiate between *rivers* and *lakes* from otherwise unlabeled bodies of water. The main focus of the work was to provide a service to determine whether a body of water should be split into a *lake* and a *river*, and to provide identified and labeled outputs when such splits occur. When no split was required, each body of water was labeled as either *lake* or *river*. The procedure for applying the model was to:

1. Recommend whether a water body should be split into lake and river features.
2. Evaluate the confidence level of a recommendation.
3. Apply the recommendation to the dataset.
4. Test and correct the resultant dataset for topological and cartographical issues.
5. Serve the data on an OGC API - Features interface using a MapML encoding.

Each service component was fronted by the relevant OGC interface, WPS for workflow management and OGC API - Features for features.

Chapter 7. Québec Model MapML Service

In support of the Testbed-15 Québec Lakes/Rivers experiments, Ecere was responsible for the development of a service component to facilitate access to the potentially large volumes of vector data produced by running the differentiation Machine Learning model.

7.1. Service description

Ecere developed and deployed a new MapML-enabled OGC API end-point based on its newest iteration of the GNOSIS Map Server and Software Development Kit.

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 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 and was the focus of this activity)

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

7.2. 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 Québec Lakes/Rivers Machine Learning activities are organized within group of collections named *GRHQ*. GRHQ stands for *Géobase du réseau hydrographique du Québec* [https://mern.gouv.qc.ca/repertoire-geographique/reseau-hydrographique-grhq/],

free and open hydrography data covering the entire Province of Québec. These data are provided by the *Ministère de l'Énergie et des Ressources naturelles* (MERN). Together with High Resolution Digital Elevation Models, this was the source hydrography data set used for both training and testing the machine learning model.

Originally, only the output data from the *D104 - Quebec Lakes/Rivers Machine Learning Model* component was intended to be served. However due to constraints delaying the availability of that output data, the source hydrography vector data was provided instead.

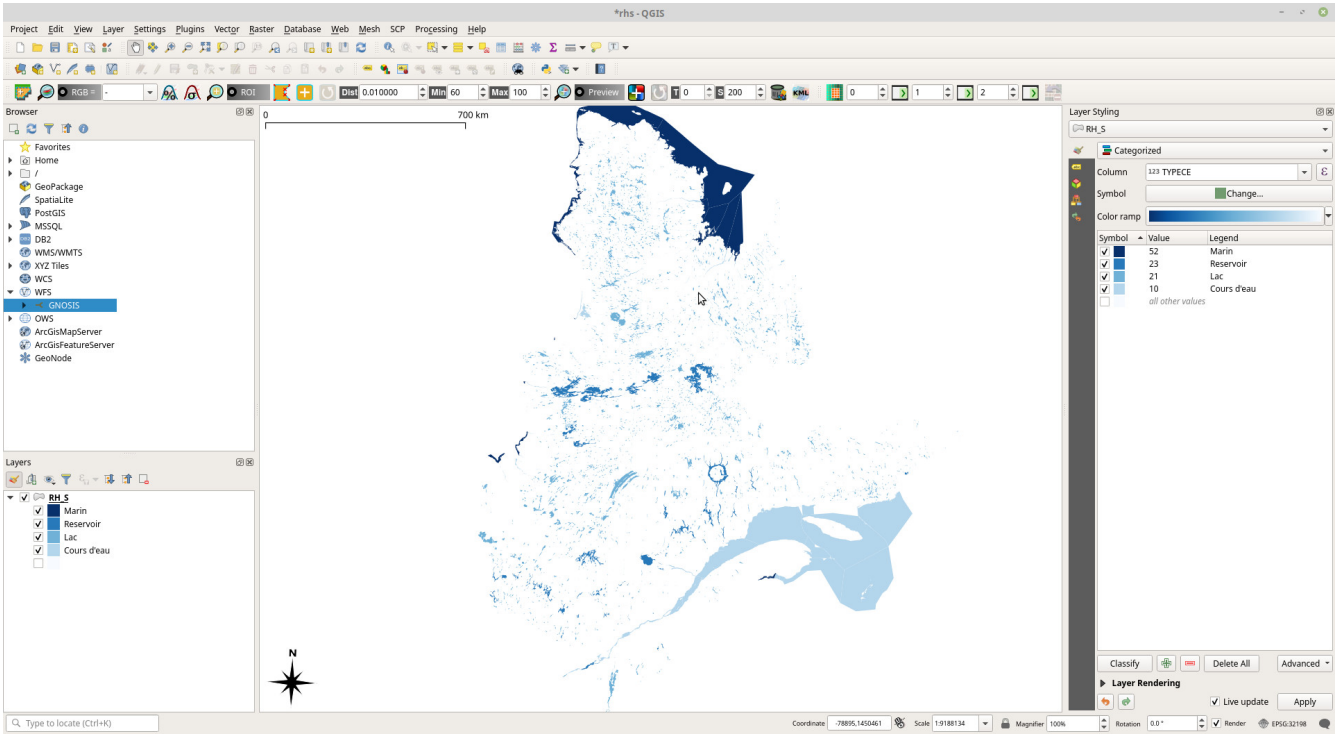

Figure 1. ***GRHQ served from GNOSIS Map Server visualized in QGIS (Overview)***

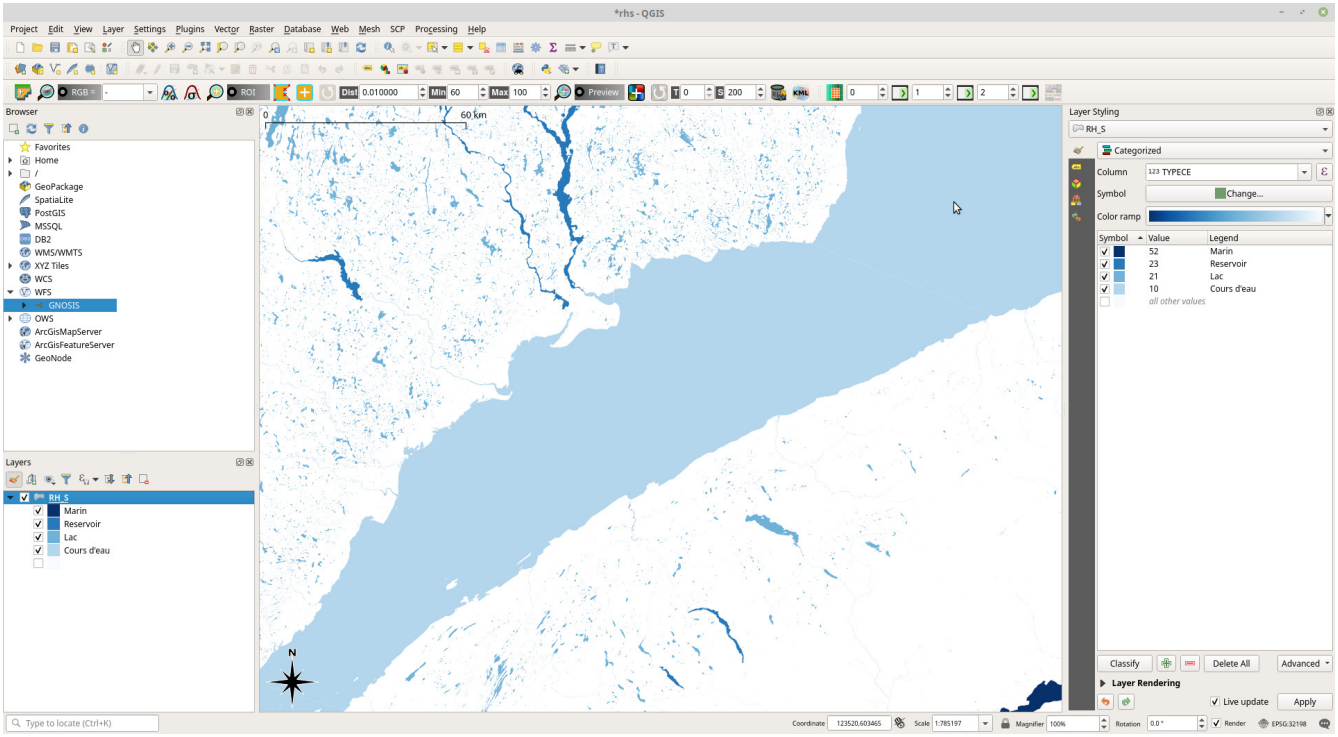

Figure 2. ***GRHQ served from GNOSIS Map Server visualized in QGIS (Zoomed in)***

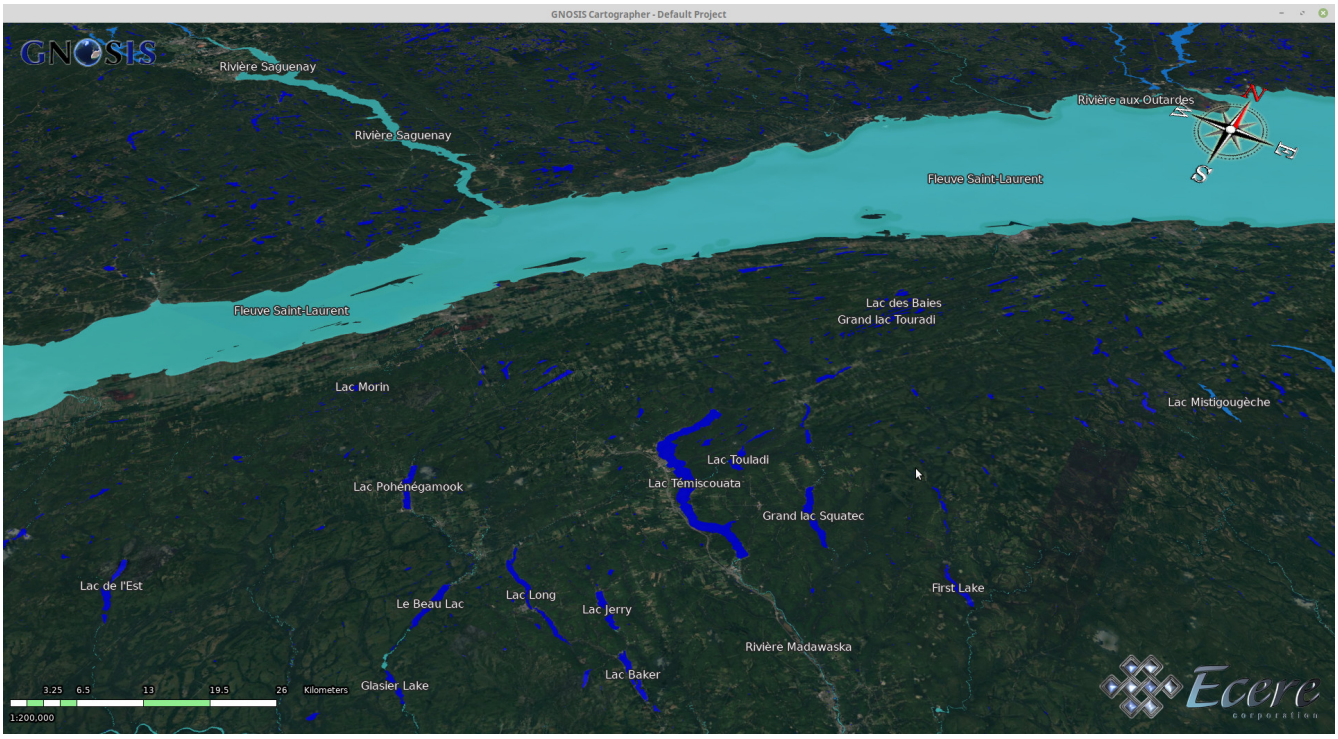

Figure 3. ***GRHQ served from GNOSIS Map Server visualized in GNOSIS Cartographer***

Once results from running the model became available, they were added to the service. However, these results only reached the stage of defining bounding boxes within which lakes were detected in the dataset, not a differentiated collection of polygon features as expected. This layer for these detection bounding boxes is available as *detections002*, and covers only the small area over which the machine learning model was used.

These two layers are available from the service:

- RH_S [http://maps.ecere.com/geoapi/collections/GRHQ/RH_S] (réseau hydrographique — surfaces, polygonal features of the hydrographic network)
- detections002 [http://maps.ecere.com/geoapi/collections/GRHQ/detections002] (bounding boxes resulting from D104 machine learning model, provided by CRIM)

The MapML representation for the description of these collections is available at:

- http://maps.ecere.com/geoapi/collections/GRHQ/RH_S?f=mapml (RH_S)
- http://maps.ecere.com/geoapi/collections/GRHQ/detections002?f=mapml (detections002)

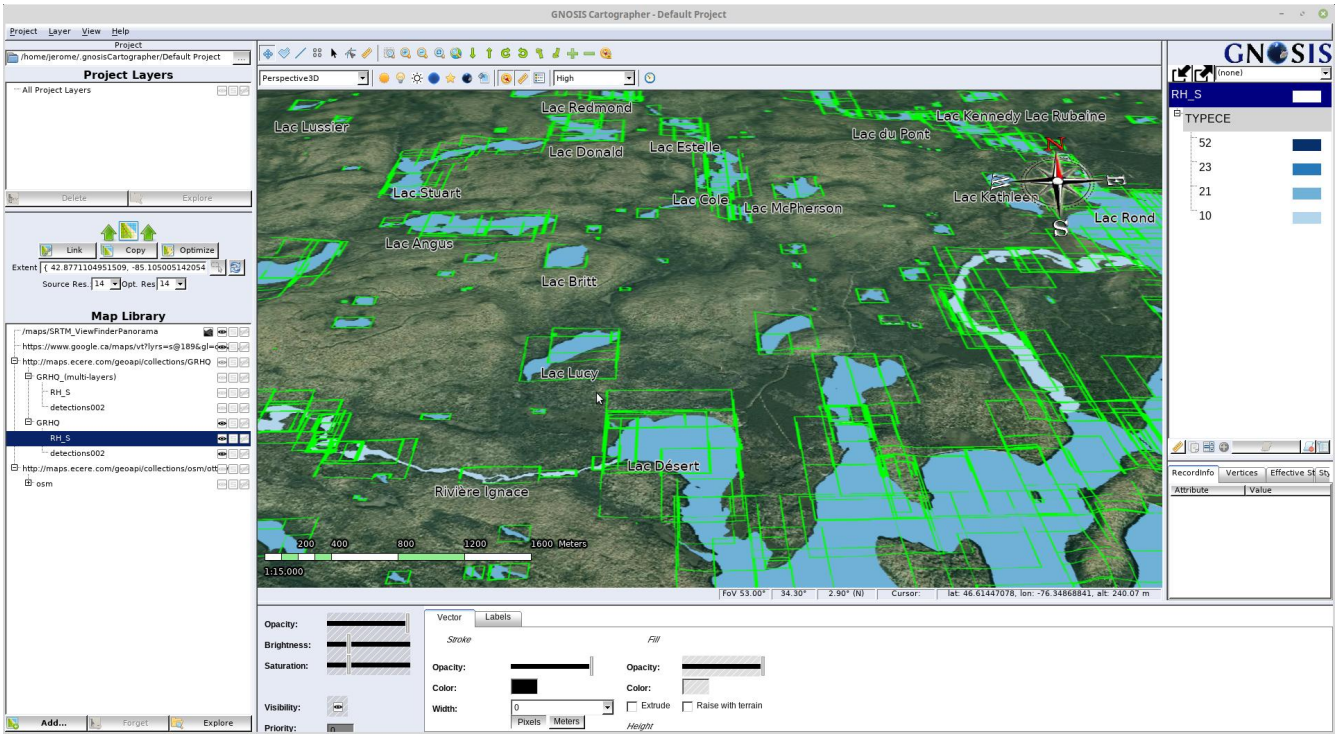

Figure 4. ***GRHQ and lakes detection bounding boxes served from GNOSIS Map Server visualized in GNOSIS Cartographer, Google Imagery***

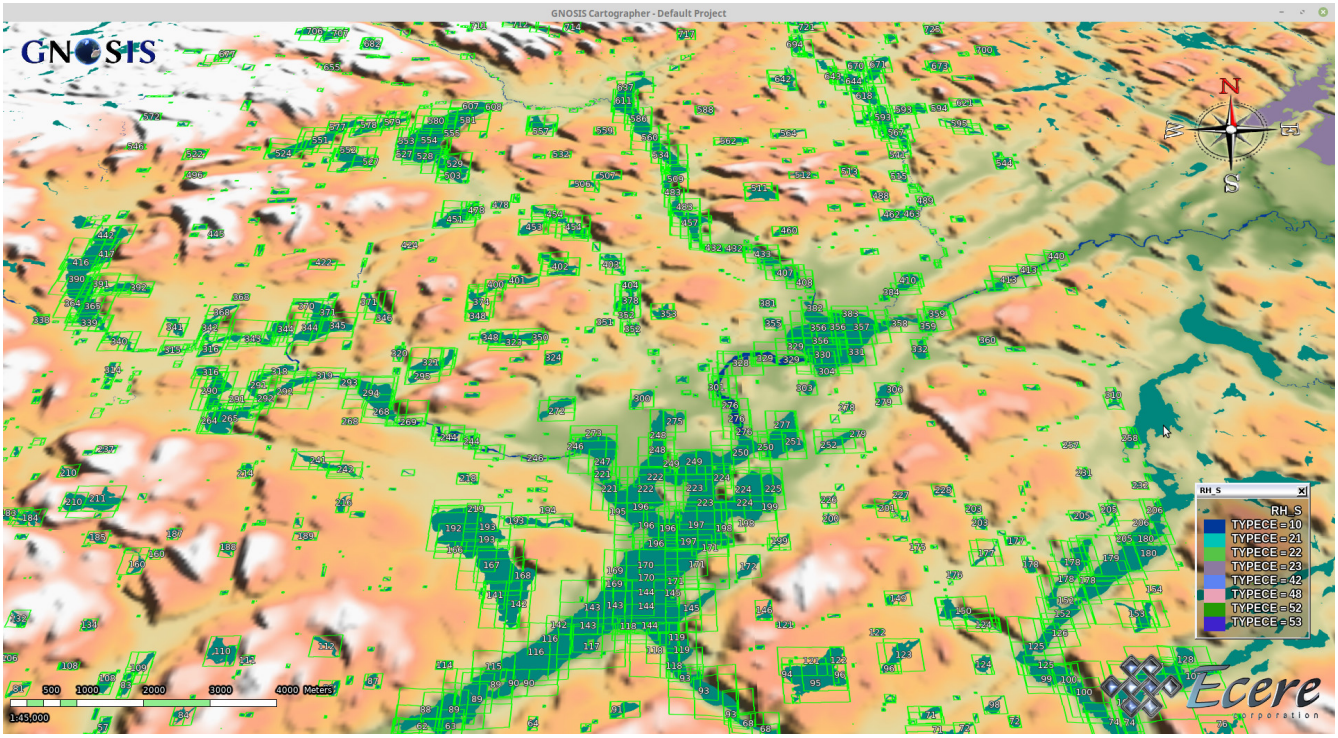

Figure 5. ***GRHQ and lakes detection bounding boxes served from GNOSIS Map Server visualized in GNOSIS Cartographer, SRTM ViewFinderPanorama Elevation***

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/GRHQ/RH_S .

Note that the GRHQ provides several other layers, in addition to the RH_S polygonal features layer. These layers have not been loaded onto the service for reasons of storage capacity, processing time, and the fact that they were not required for the activity. However, the entire set of layers was retrieved (necessitating a laborious manual process), and converted into a single GeoPackage,

making the data easier to use and transfer. The downside is that the GeoPackage is 23 gigabytes! This GeoPackage was uploaded to a File Transfer Protocol (FTP) site hosted by OGC, in case the data might prove useful to participants or sponsors for continuity of this work, such as to train and/or run the machine learning model over the whole province.

7.3. MapML and OGC API integration

Originally the main capability that this activity was to deliver was described as a *WFS 3.0 MapML* service, to be based on the draft version 3 of the OGC Web Feature Service (WFS) standard. Between the time the Testbed 15's Call for Proposal (CFP) was released and the end of the activity, WFS 3 had officially become the *OGC API - Features* standard, the first of a new OGC API family of modular service capabilities (OGC 17-069r3). There was also an interest within the initiative to investigate the possibility of supporting tiled vector data. This interest was also the subject of separate work within the testbed as shown by the development of an *OGC API - Tiles* draft specification.

The work done in this initiative attempted to adequately determine how MapML and the OGC API could best integrate in a complimentary manner. This relied on discussions between the stakeholders during the initiative, as well as past and on-going experience of the participants with the evolving OGC API family of specifications within other projects of the OGC Innovation and Standards Programs.

The first realization was that MapML is used for at least two traditionally distinct purposes in the geospatial information community, notably:

- As a document defining a map, referencing geospatial data and styles available separately
- As an encoding for vector data, whether on its own or within such an aforementioned map definition document

Please note that GeoJSON is already a well-established encoding widely used and supported by libraries. GeoJSON is also the default encoding for OGC API – Features. Therefore, the recommendation was made that perhaps MapML brings significantly more value as a map definition language (e.g. as an HTML map element within a web page) than it does as another encoding for geospatial data.

Nonetheless, the service developed focused on the encoding aspect which was identified as the primary requirement but also tried to accommodate the definition and reference use case, notably to enable MapML clients to access data using the Tiles API.

7.3.1. MapML representation of an OGC API collection description

The integration of map definition and linking to separate data and styles with the OGC API was done at the level of the collection description resource. An attempt was made at providing a MapML representation of the collection resource. Instead of the default JSON representation, this representation is returned by specifying the f=mapml parameter when requesting the collection description (*/collections/{collectionId}?f=mapml*).

Due to the nature and purpose of the collection resource not corresponding directly with the concept of a MapML document this approach posed challenges as to exactly what could be featured

within this MapML representation. On one hand, a MapML document is expected to contain a single reference to a particular dataset. On the other, an OGC API collection can be retrieved in different manners, such as tiles or as whole features (potentially together with an intersection or clipping bounding box).

Similarly, the same problem presents itself for linking to multiple encodings of the data, such as offering features or tiles as GeoJSON, MapML, GNOSIS Map Tiles and Mapbox Vector Tiles.

The collection description resource for the OGC API is typically not parameterized, as its purpose is only to describe the particular resource being offered. A MapML representation should therefore also avoid introducing parameters to alter the response of retrieving a collection description resource. This ensures the possibility of an OGC API simply directly serving data from a static file system structure. Instead, it would be best to make it possible for a higher-level geospatial content management system to reference a single generic MapML collection resource, while selecting specific capabilities.

In order for a MapML representation of the collection description resource to integrate well with the OGC API, it would be ideal if it could be harmonized as much as possible with its typical JSON equivalent. This harmonization would cover aspects such as:

- The types of link relations,
- The ability to list tiled and non-tiled data links, of multiple encodings,
- The ability to list multiple usable styles, without a specific being already selected.

It was noted that a client could consider the type attribute of a link to recognize and select the best supported encoding, and ignore alternate contents.

The MapML community would need to assess whether it is still possible and desirable to achieve this harmonization based on the current evolutionary stage of the MapML specification.

Alternatively, the OGC API integration could focus solely on the encoding, and a geospatial content management system could instead directly refer to the tiles or items being served by the OGC API, avoiding the integration challenges of the collection description resource altogether.

Because no consensus could be reached during the testbed, it was decided to only provide a link to the tiled data within the collection description (rel=tile). MapML is the encoding specified for those vector tiles, and marks the first attempt at using MapML to encode vector tiles or integrating vector tiles within a MapML document. The links to the features (rel=features) were still included in the document, but commented out. A link to the items was still included together with a bbox parameter for the purpose of querying (rel=query). Links to other supported encodings were also provided within the document, but commented out.

GRHQ/RH_S collection description represented as MapML

```
<mapml>
   <head>
      <title>RH_S</title>
      <base href="http://maps.ecere.com/geoapi/collections/GRHQ/RH_S"/>
      <meta charset="utf-8"/>
      <meta content="text/mapml" http-equiv="Content-Type"/>
```

```
      <!--Collection Information-->
      <link rel='self' type='text/mapml' title='Information about the RH_S data (as
MapML)'
         href='./?f=mapml'/>
      <link rel='alternate' type='text/html' title='Information about the RH_S data
(as HTML)'
         href='./'/>
      <link rel='alternate' type='application/json' title='Information about the RH_S
data (as JSON)'
         href='./?f=json'/>
      <link rel='alternate' type='text/plain' title='Information about the RH_S data
(as ECON)'
         href='./?f=econ'/>

      <!--Tiling Schemes-->
      <link rel='tilingSchemes' type='text/html'
         title='Tiling schemes for RH_S (tiles interface; as HTML)' href='./tiles'/>
      <link rel='tilingSchemes' type='application/json'
         title='Tiling schemes for RH_S (tiles interface; as JSON)' href=
'./tiles?f=json'/>
      <link rel='tilingSchemes' type='text/plain'
         title='Tiling schemes for RH_S (tiles interface; as ECON)' href=
'./tiles?f=econ'/>

      <!--Styles-->
   </head>
   <body>
      <extent units="WGS84">
         <input name="zoomLevel" type="zoom" min="0" max="15"/>
         <input name="minLon" type="location" units="gcrs" axis="longitude" position=
"top-left"
            min="-85.1050051420546" max="-50.8421313888137"/>
         <input name="minLat" type="location" units="gcrs" axis="latitude" position=
"bottom-right"
            min="42.8771104951509" max="63.0380382893297"/>
         <input name="maxLon" type="location" units="gcrs" axis="longitude" position=
"bottom-right"
            min="-85.1050051420546" max="-50.8421313888137"/>
         <input name="maxLat" type="location" units="gcrs" axis="latitude" position=
"top-left"
            min="42.8771104951509" max="63.0380382893297"/>

         <!--Features Items-->
         <!--Note: requesting large extent may result in data
            more generalized than requested zoom level-->
         <!--link rel='features' type='text/mapml' title='RH_S (as MapML)'
            tref='./items.mapml?tilingScheme=CRS84Quad2L0Tiles&amp;
            zoomLevel={zoomLevel}&amp;clipbox={minLon},{minLat},{maxLon},{maxLat}'/>
         <link rel='features' type='application/vnd.geo+json' title='RH_S (as
GeoJSON)'
```

```
            tref='./items.json?tilingScheme=CRS84Quad2L0Tiles&amp;
            zoomLevel={zoomLevel}&amp;clipbox={minLon},{minLat},{maxLon},{maxLat}'/>
         <link rel='features' type='application/vnd.geo+econ' title='RH_S (as
GeoECON)'
            tref='./items.econ?tilingScheme=CRS84Quad2L0Tiles&amp;
            zoomLevel={zoomLevel}&amp;clipbox={minLon},{minLat},{maxLon},{maxLat}'/>
         <link rel='features' type='text/xml; subtype=gml/3.1.1' title='RH_S (as GML)'
            tref='./items.gml?tilingScheme=CRS84Quad2L0Tiles&amp;
            zoomLevel={zoomLevel}&amp;clipbox={minLon},{minLat},{maxLon},{maxLat}'/>
         <link rel='features' type='application/vnd.mapbox-vector-tile'
            title='RH_S (as Mapbox Vector Tile)'
            tref='./items.mvt?tilingScheme=CRS84Quad2L0Tiles&amp;
            zoomLevel={zoomLevel}&amp;clipbox={minLon},{minLat},{maxLon},{maxLat}'/-->

         <input name="row" type="location" axis="row" units="tilematrix"/>
         <input name="col" type="location" axis="column" units="tilematrix"/>

         <!--Tiles-->
         <link rel='tile' type='text/mapml' title='Tiles for RH_S (as MapML)'
            tref='./tiles/CRS84Quad2L0Tiles/{zoomLevel}/{row}/{col}.mapml'/>
         <!--link rel='tile' type='application/vnd.mapbox-vector-tile'
            title='Tiles for RH_S (as Mapbox Vector Tiles)'
            tref='./tiles/CRS84Quad2L0Tiles/{zoomLevel}/{row}/{col}.mvt'/>
         <link rel='tile' type='application/vnd.geo+econ'
            title='Tiles for RH_S (as GeoECON)'
            tref='./tiles/CRS84Quad2L0Tiles/{zoomLevel}/{row}/{col}.econ'/>
         <link rel='tile' type='application/vnd.geo+json'
            title='Tiles for RH_S (as GeoJSON)'
            tref='./tiles/CRS84Quad2L0Tiles/{zoomLevel}/{row}/{col}.json'/>
         <link rel='tile' type='text/xml; subtype=gml/3.1.1'
            title='Tiles for RH_S (as GML)'
            tref='./tiles/CRS84Quad2L0Tiles/{zoomLevel}/{row}/{col}.gml'/-->
         <!--link rel='tile' type='application/vnd.gnosis-map-tile'
            title='Tiles for RH_S (as GNOSIS Map Tiles)'
            tref='./tiles/CRS84Quad2L0Tiles/{level}/{row}/{col}.gmt'/-->

         <link rel='query' type='text/mapml' title='RH_S (as MapML)'
            tref='./items.mapml?tilingScheme=CRS84Quad2L0Tiles&amp;
            zoomLevel={zoomLevel}&amp;bbox={minLon},{minLat},{maxLon},{maxLat}'/>
      </extent>
   </body>
</mapml>
```

7.3.2. Styling considerations

The testbed examined the question of providing links to styles applicable to a collection.

One viewpoint posited that an OGC API collection is strictly data, facilitating its presentation in different ways. Under this view, the collection description would remain neutral of the style, but could list the styles which the client could separately decide to apply (e.g., in a higher-level MapML

document including or linking to this collection). A collection might have defined for it one or more styles. The collection description resource, which itself would be style-agnostic, would provide information about the availability of these styles. A *style* parameter could potentially be used within a style sheet link to name the different styles, and the higher-level document could pick a style, by also specifying a *style* parameter when defining a <layer>.

It was also noted that the specialized requirements of cartographic styling such as scale-dependent styles, complex expressions, etc. can be more complex than what Web CSS was intended to address. To address these more advanced needs, MapML-enabled clients could consider supporting Web-CSS inspired specifications such as CartoCSS, GeoCSS, and GNOSIS CMSS.

An alternative viewpoint was that HTML/MapML combines data and affordances, but does separate presentation (style) from content. Styles (CSS) are separated from (and linked to) the content. Under this view, a named style could be included in a parameter. A MapML client could interpret multiple style sheets linked from a single collection description as styles which should all be applied in cascade.

The question of selecting alternative styles for MapML was examined in Testbed-14, with findings presented in the Testbed-14 MapML Engineering Report [http://docs.opengeospatial.org/per/18-023r1.html#_selecting_alternative_styles_for_mapml]. Since the styling aspect was not a core requirement for this component, a resolution was not pursued. It is recommended that these considerations be investigated in future work that takes into consideration the growing OGC API family [https://www.opengeospatial.org/pressroom/pressreleases/3106] of standards.

Sample code appears below as an illustration.

Styles section for NaturalEarth/cultural/ne_10m_admin_0_countries collection description

```
...

<!--Styles-->
<link rel='stylesheet' type='application/vnd.gnosis.cmss+eccss' style='default'
   href='./styles/default.cmss'/>
<link rel='stylesheet' type='application/vnd.ogc.sld+xml' style='default'
   href='./styles/default.sld'/>
<link rel='stylesheet' type='application/vnd.mapbox.style+json' style='default'
   href='./styles/default.json'/>

<link rel='stylesheet' type='application/vnd.mapbox.style+json' style='mbglImportTest'
   href='./styles/mbglImportTest.json'/>
<link rel='stylesheet' type='application/vnd.gnosis.cmss+eccss' style='mbglImportTest'
   href='./styles/mbglImportTest.cmss'/>
<link rel='stylesheet' type='application/vnd.ogc.sld+xml' style='mbglImportTest'
   href='./styles/mbglImportTest.sld'/>

<link rel='stylesheet' type='application/vnd.ogc.sld+xml' style='sldImportTest'
   href='./styles/sldImportTest.sld'/>
<link rel='stylesheet' type='application/vnd.gnosis.cmss+eccss' style='sldImportTest'
   href='./styles/sldImportTest.cmss'/>
<link rel='stylesheet' type='application/vnd.mapbox.style+json' style='sldImportTest'
   href='./styles/sldImportTest.json'/>

...
```

7.4. MapML encoding

The key capability to develop within this service component was the ability to encode vector features according to the schema defined in the MapML specifications [https://maps4html.github.io/MapML/spec/]. This schema is based on the OGC Simple Features data model. Instead of the default GeoJSON encoding, vector features are encoded as MapML, by specifying the *f=mapml* parameter, or appending a .mapml extension, for the */items* resource or for a tile resource.

7.4.1. Issues encountered

Some issues with the MapML encoding specification were encountered, reported and resolved during this initiative:

- There was an inconsistency with the casing of the tags, which should have been all lowercase (issue 43 [https://github.com/Maps4HTML/MapML/issues/43]);
- Multipolygons should have considered a single valid polygon (also discussed within issue 43 [https://github.com/Maps4HTML/MapML/issues/43#issuecomment-507018476]);
- The <coordinates> tag may be unnecessary and could be omitted from within a <multipoint>. This would make the schema more consistent with the GeoJSON encoding and reduce overhead (issue 44 [https://github.com/Maps4HTML/MapML/issues/44]).

7.4.2. Artificial segments

In order to facilitate re-combining tiled polygonal geometry and avoiding rendering a stroke at tile edges resulting from polygons that cross tile boundaries, a time-proven approach has been to mark the artificial segments introduced by the tiling. See Testbed 13 - Vector Tiles ER [http://docs.opengeospatial.org/per/17-041.html] for a detailed discussion of the topic. Two opposite approaches were proposed and implemented for specifying those artificial segments in the MapML encoding.

Interspersed within coordinates

The approach currently implemented marks the artificial segments directly within the coordinates string, beginning with an opening tag and ending with a closing tag after the last vertex, from which segments should start to be drawn again. This approach more readily conveys to a human when looking at the MapML encoding where the artificial segments begin and end than using numeric indices. This approach can be argued to be more in line with the MapML concepts.

Artificial segments marked within coordinates

```
<geometry>
   <multipolygon>
      <polygon>
         <coordinates>-75.2879795367149 44.850602568211 -75.1641182933529
44.9017607809925
         -75.1246337667094 44.9295287035342 -75.0033046832082 44.9618030122688
         -74.9788414438748 44.9600862937191 -75.0098711316609 44.9434341237869
         -75.1222303607398 44.8970398049808 -75.3122711041934 44.8312465665629
         -75.2879795367149 44.850602568211</coordinates>
      </polygon>
      <polygon>
         <coordinates>-75.0044205502655 44.9823177989379 -74.9599575398279
44.9924035204175
         -75.0033046832082 44.9618030122688 -75.0044205502655
44.9823177989379</coordinates>
      </polygon>
      <polygon>
         <coordinates>-74.9483696896173 44.9460521195752 -74.9788414438748
44.9600862937191
         -74.9599575398279 44.9924035204175 <span class='noline'>-74.8956235121773 45
         -74.8597870124519 45</span> -74.8812889122871 44.9769959714338
         -74.9483696896173 44.9460521195752</coordinates>
      </polygon>
      <polygon>
         <coordinates>-74.9599575398279 44.9924035204175
         <span class='noline'>-74.9609017350302 45 -74.928713262223 45</span>
         -74.9599575398279 44.9924035204175 <span class='noline'>-74.926052348471 45
         -74.9117177485809 45</span> -74.9599575398279 44.9924035204175</coordinates>
      </polygon>
   </multipolygon>
</geometry>
```

At the end of the coordinates

The approach initially proposed and favored by Ecere was to specify all artificial segments using a tag after the coordinates string (right before the </coordinates> closing tag). A <noline/> tag with 'from' and 'to' parameters, with values representing the vertex index within the coordinate string where the artificial segments begin and end.

This has the advantage of keeping the coordinates string intact, so that a simpler or unaware parser that does not expect tags to be present within this coordinates string, is not affected by the existence of such a <noline/> tag. This approach may be slightly simpler and efficient to implement from both a parser and writer perspective, and may also result in fewer characters due to being a single tag (depending on the length of the segments and the number of digits to specify indices).

An alternative view posited that since MapML is intended to be processed by browsers, which provide similar functionality for human-readable text with markup, this might not be a significant limitation. For example, browsers provide not only internal code infrastructure to manage this

design, but also document / node / element APIs to client script code in support of that. If feature geometries were directly supported by browser infrastructure code, even more relevant script APIs could be supplied, making in-browser geospatial Web application development even simpler. The advantage of ... is that it is inherently compatible with the CSS cascade, implemented by browsers. A second reason is that it separates presentation (style) from content. A <noline> tag is inherently presentation-oriented, whereas a can have the style characteristics as specified by the CSS rule that targets it, including not drawing it or enhancing it in some other way.

A counter-argument was made that <noline> is not necessarily presentation-oriented, and perhaps the name of the tag could be changed to better reflect that. The <noline> tag is still strictly data, marking the segment as artificially introduced by tiling. This is a property of the data rather than a presentation attribute. It was also noted that if MapML is used as an encoding, a non-browser client wanting to support MapML would also need to parse and display MapML content.

Artificial segments marked at the end of coordinates

```
<geometry>
   <multipolygon>
      <polygon>
         <coordinates>-75.2879795367149 44.850602568211 -75.1641182933529
44.9017607809925
         -75.1246337667094 44.9295287035342 -75.0033046832082 44.9618030122688
         -74.9788414438748 44.9600862937191 -75.0098711316609 44.9434341237869
         -75.1222303607398 44.8970398049808 -75.3122711041934 44.8312465665629
         -75.2879795367149 44.850602568211</coordinates>
      </polygon>
      <polygon>
         <coordinates>-75.0044205502655 44.9823177989379 -74.9599575398279
44.9924035204175
         -75.0033046832082 44.9618030122688 -75.0044205502655
44.9823177989379</coordinates>
      </polygon>
      <polygon>
         <coordinates>-74.9483696896173 44.9460521195752 -74.9788414438748
44.9600862937191
         -74.9599575398279 44.9924035204175 -74.8956235121773 45
         -74.8597870124519 45 -74.8812889122871 44.9769959714338
         -74.9483696896173 44.9460521195752<noline from='3' to='4'/></coordinates>
      </polygon>
      <polygon>
         <coordinates>-74.9599575398279 44.9924035204175
         -74.9609017350302 45 -74.928713262223 45
         -74.9599575398279 44.9924035204175 -74.926052348471 45
         -74.9117177485809 45 -74.9599575398279 44.9924035204175
         <noline from='1' to='2'/><noline from='4' to='5'/></coordinates>
      </polygon>
   </multipolygon>
</geometry>
```

7.5. OGC API modules

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

7.5.1. Common

Table 1. Ecere OGC API - Common resources

Resource path	Description
/	Landing page.
/api	API description *(NOTE: currently missing)*
/conformance	API conformance *(NOTE: currently missing)*
/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/GRHQ (Québec hydrography network)

http://maps.ecere.com/geoapi/collections/GRHQ?f=mapml (MapML representation, listing commented out sub-collections)

http://maps.ecere.com/geoapi/collections/GRHQ/RH_S (hydrographic network surfaces)

http://maps.ecere.com/geoapi/collections/GRHQ/RH_S?f=mapml (MapML representation)

http://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countries (Natural Earth Countries)

http://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countries?f=mapml (as MapML)

http://maps.ecere.com/geoapi/collections/osm/ottawa/roads (Ottawa Roads from OpenStreetMap)

http://maps.ecere.com/geoapi/collections/osm/ottawa/roads?f=mapml (as MapML)

7.5.2. Vector features

Table 2. 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/GRHQ/RH_S/items (default GeoJSON encoding of the hydrographic network) http://maps.ecere.com/geoapi/collections/GRHQ/RH_S/items.mapml (MapML encoding of the hydrographic network)

NOTE	This example request returns geometry and attributes covering the entirety of a very large dataset. A zoom level lower than the maximum will be used for this request due to the extreme size of the data, but will still require some intense processing from the server side. However, this processing time was greatly reduced by optimizations performed during the initiative, after a performance problem was discovered. The transfer time was also improved by implementing support for deflate and gzip content encoding. However once decompressed, the response for this request is still a large 55 megabytes of MapML encoding, which e.g. browsers will struggle to render as text.

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]);
- *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 not included if specified and not including 'geometry').

By specifying a zero-area box for the location (equal lower left and upper right coordinates, the intersecting bounding box has notably been found useful in a MapML client to query the feature properties:

http://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countries/items?bbox=-75,45,-75,45

Figure 6. ***Only Canada, as a whole feature, returned by the above request***

The geometry and only query specific attributes can also be omitted:

http://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countries/items.mapml?bbox=-75,45,-75,45&properties=name,pop_est

MapML output from the above request, omitting geometry

```
<mapml>
  <head>
    <title>ne_10m_admin_0_countries</title>
    <base href=

"http://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countri
es"/>
    <meta charset="utf-8"/>
    <meta content="text/mapml" http-equiv="Content-Type"/>
  </head>
  <body>
    <extent units="WGS84">
      <input name="zoomLevel" type="zoom" value="2" min="0" max="6"/>
      <input name="minLon" type="location" value="-180" units="gcrs" axis="longitude"
        position="top-left" min="-180" max="180"/>
      <input name="minLat" type="location" value="-90" units="gcrs" axis="latitude"
        position="bottom-right" min="-90" max="90.0002058236639"/>
      <input name="maxLon" type="location" value="180" units="gcrs" axis="longitude"
        position="bottom-right" min="-180" max="180"/>
      <input name="maxLat" type="location" value="90.0002058236639" units="gcrs" axis
="latitude"
        position="top-left" min="-90" max="90.0002058236639"/>

      <link rel='features' type='text/mapml' title='ne_10m_admin_0_countries (as
MapML)'
        tref=
'http://maps.ecere.com/geoapi/collections/NaturalEarth/cultural/ne_10m_admin_0_countri
es/
        items.mapml?tilingScheme=CRS84Quad2L0Tiles&amp;
        zoomLevel={zoomLevel}&amp;clipbox={minLon},{minLat},{maxLon},{maxLat}'/>
    </extent>
    <feature id="fclass.68" class="fclass">
      <properties>
        <table>
          <thead>
            <tr>
              <th role="columnheader" scope="col">Property name</th>
              <th role="columnheader" scope="col">Property value</th>
            </tr>
          </thead>
          <tbody>
            <tr><th scope="row">name</th><td itemprop="name">Canada</td></tr>
            <tr><th scope="row">pop_est</th><td itemprop="pop_est">33487208</td></tr>
          </tbody>
        </table>
      </properties>
    </feature>
  </body>
</mapml>
```

On the other hand, the clipping bounding box is preferable for requesting only a portion of a high resolution very large single feature, such as 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.

7.5.3. (A)RGB Imagery

Table 3. 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/BMNG%202004/items (NASA Blue Marble Next Generation imagery mosaic)

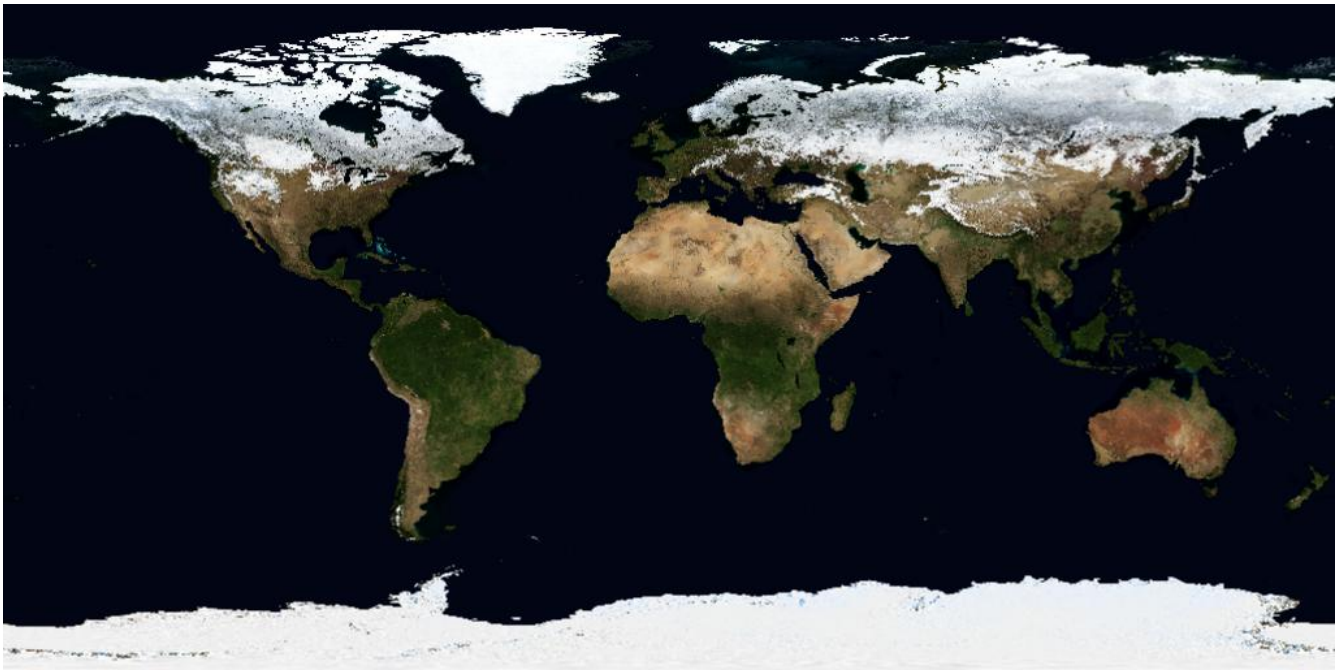

*Figure 7. **NASA Blue Marble Next Generation imagery moasic returned by above request***

7.5.4. Coverage

Table 4. 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/SRTM_ViewFinderPanorama/items (SRTM ViewFinderPanorama elevation data)

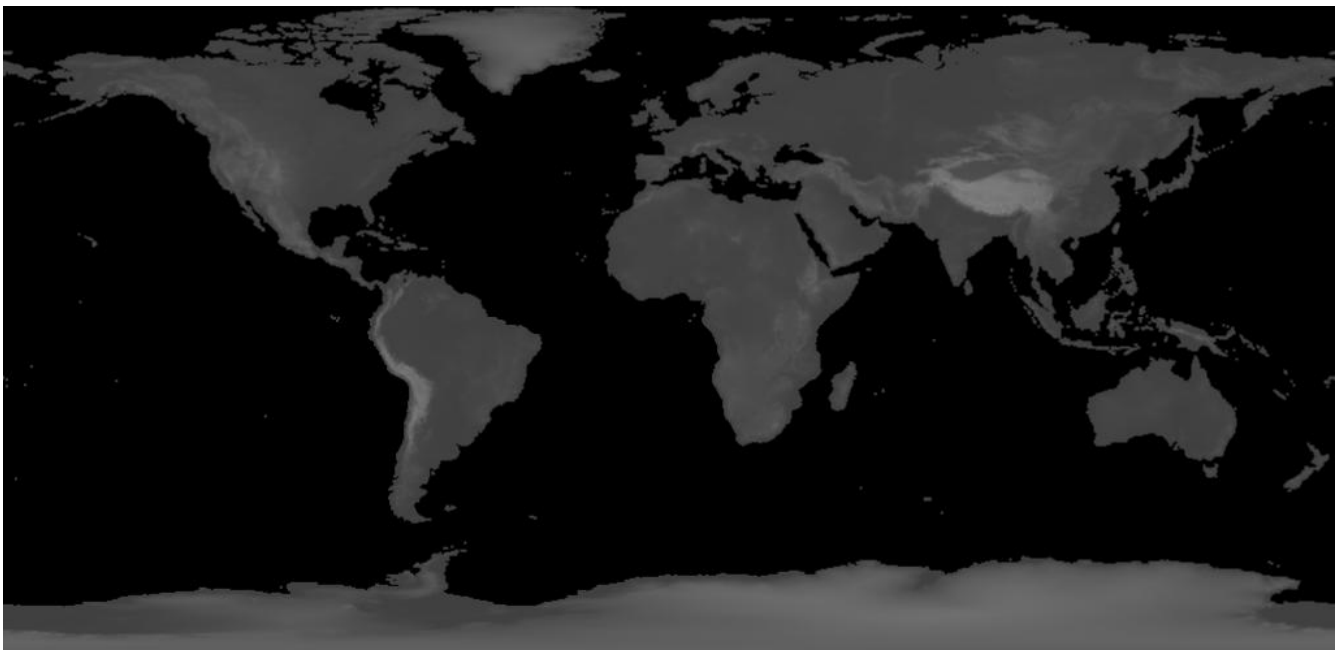

*Figure 8. **ViewFinderPanorama SRTM elevation data returned by above request***

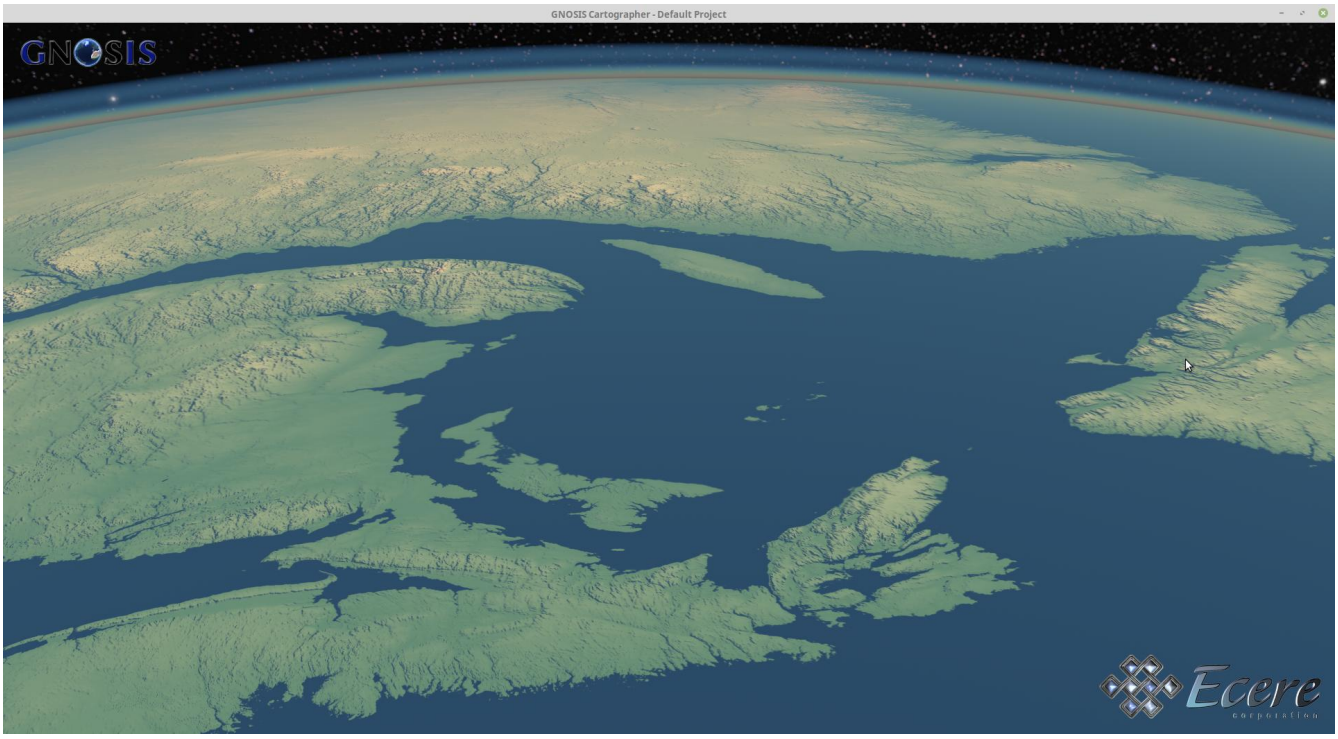

Figure 9. ***Elevation used for building 3D terrain mesh, hill shading and coloring in GNOSIS Cartographer***

7.5.5. Tiles

Table 5. 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

Tiled data 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 tiling schemes supported include:

- 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] (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] (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] (EPSG:4326, same as

GoogleCRS84Quad, but offset by one level to avoid intricacies);

- 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], defined in OGC TMS 1.0 Appendix H.2 [http://docs.opengeospatial.org/is/17-083r2/17-083r2.html#106]);
- 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 was added during the initiative to support the MapML needs, as currently only the EPSG:4326 tiling scheme is officially supported by MapML.

Example direct paths to tiles:

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

osm/ottawa/tiles/CRS84Quad2L0Tiles/14/4053/9493.mvt [http://maps.ecere.com/geoapi/collections/osm/ottawa/tiles/CRS84Quad2L0Tiles/14/4053/9493.mvt] (multi-layer Mapbox Vector Tile, OSM Ottawa) GRHQ/RH_S/tiles/CRS84Quad2L0Tiles/6/16/37.mapml [http://maps.ecere.com/geoapi/collections/GRHQ/RH_S/tiles/CRS84Quad2L0Tiles/6/16/37.mapml] (hydrographic network surfaces tiled vectors, as MapML)

Figure 10. ***Hydrography network vector tile***

SRTM_ViewFinderPanorama/tiles/CRS84Quad2L0Tiles/6/16/37 [http://maps.ecere.com/geoapi/collections/SRTM_ViewFinderPanorama/tiles/CRS84Quad2L0Tiles/6/16/37] (elevation data tile)

Figure 11. ***Elevation data tile***

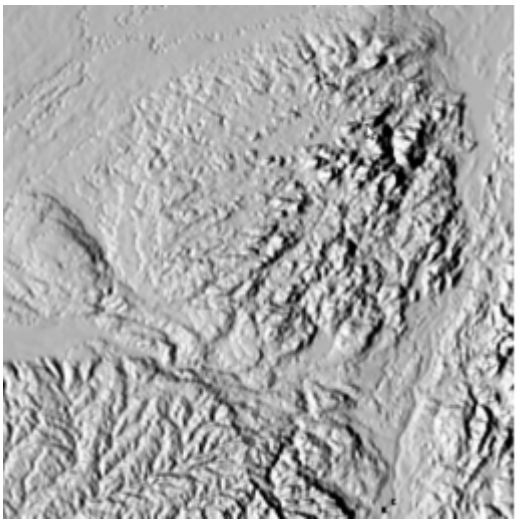

Figure 12. ***Elevation data tile (hillshaded with QGIS)***

BMNG%202004/tiles/CRS84Quad2L0Tiles/6/16/37 [http://maps.ecere.com/geoapi/collections/BMNG%202004/tiles/CRS84Quad2L0Tiles/6/16/37] (imagery tile)

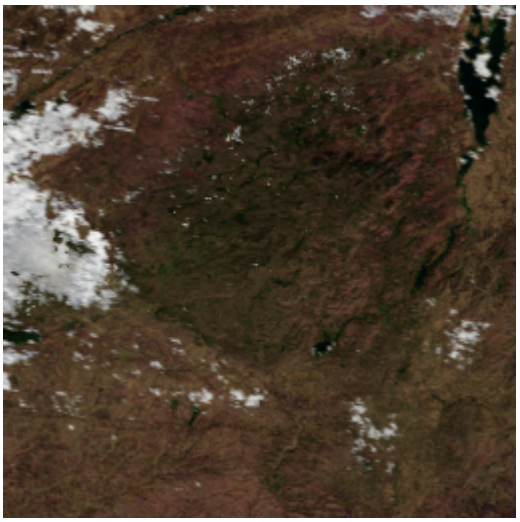

Figure 13. ***Imagery mosaic tile***

7.5.6. Styles

Table 6. Ecere OGC API - Styles resources

Resource path	Description
/styles	The list of all styles *(NOTE: missing as of November 25th, 2019)*
/styles/{styleId}	The style data for a given style *(NOTE: missing as of November 25th, 2019)*
/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. This functionality is experimental as numerous challenges are posed by major differences in styling approach from one styling language and renderer to another. This workflow is being improved to produce better conversion results.

The association between the styles and the data layers is done by storing style sheets with the source data store itself, or through the concept of a *stylable layer set*. This 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/

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] (Default Natural Earth style, default CMSS encoding)

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] (Default Natural Earth style, Mapbox GL encoding)

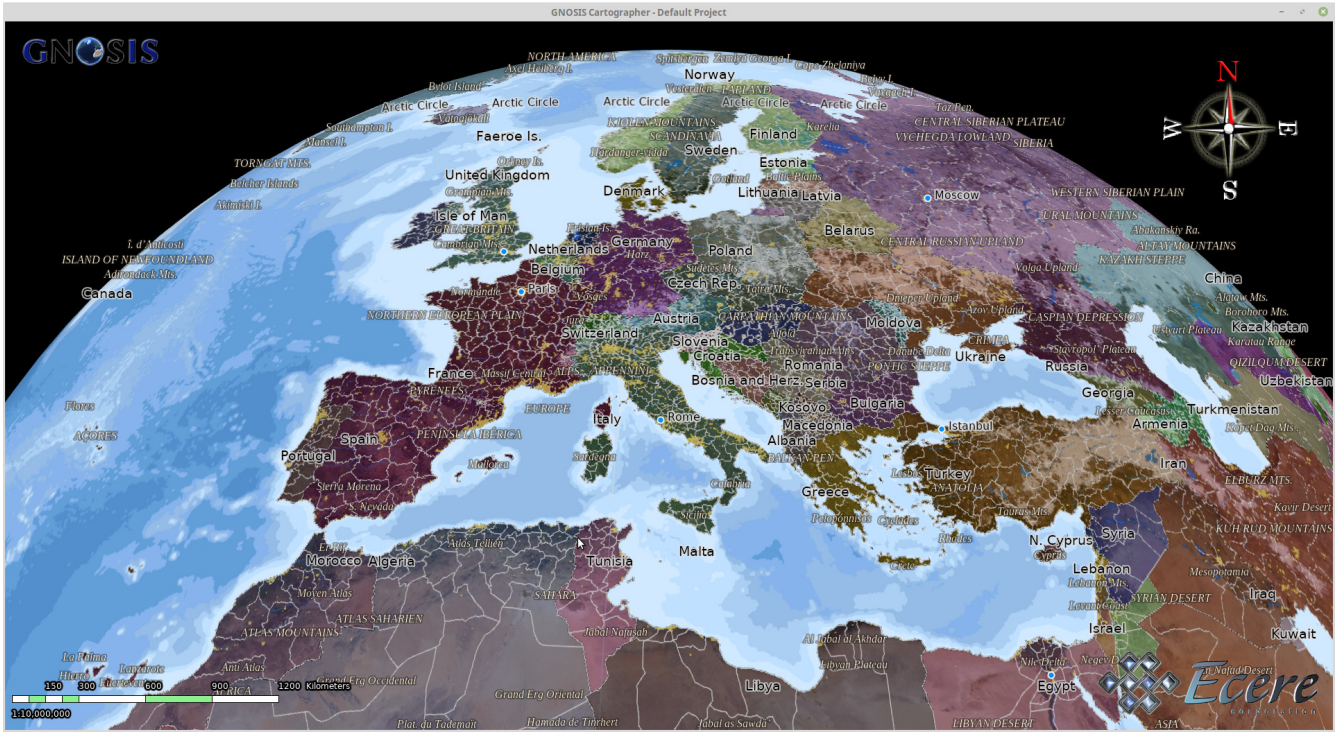

Figure 14. ***Default styles for Natural Earth***

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] (Default style for OpenStreetMap, default CMSS encoding)

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] (Default style for OpenStreetMap, Mapbox GL encoding)

Chapter 8. Quebec model MapML Client

The Quebec model MapML Client (D106) displays MapML documents served by an implementation of the *OGC API - Features* standard. The final geospatial results of the ML model are provided in a vector format. The data at its different stages are displayed using MapML served by *OGC API - Features*.

A scenario includes a single WPS instance that serves as an interface to the vectorization model. The service/model needs to connect to various OWS instances at the backend. The WFS instance serves the results of the vectorization process as MapML. A dedicated client application interacts with the service instances and displays the MapML data.

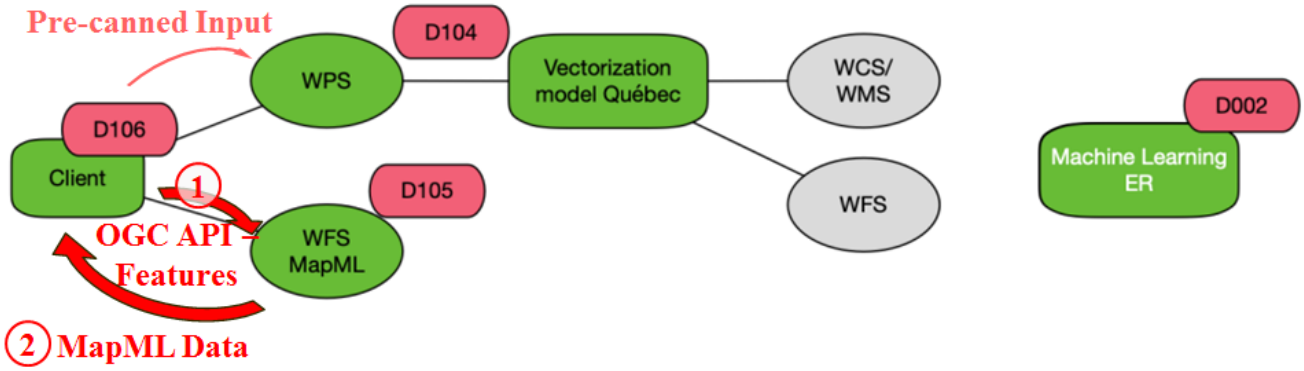

Figure 15. Quebec scenario work items

The scenario begins when a client receives a notification from a WPS instance informing the client that it is ready to provide a service. After that, the client requests features from an *OGC API - Features* instance, and receives a MapML document which describes a final geospatial result of the ML model as a vector format.

8.1. Component Summary

The Quebec model MapML Client (D106) is a web application which is executable on a web browser. The D106 application logic is in the client-side of the web application, and the logic runs on a browser engine inside a web browser.

The D106 client supports the following features:

- Showing vector features (MapML and GeoJSON format) served by *OGC API - Features*;
- Supporting spatial and temporal constraints;
- Executing generic operations served by WPS 2.0;
- Showing single and tile images served by Web Map Service (WMS), WMTS, ArcGIS REST, and XYZ services;
- Supporting a pixel picker tool.

Figure 16 shows the top-level software architecture of the D106 client.

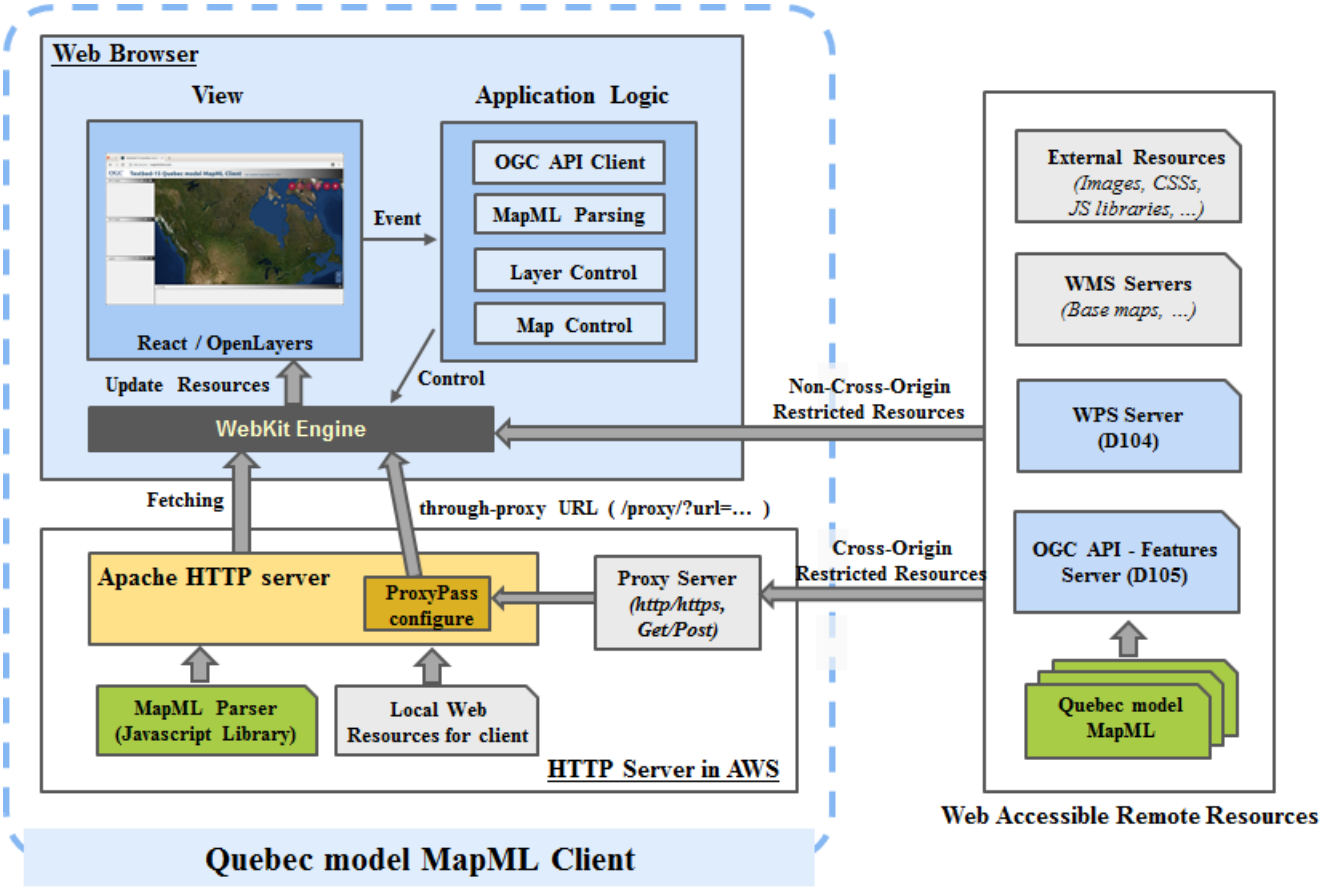

Figure 16. Top-level Software Structure

The server-side components consisted of a web server, a proxy server, and application resources such as HTML files, scripts, images, and so on. A web browser loaded all the D106 application relative resources, and then a browser engine (e.g., *WebKit* or *Blink*) executed them as client-side components.

The D106 client sometimes needed to fetch cross-origin XML or JSON content, such as a response of GetCapabilities of WPS or WMS/WMTS, Collections list of *OGC API - Features*, MapML documents, etc. Cross-origin requests are not permitted under a *Cross-origin resource sharing* (CORS) same-origin security policy in the browser.

For purposes of the testbed, a proxy mechanism was built to work around CORS prohibitions. Each cross-origin URL was encapsulated in a new same-origin URL, and then sent to a proxy server for bypass.

The longer-term recommendation is to use *CORS headers* to designate resources as being shared. When sharing is not desired, the browser would enforce CORS same-origin security policy.

8.2. Component Design

The D106 deliverable is a web application, and its software structure is based on *Model-view-controller* (MVC). Each client-side component has its own model, view, and controller, and the components only share a few variables and methods with each other.

In the D106 deliverable, the client has 5 user interface components and 1 MapML component:

- **Map Area**: Manipulates the map area supported by OpenLayers.
- **WFS Client**: Fetches collections and their items served by *OGC API - Features*.
- **Layers**: Manages vector and image layers.
- **WPS Client**: Fetching and executing operations served by WPS 2.0.
- **Toolbar**: Supports spatial and temporal constraints, additional tools, and settings menu.
- **MapML**: Manipulates MapML document:
  - **Parser**: Parses MapML document and then produces a MapML document object model;
  - **Document Object**: MapML document object model;
  - **OpenLayers Front-end**: Creates a layer-objects array of OpenLayers from a MapML document object model.

Figure 17 shows a sequence diagram for the client fetching a collections list served by an *OGC API – Features* instance, which writes and returns the item as a MapML document.

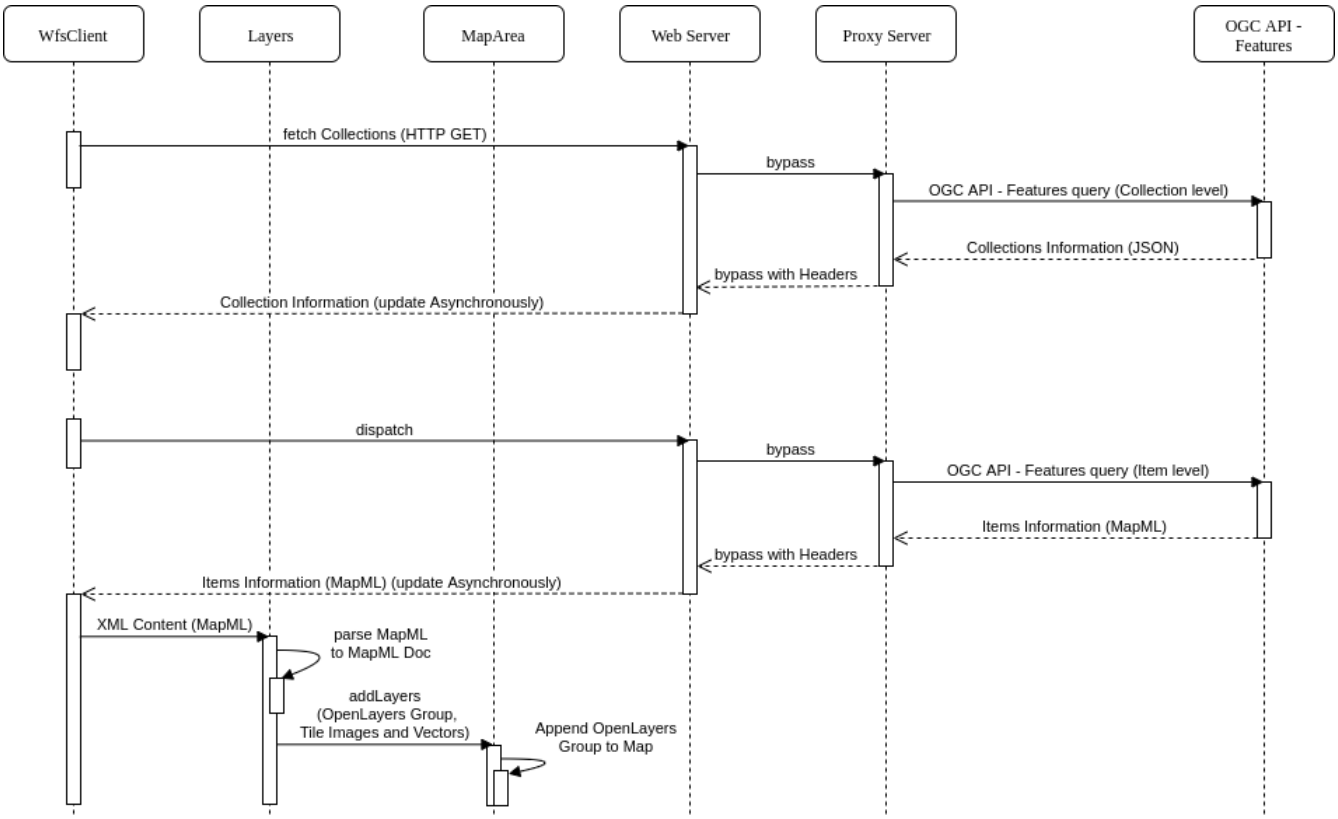

Figure 17. Fetching MapML document from OGC API - Features

8.2.1. Step 1. Acquiring a Collections List

An end-user interaction triggers a request to fetch a Collection from an *OGC API - Features* service endpoint. The *WFS Client* sends the request via *Ajax* to a local web server, which uses the Proxy Server to bypass the CORS restriction. (As described above, it is recommended that a longer-term approach should use CORS headers instead of a proxy.) The Collection information asynchronously returns the result to the *WFS Client*.

8.2.2. Step 2. Acquiring a MapML document

The *WFS Client* has a user interface for setting *OGC API - Features* parameters, such as selecting a collection, appending spatial and temporal constraints, setting zoom level or output format, and so on. The end-user can request features information by clicking the *AddLayer* button in the user interface. The *WFS Client* sends a request for acquiring item features, and the *WFS Client* receives a MapML document containing the desired item features.

8.2.3. Step 3. Showing features from the MapML document

Now the *WFS Client* has the desired MapML document containing multiple <*feature*> elements. The *WFS Client* component sends this MapML content to the *Layers* component for adding the layer into the map area. The *Layers* component parses the MapML text content to a *MapML Document Object* model by using the *MapML Parser*. The view of D106 is based on an OpenLayers map library, so the *MapML Document Object* is converted using the *MapML OpenLayers Front-end*. After conversion, the generated layer object is appended as a layer of the OpenLayers map view.

8.3. Implementation Approach

To show a MapML document in the map area, the D106 component parses the MapML document by using the *MapML Parser* and then converts it to an OpenLayers layer object using the *MapML OpenLayers Front-end*. Specifically, the parser converts all <*feature*> elements to a single GeoJSON object for ready showing in the map area.

A suggestion was also made to consider converting the MapML into canvas commands or Scalable Vector Graphics (SVG) objects directly, especially if there is a risk of potentially exhausting available memory.

8.3.1. MapML Parser

A raw MapML document is text content, and the *MapML Parser* converts the plain text of MapML documents to a *MapML Document Object*. Figure 18 shows how each statement in a MapML document becomes a part of a *MapML Document Object*. The <*head*> element becomes a *head* key, which is a top key of the object. The <*extent*> element becomes an *extent* key, which is a sibling of the *head* key. Each <*feature*> element becomes an element of a *features* array in a *FeatureCollection* type of *geojson* key. The parser packs all <*feature*> elements into a single FeatureCollection of a GeoJSON object.

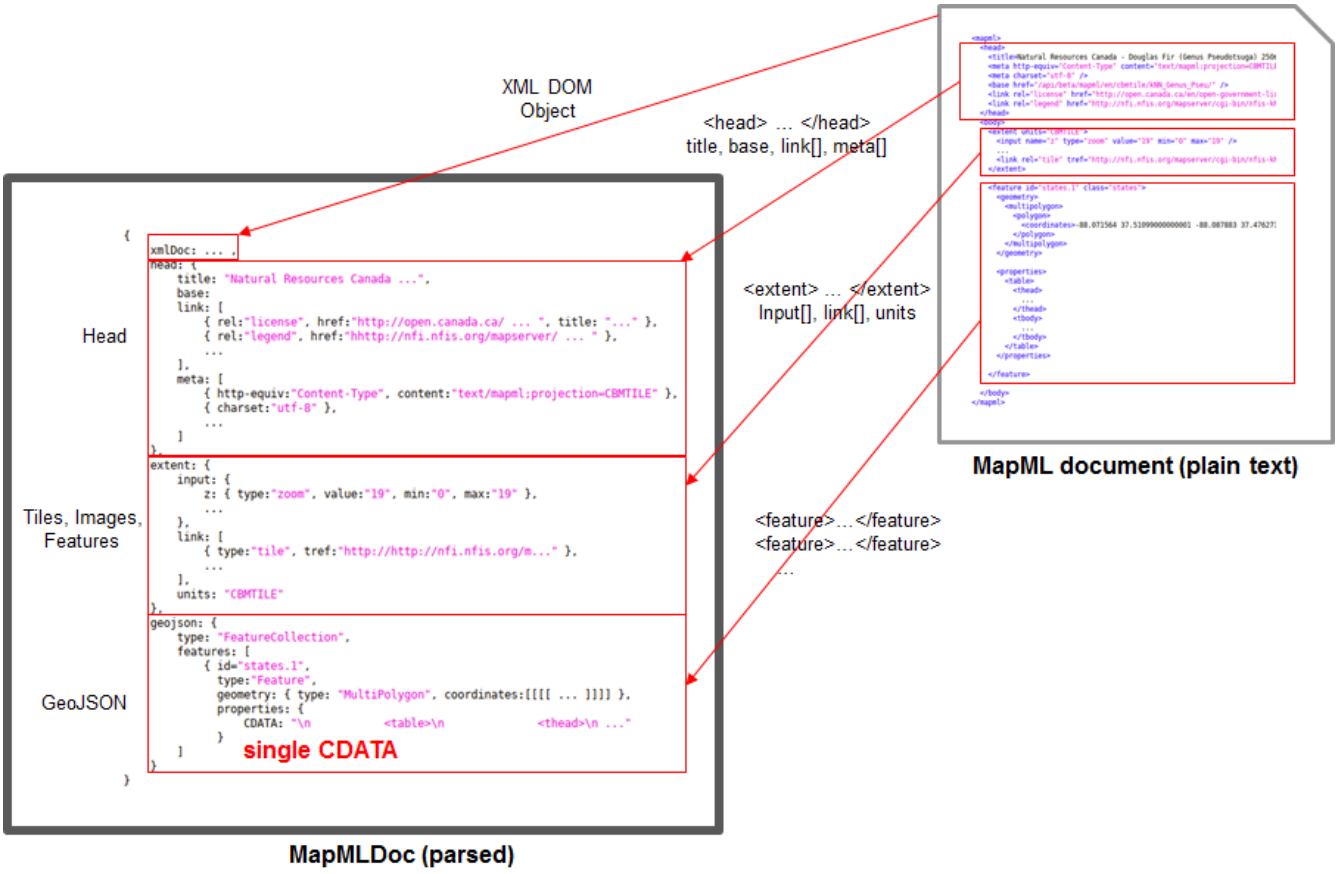

Figure 18. MapML document parsing

Unlike the GeoJSON specification, the <*properties*> element of MapML is a free HTML format, so the parser converts it as a single property named "CDATA".

An alternative view was posited that the idea of MapML is to be compatible with the HTML parser, so that MapML becomes part of HTML (obviating any need to convert to JSON first). For now, MapML services can be relied upon to be mostly compatible with MicroXML rules, which enables use of the browser's built-in XML parser. The HTML parser could also be used if its heuristics are known, enabling manipulation of the output DOM directly using browser-provided DOM APIs. Under this view, MapML is an HTML encoding of geospatial information, which is where the semantics of geospatial information are currently absent. MapML fills this need.

8.3.2. OpenLayers Front-end

A *MapML Document Object* has an extent array and a GeoJSON object. In most cases, the item features served by *OGC API - Features* are described in the <*feature*> elements in the MapML response. Therefore, the most important part of the D106 deliverable as an *OGC API - Features* Client is converting from <*feature*> elements of MapML to a single GeoJSON object.

To become a MapML viewer beyond being simply an *OGC API - Features* client, the D106 component had to support <*extent*> elements as describing how a layer of map area is retrieved. Each <*link*> in the <extent> element can describe raster or vector layers, and each of them will become an (internal) layer. Figure 19 shows not only how a GeoJSON object becomes a vector layer in OpenLayers, and also how each <*link*> in an extent will become a proper layer in OpenLayers in its own right.

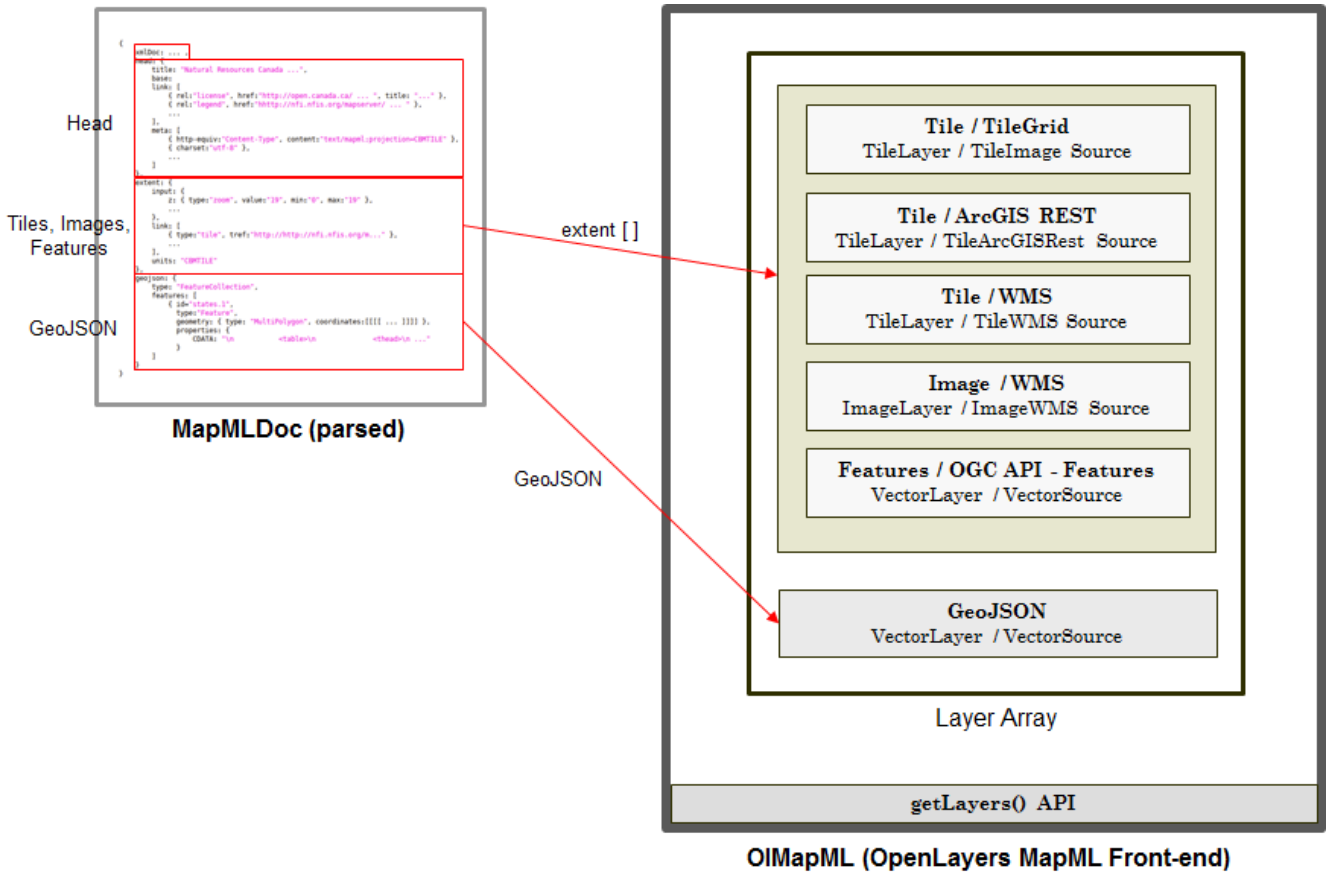

Figure 19. OpenLayers front-end for MapML

The name *OlMapML* is a Node.js package name that supports APIs for getting a layers array of OpenLayers. The following example shows how to use the *OlMapML* Node.js package.

```
import MapML from './MapML.js';
import OlMapML from './OlMapML.js';
...
let content = "<mapml> ... </mapml>";

let mapml = new MapML();
let mapmlDoc = mapml.parse( content );   // returns MapMLDoc
let geojson = mapmlDoc.getGeoJSON();     // GeoJSON object
// or
let olMapML = new OlMapML( mapmlDoc, { projection: 'EPSG:3857' } );   //
OpenLayers front-end
let layers = olMapML.getLayers();        // OpenLayers Layer array (both <extent>
and <feature>s)
...
```

8.4. Component Implementation

The D106 deliverable D106 was a web application based on a React [https://reactjs.org/] user-interface development framework. The deliverable was deployed on a Linux virtual machine supported by the *Amazon Elastic Compute Cloud* (Amazon EC2) web service. The reader is invited to visit an

instance of the D106 deliverable [http://mapmlclient.com].

8.4.1. Runtime Environment (for the D106 web application)

- **Processor**: Intel Xeon CPU E5-2676 v3 @ 2.40GHz
- **Memory**: 1GB
- **Storage**: 10GB SSD
- **OS**: Linux Ubuntu 16.04 LTS
- **Web Server**: Apache HTTP Server 2.4.18
- **Proxy Runtime**: Python 2.7.12

8.4.2. User Interface

Figure 20 shows the top-level screen view on a modern web browser.

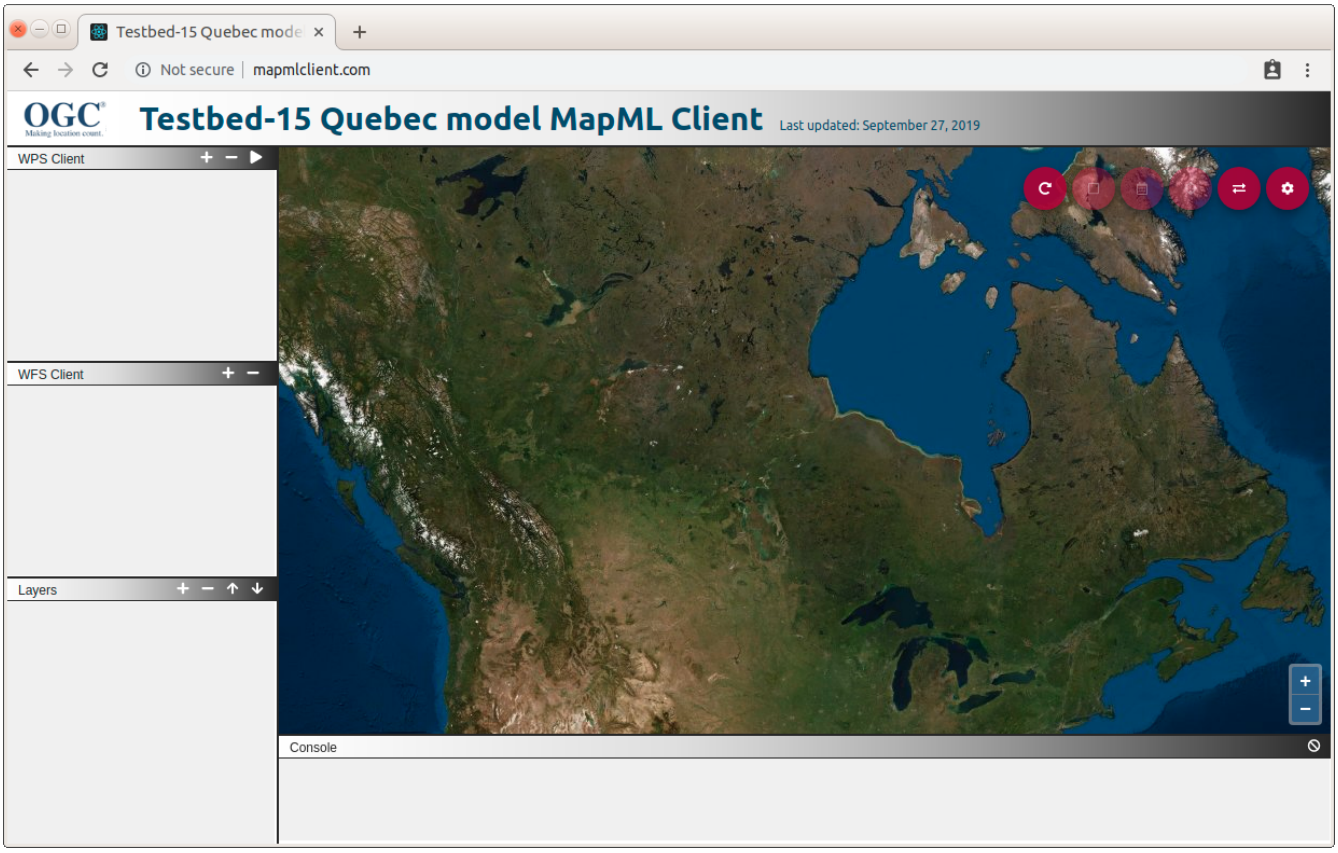

Figure 20. Top-level Main Screen Capture

There are six sections in the main view. Two Client (WPS, WFS) sections and one Layers section are in the left area, one console section is in the bottom area, one toolbar section is in the top right side, and one map view section is in center area.

- **Map Area**: OpenLayers map area (with a basemap, changeable on Settings).
- **WPS client**: Generic WPS 2.0 client.
- **WFS client**: Generic *OGC API - Features* client, especially supports the MapML item format.
- **Layers**: Layer control, adding a WMS, GeoJSON, or MapML layer directly from a URL or a local

file.

- **Console**: Displays logs, especially when fetching URLs internally.
- **Toolbar**: Reloading, Bounding box selector, Date time selector, Pixel picker, Translating tool, and Settings.

Figure 21 shows how to add a new *OGC API - Features* service endpoint.

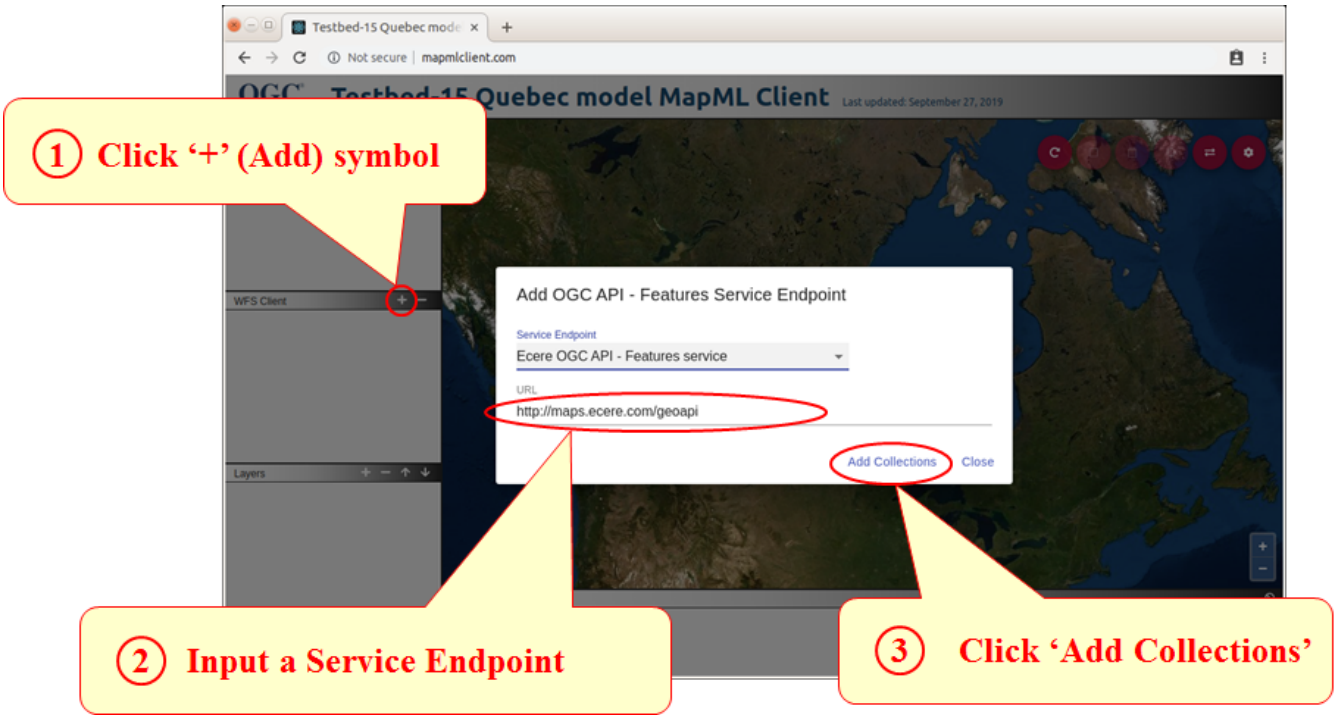

Figure 21. Input OGC API - Features Service Endpoint

Figure 22 shows an *OGC API - Features* item parameter setting dialog box for fetching feature items.

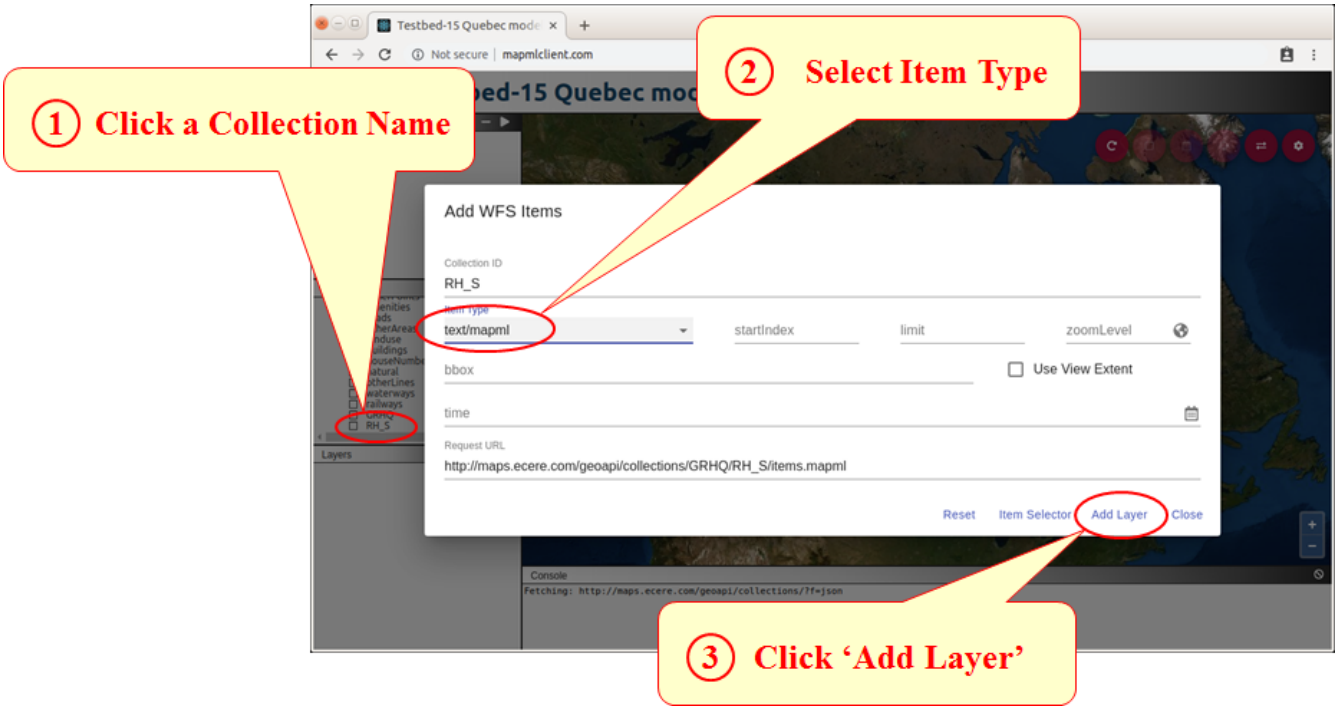

Figure 22. Add Feature Items as a Map Layer

Figure 23 shows that vector features described in a MapML document display on the map area as a

vector layer. An end-user can select any feature in the map area, and its property information is shown in a dialog box.

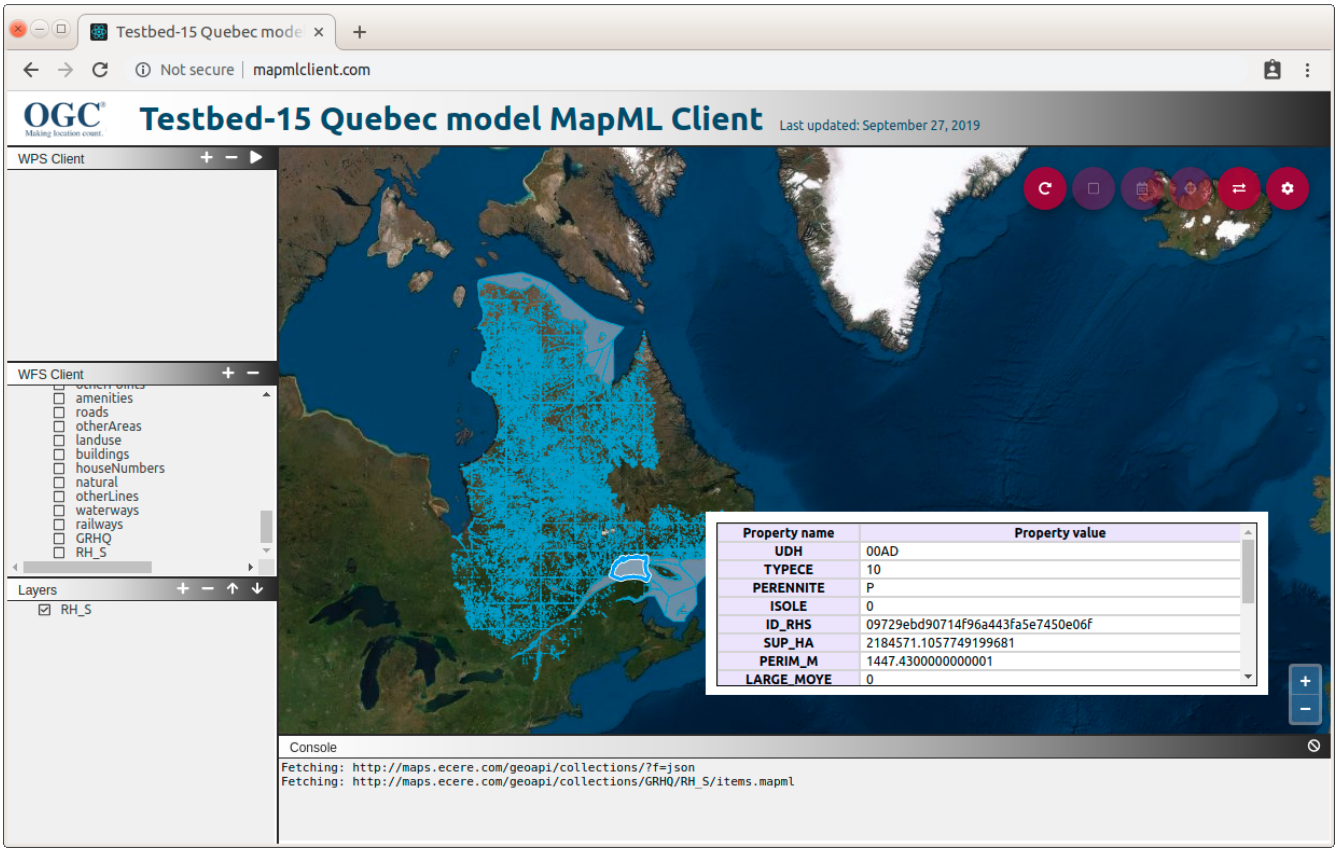

Figure 23. Display Features in a MapML document

8.4.3. Future Considerations

Although the Quebec model MapML Client D016 deliverable design was based on using vector features described in a MapML document, MapML has potential benefits as a metadata document in describing an *OGC API - Features* implementation and a selection of features from a data source. The following are advantages of MapML as a service or client implementation of the *OGC API - Features - Part 1: Core* standard.

MapML extension for OpenLayers

This deliverable implementation uses OpenLayers as a front-end map manipulation library, but it is not a formal extension of the library. To become a formal extension of OpenLayers, the MapML front-end should be implemented based on the abstract OpenLayers APIs, such as *format*, *source*, and *layer* in the library hierarchy. The MapML parser of this deliverable can become a *format* class, the fetching WFS logic can become a *source* class, and the front-end logic could become a *layer* class.

Recursive features link

This deliverable implementation uses *feature* elements to show vector features data. MapML might be capable of recursively including vector features from different service providers, realizable if a client supported inner features links recursively.

Vector Tiles

This deliverable implementation partially supports vector tiles feature data, if a restricted map projection type and tiling scheme are selected. MapML documents can include multiple vector tiles features links, and these links are supported without any restricted conditions.

MapML for a metadata document

MapML is suitable for representing not only a selection of features from a data source, but may also be useful as a metadata document describing a collection of information. It could provide links into the collection using map semantics (bounding boxes, tiles, images, features etc.).

In future work, it might be interesting to see how a crawler could use a MapML representation of a collection document to index a collection. Not only could it use information in the <head> element such as Dublin Core metadata, but it could actually use the affordances to link into the collection and crawl the feature information in the collection, in much the same way as HTML allows crawlers to follow links they find to other HTML resources. Use of the crawled resources via a search engine could constitute “Discovery” use case resolution.

Appendix A: Revision History

Table 7. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
August 22, 2019	Scott Serich	.1	all	initial IER draft
October 23, 2019	NRCan	.2	all	sponsor edits
October 28, 2019	Gil Heo	.3	client content	client content edits
October 30, 2019	Je ro me Jacovella-St-Louis	.4	service content	service content edits
October 30, 2019	Scott Serich	.5	front and back sections	clean up
November 6, 2019	Scott Serich	.6	service and client content	incorporate sponsor feedback
November 10, 2019	Scott Serich	.7	service content	rework link relations and styling considerations
November 11, 2019	Scott Serich	.8	service content	rework "MapML representation of OGC API collection description"
January 6, 2020	Scott Serich	.9	all	incorporate recommended changes from Carl and Gobe